RC 20041 (88622) 4/24/95
Computer Science

IBM Research Report

Profile-driven Sampled Trace Generation

Pradeep K. Dubey and Ravi Nair

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Abstract

Trace driven simulation is a very commonly used technique for evaluating different machine design options. For faster simulations, one often needs to significantly reduce the size of the trace input. This paper proposes a new technique, referred to as, *profile-driven sampling* for obtaining a representative reduced size trace. This technique results in a reduced trace that is more *representative* of the original full trace, than those obtained without the profile feedback. Predicted performance inaccuracies, using previously published periodic time-based sampling technique, are reduced on an average by over 50 percent, using the profile driven sampling technique. Also, the *periodic window* based trace sampling techniques can be quite sensitive to sampling parameters such as, the sample size and the inter-sample interval. Such sensitivities are not applicable to the proposed *profile driven* sampling technique. This significantly simplifies the trace generation process.

1 Introduction

Trace driven simulation is perhaps the most widely used technique for evaluating different machine design options. The accuracy of such simulations in predicting actual machine performance for different benchmarks, is primarily determined by the accuracy of the machine model, and the accuracy with which the trace inputs represent the intended benchmarks. Accuracy on both these counts is bounded by that obtainable via modeling every known detail of the machine and by using the complete benchmark traces as input. This paper is concerned with the accuracy of trace inputs. As the industry standard benchmarks increase in size, from several thousands instruction long *dhrystone* to almost 100 billion instruction long SPEC92 benchmark suite, simulations using complete trace inputs have become prohibitively expensive. For example, a typical simulator of current target microprocessors takes on an average more than 20000 instructions per simulated instruction. Therefore, it would take more than 200 days to complete full traces of SPEC92 suite, on a host machine with an average performance of 100 MIPS. As a result, to be able to study several successive design tradeoffs per day, a 1000 fold speed improvement becomes almost a necessity. Generally speaking, there are three options available: a) higher-level (less detailed) target machine model requiring fewer simulation instructions per simulated instruction, b) changing the host machine environment, e.g., by running parallel simulations, to improve the effective simulation speed, and c) reducing the input trace size. This paper is focussed on the last option, namely, trace size reduction. Thus, the other two options should be considered orthogonal to the discussion below.

The paper proposes a new technique for obtaining a representative reduced size trace. The technique, referred to as *profile driven trace generation* is based on a two-step approach. In the first step, the complete benchmark is executed in the trace-generation environment to generate its dynamic profile. In the second step, the benchmark is executed again for generating a reduced trace, based on the profile information. The trace size reduction factor is specified as a parameter during the second step. This technique results in a reduced trace that is more *representative* of the original full trace, than those obtained without the profile feedback. Predicted performance inaccuracies, using previously published periodic time-based sampling technique, are reduced on an average by over 50 percent, using the profile driven sampling technique proposed in this paper. Also, the *periodic window* based previously published trace sampling techniques [7, 2] can be quite sensitive to sampling parameters such as the sample size and the inter-sample interval. Such sensitivities are not applicable to the proposed profile driven sampling technique. This can significantly simplify the trace generation process, as the only parameter that needs to be selected (or optimized) under the proposed scheme is the *trace size reduction factor*.

The following section discusses some of the previous techniques for generating reduced length traces. Section 3 describes the *profile driven trace generation* technique. Section 4 offers a validation of this approach. The final section provides concluding remarks.

2 Background

Trace-driven simulation is probably the most popular method for evaluating processor performance. This includes simulations of inter-pipeline and intra-pipeline dependencies, and memory system latencies. Several approaches have been used in the past for trace generation. These can be broadly classified into those that are *software-based* and those that are *hardware-based.* This paper is solely concerned with the software-based approach. It should be noted that the issue of input trace size reduction, discussed in this paper, is distinct and orthogonal to that of reducing the *trace file size.* The latter is concerned with the storage and efficient generation of a large program trace [8], whereas, the former is concerned with simulating fewer instructions. Also note that program *tracing* is different from program *profiling. Tracing* is aimed at generating a listing of the program instructions (or memory references, or both), in the proper run-time order. However, *profiling* is a much quicker process which simply records the total count of the number of times different program instructions execute. A reduced size trace, which can be any subset of the instructions [1] in the original full size trace, differs from the full trace in following two aspects:

- There are instructions in the full trace which are simply not present in the short trace,
- The instructions present in the short trace may be in a *different context* than in the full trace. This is a concern because an instruction execution is almost always dependent on that of its predecessors.

