

Simulation Differences Between Academia and Industry: A Branch Prediction Case Study

Gabriel H. Loh
Georgia Institute of Technology
College of Computing
Atlanta, GA, USA
loh@cc.gatech.edu

Abstract

Computer architecture research in academia and industry is heavily reliant on simulation studies. While microprocessor companies have the resources to develop highly detailed simulation infrastructures that they correlate against their own silicon, academic researchers tend to use free, widely available simulators. The differences in instruction set architectures, operating systems, simulator models and benchmarks create a disconnect between academic and industrial research studies. This paper presents a comparative study to find correlations and differences between the same microarchitecture study conducted in two different frameworks. Due to the limited availability of industrial simulation frameworks, this research is limited to a case study of branch predictors. Encouragingly, our simulations indicate that several recently proposed branch predictors behave similarly in both environments when evaluated with the SPECcpu benchmark suite. Unfortunately, we also present results that show that conclusions drawn from studies based on SPECcpu do not necessarily hold when other applications are considered.

1. Introduction

One of the primary tools of microarchitecture research is simulation. A large portion of academic processor research uses SimpleScalar [2] or some other publicly available (and usually free) infrastructure to conduct their performance studies. Researchers in academia do not have access to the simulators used in industry for a variety of reasons related to intellectual property and confidentiality. As a result, academics have largely been working in the dark, not really knowing whether or not the results they produce have any relation to the studies conducted privately in industry.

The simulation infrastructure has two major components: the simulator code itself, and the benchmarks/traces used to feed the simulators. While we do not have access to a full timing simulator from industry, the recent Championship Branch Prediction (CBP) contest [1] has made a limited number of industrial branch traces and a basic branch predictor simulation framework available to the public. In this study, we compare the Intel x86 traces with more conventional academic traces to attempt to answer some basic questions about the value of academic studies.

Due to the nature of the available infrastructure, this study focuses on the particular microarchitecture sub-domain of branch prediction. We try to answer the following questions. How similar or different are the traces used by industry and academia? Do conclusions drawn from studies with academic traces extend to studies performed with industrial traces? Since the CBP traces also include multimedia and server workloads, we also ask, is the SPECcpu suite really representative of the types of applications in which industry is interested?

The primary contribution of this study is to provide a first-order sanity-check on simulation methodology employed in academic research studies. A strong correlation of results between branch-prediction-only studies does not necessarily imply that other performance studies are likely to show similar correlations. On the other hand, a great disconnect in the branch prediction studies would suggest that more detailed and sophisticated studies are less likely to be relevant. Put another way, if academics cannot produce branch prediction simulation results that are valuable to industry, what chance is there that researchers and engineers in industry could draw meaningful conclusions from “cycle-accurate” simulation results for more complicated proposals?

The community is well aware of the industry/academia gap, as evidenced by a recent discussion

panel on this topic [33], but the number of research efforts aimed at closing this gap are not many. While other research has calibrated detailed academic simulators against actual systems, these efforts compare software and hardware for the exact same processor. For example, Black and Shen compared their Microarchitecture Workbench model of a PowerPC 604 against the real hardware [3], and Desikan et al. compared their sim-alpha model to an actual Alpha 21264 [8]. Calder et al. compared a cycle-level simulator of a DEC 21064 against the real hardware [4]. Furthermore, their study showed that branch misprediction rates are highly correlated with overall performance. The novelty of our study is that it presents a cross-architecture comparison of a microarchitecture feature to observe correlation (or the lack thereof) across very different instruction sets and application traces. The importance of this type of study is that it reflects the typical case where there is a mismatch in instruction sets, traces, operating systems and simulators between the academic producers of research and the industrial consumers.

The rest of this paper is organized based on answering the questions posed earlier. Section 2 first lays out the ground work describing the two simulation frameworks and the simulated predictors evaluated in this study. Section 3 compares SimpleScalar SPECint results with the CBP SPECint results. Section 4 repeats the evaluation using the SPECfp benchmarks. Section 5 addresses the question of the comprehensiveness of the SPECcpu benchmark suite, comparing results to the CBP multimedia and server workloads. Section 6 draws some final conclusions and makes some recommendations for future interaction between industry and academia.