These differences have important implications in evaluating the *representativeness* of the short trace. For example, the former implies that the instruction mix of the small trace may be different from that of the full trace, and the latter implies different basic block frequencies, or different behavior of history-sensitive buffers, such as, branch prediction tables, write buffers, completion buffers, and caches. The context-difference concern also implies that whereas, a completely random sampling may offer the same instruction mix between the short and full traces, it may be a very poor representative of the full trace in the presence of inter-instruction dependencies. Therefore, almost all trace size reduction techniques use one or more time contiguous *samples* of instructions. The following sampling techniques have been reported:

- *One large sample*: A single sample of varying sizes (from less than 1 million to 10 million instructions) has been used to reduce full size traces (of few billion instructions). This early approach to trace size reduction was later modified to skip the first few (several million) instructions to obtain a more representative sample. A significant number of performance projections of a wide variety of processor models [1, 3, 5, 10] are based on such sampling.
- *Multiple periodic samples*: This approach is based on stitching together several time contiguous samples, taken at a fixed interval [7, 2]. In other words, after tracing a fixed size sample, a fixed number of instructions is skipped. The inter-sample interval is normally adjusted to cover the full trace.

[1] Although a trace used for memory system evaluation may simply consist of memory references, we do not make this distinction, and individual elements of a trace are always referred to as *instructions.*

- *Multiple random samples*: Instead of periodically distributing the samples over the full trace length, this approach distributes the samples randomly over the full trace, by randomly varying the inter-sample interval [9].

It is possible for each of the three approaches mentioned above to get a short trace which is not a good representative of the full trace. For example, consider a full trace which spends 90 percent of its time in a highly parallelizable code region and the remaining 10 percent in a serial code region. It is quite possible to create a short trace, say 1/1000 in length of the full trace, using periodic sampling technique, such that it mostly consists of the serial region of the code. The random sampling technique does not guarantee the original mix of instructions. In fact, quite often short traces created through these techniques have to go through an *initial validation* phase, in which the short trace is compared against the full trace for frequencies such as, instruction frequencies, basic block frequencies etc. Lauterbach [9] offers such validation for the random sampling technique, using SPEC92 benchmarks.
Final validation of the representativeness of any short trace is often a very time consuming process, as it involves comparing the simulated performances of a given micro-architecture using the full trace and short trace inputs. As has been discussed before, the full-trace simulations can be very long (few weeks for several SPEC92 benchmarks) and the primary aim of short traces is to replace these long simulation runs with shorter ones.

3 Profile Driven Trace Sampling

Note that the classification of sampling techniques presented in the previous section assumes *time-based* sampling. Alternately, a sampling method can also be based on some other aspect of the system being simulated. For example, simulation of the branch unit of a processor may use a trace input based on sampling the *branch* instructions only. Kessler et. al. [6] propose a *set sampling* technique for cache simulation, which samples references to a specific cache set. The *profile driven sampling* technique introduced in this paper is also a non-time-based sampling technique, and it is based on sampling references to a basic block.
The profile information for the benchmarks was generated using a program called *xprof* [11]. This program instruments the executable to produce counts corresponding to the frequency of execution of basic blocks. The trace information is produced by a program called *xtrace* which instruments the executable to produce the actual trace of instructions and memory references. (While *xtrace* could have been used also to generate the profile information, a specialized profile generator speeds up the process by an order of magnitude.)
The output from *xtrace* is filtered through a *sampler* which takes as inputs the frequency generated by *xprof* along with a parameter which indicates the desired scaling. This enables calculation of the number of occurrences of each basic block that should be output by the *sampler*. Different strategies may be used in determining which occurrences of each basic block should be selected. The simplest scheme to choose n occurrences of a basic block, and the one for which results are reported here, is to choose the first n occurrences of the block.

4 Validation

The initial and final validation steps mentioned in Section 2, often have to be repeated with different sampling parameters, until a reasonable match between the sampled and full traces is established with reasonable confidence. In the absence of any known theory which can predict the optimal set of sampling parameters for a given trace across a set of micro-architectural tradeoffs, there is unfortunately no simple substitute for this iterative process. However, this iterative process can be improved. The *profile driven sampling* technique described in the previous section, accomplishes this by obviating the need for many initial validation phase checks identified above. It also reduces the number of sampling parameters to a minimum of one, namely, the trace size reduction factor. This leads to a significant simplification of the trace generation process overall.
In the periodic window tracing, sampled trace statistics such as, frequency of different instruction types and frequency of different basic block sizes, need to be validated against those of the full trace, for quick assessment of the representativeness of the sampled trace [9], as there is no guarantee that these statistics would match with full trace statistics. However, in the case of profile-driven sampling the trace generation process explained above guarantees that statistics such as, instruction mix distribution, and basic-block size distribution, would match those of the full trace, thus bypassing the need of any initial validation. Details of the final validation, via micro-architecture simulation are provided below.

4.1 Benchmarks

Table 1 lists the benchmarks used for validation. The benchmarks were chosen from the SPEC92 benchmark suite. In order to improve the full trace simulation time, the chosen benchmarks either have their source modified, or were run with a subset of the reference input. This however does not impact the original goal of around 1000 fold reduction in the input trace size. In fact, the ratio of the combined full trace sizes to the combined sampled trace sizes is almost 2800. The length of the sampled trace was 100,000 instructions for all the benchmarks, except *Gcc*. A smaller sampled trace size of 40,000 instructions was chosen for *Gcc* because for the selected input files, the full trace length of *Gcc* was considerably smaller than the rest of the benchmarks.

4.2 Processor Model

For the purpose of final validation of the representativeness of the traces, the traces are used as input to a cycle-by-cycle simulator of a typical superscalar processor. The hypothetical processor modeled by this simulator contains four execution units: two integer units, one floating point unit, and one branch unit. A *direct* conditional branch refers to a typical conditional branch to a target address or offset encoded in the branch instruction, whereas, an *indirect* conditional branch refers to a conditional branch whose target address is contained in another register. Direct conditional branches are predicted *taken* or *not-taken* using a 2-bit prediction scheme, and a 256 entry prediction table. Indirect conditional branches are always predicted as *not-taken*. The branch unit scans ahead 5 instructions to locate a branch instruction. A

Benchmark	Input or (Source Change)	Full Trace Length (millions)	Sampled Trace Length
Espresso	bca	435	100,000
Alvinn	(EPOCHS=10)	322	100,000
Sc	loada1	325	100,000
Sc	loada3	212	100,000
Li	7-queens	249	100,000
Gcc	cexp	18.6	40,000
Gcc	optabs	51.6	40,000

Table 1: Benchmarks used

maximum of two branches may be pending (predicted but unresolved) at any time. Upto three instructions may be dispatched and completed every cycle. Instructions may execute out-of-order but they wait in a 16-entry completion buffer for in-order completion. Those waiting in the completion buffer are considered *finished* but not *completed*. Subsequent instructions can use the results generated by the *finished* instructions. Register renaming is used to reduce non-essential data dependences. Each integer unit is capable of executing register-to-register as well as load/store instruction. Each integer unit has an instruction queue 4-deep, where instructions wait for their operands to become available. Floating-point load/store instructions are executed on one of the integer units. Instruction and data caches are 16K each, with a line size of 64 bytes. The instruction cache is capable of returning 8 instructions per cycle. The data cache has a single cycle latency. A cache miss results in 8 extra cycles for second-level cache access. The second-level cache is assumed to be infinite in all the reported results in this paper. The data cache is *non-blocking* in that subsequent cache accesses are blocked only after the second miss.

4.3 Results

Table 2 contains the predicted CPIs for the chosen set of benchmarks, assuming infinite instruction and data caches. Predictions are provided for both the *periodic window* and *profile-driven* trace sampling techniques. Full trace CPIs are also provided for assessing the accuracy of the predicted CPIs. Note that profile-driven sampling improves the predicted CPI accuracy for all the benchmarks. The improvement ranges from as small as 10 percent to as high as almost 97 percent. It is also interesting to note that whereas, periodic window sampling overestimates as well as underestimates the full trace CPI, the profile-driven sampling always somewhat overestimates the full trace CPI for the chosen set of benchmarks.