2. Description of Simulation Infrastructures

This section describes the two infrastructure setups used in this study. First we detail the branch predictor simulator and trace set provided for the Championship Branch Prediction (CBP) competition. The second part describes the SimpleScalar-based infrastructure and its associated benchmarks and traces. Lastly, we also describe the branch prediction algorithms evaluated in our studies.

2.1. Championship Branch Prediction

The goal of the Championship Branch Prediction (CBP) competition is to encourage the innovation of new branch prediction technologies in a forum where multi-

ple designs can be directly compared against each other in a common framework [1]. To realize this, the CBP organizers have provided a branch predictor simulation infrastructure that processes branches from a pre-recorded x86 instruction trace. The simulator reads the information for one branch at a time. The predictor’s lookup and update functions are each called once per branch. There is no other simulation of any other component of the processor at either a functional or timing level.

The traces that are inputs to the simulator were collected from 20 different applications, five each from four different application groups. The first two groups are the traditional SPECint and SPECfp benchmarks, although it is unclear which ten applications were chosen from the 26 integer and floating point programs. The other two groups of traces come from multimedia and server workloads, with no indication as to their origins. Each trace contains branches from about 30 million total instructions (branch and non-branch); the total number of branches per trace varies based on the number of instructions per branch for each application. The traces include both user and system activity.

2.2. SimpleScalar Alpha

Our “academic” framework is based on the SimpleScalar (SS) toolset for the Alpha instruction set architecture [2]. In particular, we use the in-order branch predictor simulator *sim-bpred* because it most closely matches the functionality of the CBP infrastructure. The SS simulator performs a functional simulation of the benchmark, but we do not make use of any information that is not available to the CBP simulator, such as register or memory values. The SS framework is also limited in that it does not provide branch activity associated with system calls.

The traces used for the SimpleScalar studies come exclusively from the SPEC2000 CPU benchmark suite. These cover all of the integer applications, and all but one floating point application. The single omitted floating point application was *facerec* which we could not run because we lacked the correct Alpha Fortran environment. Each trace consists of 100 million instructions, with the selected interval chosen based on the SimPoint single early simulation points [25]. The binaries are those publicly available from the SimpleScalar website (on which the published SimPoint intervals are based). All inputs are from the SPEC *ref* data sets, and all inputs for a given benchmark were used when possible (i.e. multiple inputs exist and representative simulation points have been determined for those inputs). These are detailed in Table 1. Due to the discrepancy between

Benchmark	Suite	Input(s)	Total Traces
bzip2	INT	graphic, program, source	3
crafty	INT	crafty.in	1
eon	INT	rushmeier	1
gap	INT	ref.in	1
gcc	INT	166.i, 200.i, expr.i integrate.i, scilab.i	5
gzip	INT	graphic, log, program random, source	5
mcf	INT	inp.in	1
parser	INT	ref.in	1
perlbmk	INT	diffmail, makerand perfect, splitmail	4
twolf	INT	ref	1
vortex	INT	lendian{1,2,3}	3
vpr	INT	route	1
art	FP	ref (two different sets of input parameters)	2
facerec	FP	<i>unsupported environment</i>	0
Remaining	FP	ref	12

Table 1. The SPEC applications and the input sets used for the SimpleScalar simulations.

the number of instructions in the two different sets of traces (CBP vs. SS/SimPoint), we also repeated all experiments using only the first 30 million instructions of the SimpleScalar SimPoint traces.

Any time averages are reported, each application is given the same weighting. This means that applications with multiple input sets will have each trace weighted less. For example, when computing the average prediction rate for the twelve integer applications, *crafty* is weighted by 1/12, while each of the three *bzip2* traces are weighted by 1/36 such that *bzip2* as a whole still receives a total weighting of 1/12.

2.3. Simulated Branch Predictors

This study evaluates several previously proposed branch prediction algorithms in different simulation frameworks. The goal is to determine whether previously published results hold when evaluated with both academic and industrial simulators, and whether conclusions from an academic setting can be extended to an industrial context.

We aim to cover a wide range of branch prediction approaches from the conventional to the most recent proposals. The predictors used in this study are the *gshare* [22], *Bi-Mode* [19], *gskewed* [23], *perceptron* [14, 15] and the path-based neural predictor [13]. The *gshare* predictor hashes the global branch history with the program counter to index a table of saturating two-bit counters that provide the branch prediction. The

Bi-Mode and *gskewed* predictors are similar, but employ different techniques to reduce the effects of destructive branch aliasing in the predictor tables. The *perceptron* uses a simple neural network to detect correlations in a potentially very long global branch history, and the path-based neural predictor incorporates branch path history to provide more context in making its predictions. We vary the hardware budget in powers-of-two from 1KB up to 128KB.