Improvement in overall CPI prediction is a function of improvements in predicted performance of various segments of the micro-architecture. We have found profile-driven sampling more effective for various other performance predictions as well. For example, Table 3 illustrates the improvement in branch prediction accuracy for *alvinn* and *Sc*, using profile driven trace input. Figures 1 and 2 illustrate the predicted utilization distributions for the 16-entry completion buffer in the assumed processor model,

Benchmark	Full Trace	Periodic Window		Profile Driven		Error Improvement
	cpi	cpi	Error (%)	cpi	Error (%)	(%)
Espresso	0.751	0.737	−1.9	0.754	+0.4	+78.95
Alvinn	0.875	0.904	+3.3	0.876	+0.1	+96.97
Sc (loada1)	0.883	0.874	−1.0	0.891	+0.9	+10.00
Sc (loada3)	0.979	0.918	−6.2	0.985	+0.6	+90.32
Li	0.891	0.904	+1.5	0.892	+0.1	+93.33
Gcc (cexp)	0.987	0.943	−4.5	1.013	+2.6	+42.22
Gcc (optabs)	0.986	0.939	−4.8	1.007	+2.1	+56.25
Geometric mean of absolute error improvement						+55.10

Table 2: Predicted infinite cache CPI accuracies using different trace sampling techniques. Parameters for periodic window sampling: a) sample-size = 4000 instructions; b) first one percent instructions of full trace skipped before starting sampling; c) inter-sample interval determined by uniform distribution of the trace samples over the full trace. Parameters for profile driven sampling: sampling factor adjusted to get the same sampled trace length as that of periodic window sampling.

for benchmarks *alvinn* and *Sc* respectively. These two benchmarks have been deliberately chosen as they represent extremes in the predicted CPI error improvements (last column of Table 2).

Table 4 contains the predicted CPIs for the chosen set of benchmarks, assuming finite instruction and data caches. Profile-driven sampling improves predicted CPI accuracy for all the benchmarks except *Gcc*, with respect to periodic window sampling. An explanation of this loss of accuracy in the finite-cache simulations of *Gcc* is given below. For other benchmarks, the improvement in CPI prediction ranges from as small as 3.57 percent to as large as 90.16 percent.

As compared to the non-cache elements of typical current processors, e.g., registers, branch prediction table, pipelines, etc., there is a much longer retention of data (lines)

Benchmark	Sampling Strategy	Branch Prediction Accuracy (%)		
		direct	indirect	total
Alvinn	Full Trace	93.13	99.99	93.51
	Profile Driven	91.70	100.0	92.17
	Periodic Window	89.89	100.0	90.46
Sc (loada1)	Full Trace	93.62	87.58	93.59
	Profile Driven	93.38	89.23	93.36
	Periodic Window	91.79	90.70	91.79

Table 3: Branch prediction accuracies using different trace sampling techniques (infinite cache).

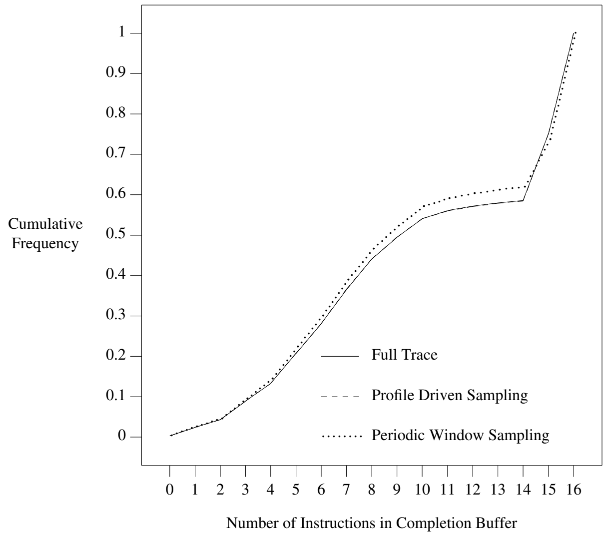

Figure 1: Completion buffer utilization (infinite cache) for *Alvinn*.

Benchmark	Full Trace	Periodic Window		Profile Driven		Error Improvement
	cpi	cpi	Error (%)	cpi	Error (%)	(%)
Espresso	0.820	0.870	+6.1	0.825	+0.6	+90.16
Alvinn	0.911	0.956	+4.9	0.929	+2.0	+59.18
Sc (loada1)	0.986	1.046	+6.1	0.992	+0.6	+90.16
Sc (loada3)	1.070	1.040	+2.8	1.041	+2.7	+3.57
Li	0.914	0.965	+5.6	0.933	+2.1	+62.50
Gcc (cexp)	1.104	1.273	+15.3	1.334	+20.8	−35.95
Gcc (optabs)	1.120	1.306	+16.6	1.325	+18.3	−10.24

Table 4: Predicted finite cache CPI accuracies using different trace sampling techniques. Sampling parameters for periodic window sampling are same as those in Table 2.

in specific memory locations (cache lines). As a result, quite often the sample size requirements for memory system modeling are much higher than that of the rest of the system. Results in [7] recommend 35 samples of 100,000 references each, whereas much smaller samples suffice for non-memory processor system modeling. The *profile driven sampling* technique proposed in this paper introduces fragmentation in the full trace not at periodic time intervals, as in *periodic window sampling*, but rather at intervals dictated by the profile-directed feedback. In case of periodic window based techniques, this fragmentation is restricted explicitly by specifying the *sample-size*. The profile-based technique, as proposed in this paper, does not offer any such explicit support. Therefore, for very small sampled traces, one would expect that the profile-driven technique results in fragmentation, which may have more representative references than the periodic sampling technique but the context for those references is less accurate than in the case of periodic window sampling. Consequently the finite-

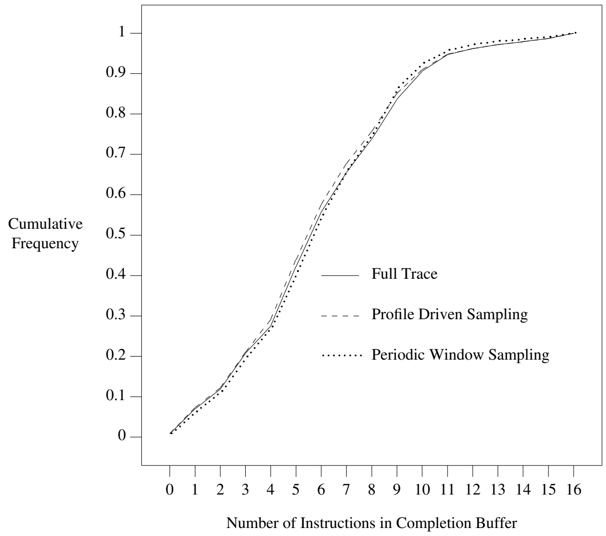

Figure 2: Completion buffer utilization (infinite cache) for *Sc*, with *loada1* as input.

cache performance prediction accuracy is expected to suffer for very small sampled traces, such as *Gcc* in this paper. We believe this to be the reason for the less accurate finite-cache performance prediction of the profile-based technique for *Gcc*, using *cexp* and *optabs* input files. In these cases, the loss of accuracy due to the context mismatch outweighs the gain in accuracy due to more representative set of references. For such cases, the use of profile-driven sampling may be restricted to be used for infinite cache modeling purposes.

4.3.1 Sensitivity on Sampling Parameters

As alluded to before, one of the non-trivial problems with trace sampling is finding the *optimum* setting of the sampling parameters. A necessary parameter common to all such techniques is the size of the sampled trace. One would like to determine the shortest sampled trace that is representative enough of the full trace for the set of intended simulations. Sampling techniques, such as periodic window technique further add to this list of parameters, by introducing the sample-size and the inter-sample interval or sample-placement, as additional parameters. This implies additional complication. Figures 3 and 4 illustrate this by plotting the sensitivity of the predicted CPI as a function of sample-size and sample-placement, respectively. Note that there is almost a 6 percent variation in the predicted values of CPI depending on the sample size, and there is almost a 15 percent variation in the predicted values of CPI depending on how the samples are placed. The *first 3rd* refers to the case where all the samples are uniformly distributed in the first one-third of the full trace, the *second 3rd* refers to the case where all the samples are uniformly distributed in the second one-third of the full trace, and the *third 3rd* refers to the case where all the samples are uniformly distributed in the last one-third of the full trace. Note that uniform placement of samples is not closer to the full trace CPI in this case. Random placement of samples [9] has not been explicitly simulated in this paper. However, in the illustrated case, one can argue that random placement of samples, can yield any

of the four cases, and thus may not necessarily be a better predictor.
Figure 5 illustrates the sensitivity of predicted CPI on the size of the sampled trace. The size of the sampled trace is varied from 20K to 400K. The superior nature of profile-driven sampling is very obvious here. It monotonically becomes a better predictor of the full-trace performance, as the sampled trace size is increased. For large sizes of sampled trace, periodic window sampling also becomes a good predictor of the full trace CPI. However, the periodic window sampling still demonstrates the non-monotonic nature of sensitivity observed in Figures 3 and 4. There is almost a 10 percent variation in the predicted values of CPI using the periodic window sampling, whereas, the same in the case of profile-driven sampling, is only about 0.3 percent.

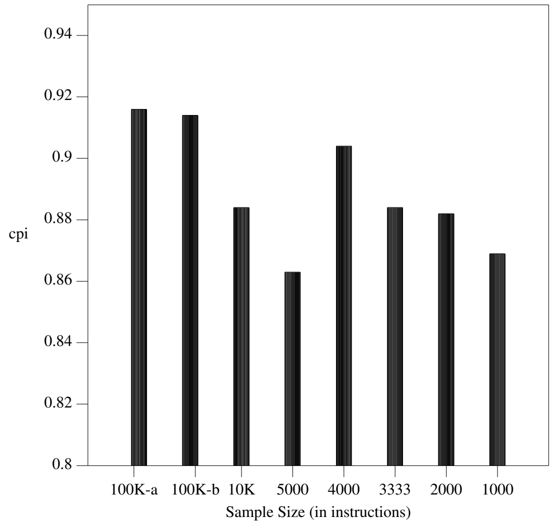

Figure 3: Sensitivity of periodic window sampling on sample size. A sampled trace of 100,000 instructions, consisting of same sized samples, uniformly distributed on the full trace (except the "100K-a" case). Size of the sample used is varied from 1000 instructions to 100,000 instructions. "100K-a" refers to the case where the single 100K instruction sample is derived from the start of the full trace, whereas, "100K-b" refers to the case where the 100K instruction sample is derived from the middle of the trace. Benchmark used: *Alvinn* (infinite-cache). There is almost a 6 percent variation in the predicted values of CPI.

5 Conclusion

Trace driven simulation is a very widely used technique for evaluating different machine design options. For faster simulations, one often needs to significantly reduce the size of the trace input. This gives rise to the problem of obtaining a reduced size trace that is a *good enough* representative of the full trace so that it can replace the full trace during the simulations.
This paper proposes a new technique for obtaining a representative reduced size trace. The technique, referred to as, *profile driven trace generation* is based on a two-step approach. In the first step, the complete benchmark is executed in the

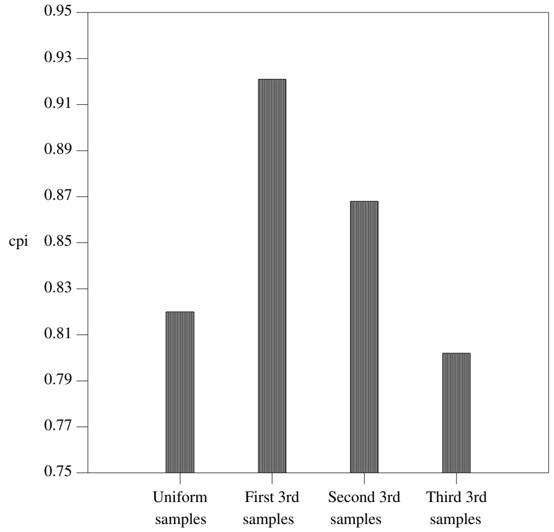

Figure 4: Sensitivity of periodic window sampling on sample placement. Five samples, 4000 instructions each derived from different portions of the full trace. Benchmark used: *Alvinn* (infinite-cache). There is almost a 15 percent variation in the predicted values of CPI.

trace-generation environment to generate its dynamic profile. In the second step, the benchmark is executed again for generating a reduced trace, based on the profile information. The trace size reduction factor is specified as a parameter during the second step.

Unlike previously published trace sampling techniques without the profile feedback (e.g., the periodic window based sampling), the proposed profile-driven sampling technique results in a reduced trace that is almost an exact *representative* of the original full trace in terms of statistics like, instruction mix, basic-block sizes. This obviates the need for many initial validation phase checks. The additional time required to generate the sampled trace is small, usually less than 10 percent. Moreover, this additional cost is a one-time cost, amortized over the several CPU hours of simulations typically performed on such traces.

In the final validation phase, we compare the predicted CPIs of a speculative superscalar processor, using trace inputs based on periodic window sampling and profile-driven sampling. Assuming infinite instruction and data caches, profile-driven sampling is found to reduce the periodic window sampling performance inaccuracies on an average by more than 55 percent. Assuming finite instruction and data caches, except for very small sampled traces, profile-driven sampling is found to improve the predicted CPI accuracy on an average by almost 40 percent.

Finally, the *periodic window* based trace sampling techniques can be quite sensitive to sampling parameters such as, the sample size and the inter-sample interval. We found a variation of as much as 15 percent in predicted performance as a function of sample placement. Such sensitivities are not observed in the *profile driven* sampling technique, as proposed in the paper. This significantly simplifies the trace generation process, as various sampling parameters do not need to be optimized.

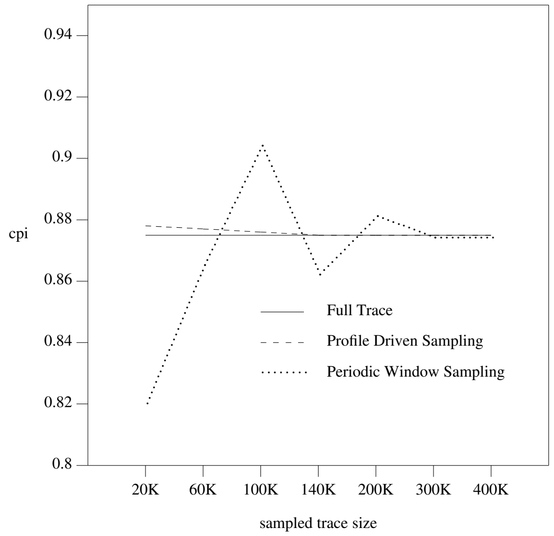

Figure 5: Sensitivity of periodic window and profile driven sampling techniques on the total size of the sampled trace. Sample size used for periodic window sampling = 4000 instructions. Samples uniformly distributed over the full trace for periodic window sampling. Notice the smooth rise in predicted CPI accuracy for the profile-driven sampling. Benchmark used: *Alvinn* (infinite-cache). There is almost a 10 percent variation in the predicted values of CPI, in case of periodic window sampling, whereas, the same in the case of profile-driven sampling is less than 1 percent (0.3 percent).

References

[1] M. Butler, T. Yeh, Y. Patt, M. Alsup, H. Scales, and M. Shebanow, "Single Instruction Stream Parallelism Is Greater Than Two," *Proc. 18th Annual International Symposium on Computer Architecture*, May 1991, pp. 276-286.

[2] J. W. C. Fu and J. H. Patel, "Trace Driven Simulation using Sampled Traces," *Proc. of the 27th Hawaii International Conference on System Sciences*, Jan 1994, pp. 211-220.

[3] W. M. Johnson, *Superscalar Microprocessor Design* Prentice Hall, 1991.

[4] D. R. Kaeli, "Issues in Trace Driven Simulation," *Proc. Performance Evaluation of Computer and Communication Systems, Joint Tutorial Papers PERFORMANCE '93 and SIGMETRICS '93*, Lecture Notes in Computer Science, Vol. 729, 1993, pp. 224-244.

[5] D. R. Kaeli and P. G. Emma, "Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns," *Proc. 18th Annual International Symposium on Computer Architecture*, May 1991, pp. 34-42.

[6] R. E. Kessler, Mark D. Hill, and D. A. Wood, "A Comparison of Trace-Sampling Techniques for Multi-Megabyte Caches," *IEEE Transactions on Computers*, Vol. 43, No. 6, June 1994, pp. 664-675.

[7] S. Laha, J. H. Patel, and R. K. Iyer, "Accurate Low-cost Method for Performance Evaluation of Cache Memory Systems," *IEEE Transactions on Computers*, Vol. 36, No. 11, Nov. 1988, pp. 1325-1336.

[8] J. R. Larus, "Efficient Program Tracing," *IEEE Computer*, May 1993, pp. 52-60.

[9] G. Lauterbach, "Accelerating Architectural Simulation by Parallel Execution of Trace Samples," *Proc. of the 27th Hawaii International Conference on System Sciences*, Jan 1994, pp. 205-209.

[10] B. Lightner, "The SPARC Lightning Processor," *Proc. Hot Chips II Symposium*, Aug. 1990.

[11] R. Nair, "Profiling IBM RS/6000 Applications," to appear in the *International Journal of Computer Simulation.*