The configurations chosen are the best found after sweeping a large design space, subject to a few constraints. The number of entries in all tables are constrained to powers-of-two. The maximum branch history allowable for a given predictor is limited by the size of its pattern history table (PHT). For example, we limit a table with 2^n entries to a branch history register of n bits. While it is possible to hash a branch history register with $m > n$ bits down to n bits [27], we chose to limit the history length to restrict the size of our design space.

As described in their original studies, the *gskewed* and *Bi-Mode* predictors are harder to directly compare against other predictors because they employ an odd number of tables that does not naturally fit into a power-of-two number of bits [19, 23]. For the *Bi-Mode* predictor, we doubled the number of entries in the choice PHT. In our implementation of the *gskewed* predictor, we used the enhanced *gskewed* algorithm with a partial update policy on the individual banks [23]. We then doubled the number of entries in the first PHT bank which corresponds to the “capacity” bank as detailed by Michaud et al. In this fashion, all predictors evaluated at a given hardware budget have a total number of bits that are very close. Minor differences exist due to different global branch history register sizes or the need for a path history buffer.

There are a large number of other candidate algorithms that we could have chosen [6, 10, 16, 20, 21, 24, 26, 27, 29–31], many of which were not considered because they use non-branch information [32], profile information [5, 7], timing information [11], or were similar to other predictors already included in the study [9]. The goal is not to conduct an exhaustive comparison of all predictors ever proposed; branch prediction merely serves as a vehicle to conduct comparisons between simulation infrastructures.

3. SPECint Comparison

This section presents the results of our comparative study using the integer portion of the SPECcpu benchmark suite. SPEC is a very common application set used in academic and industrial research. In particular

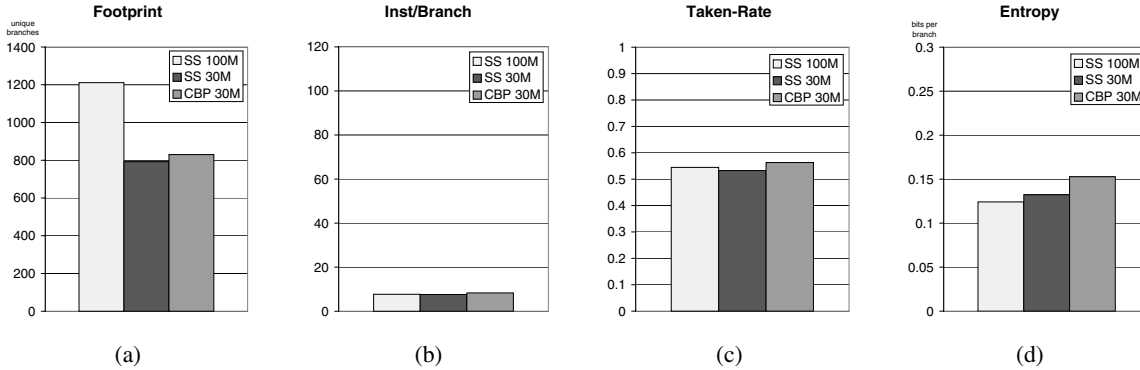


Figure 1. Basic statistics for the SPECint traces: (a) number of unique conditional branch PCs, (b) number of instructions per conditional branch (c) conditional branch taken rate, (d) conditional branch entropy.

for the branch prediction sub-domain, the focus has traditionally been on the integer applications because the floating-point benchmarks typically contain very regular control-flow structures and patterns that are easily predictable, and the performance for these applications is not limited by the branch predictor.

The first part of this section analyzes various statistics of the SimpleScalar (SS) and CBP traces independent of any particular branch predictor algorithm. The second part compares the prediction rates achieved by the different algorithms on both the SS and CBP frameworks.

3.1. Basic Trace Statistics

We measured four different statistics from both the SimpleScalar and CBP traces. The goal is to determine if there are any major differences in the traces themselves that could cause the predictor simulations to produce different results. Figure 1(a) shows the average branch footprints for the three different trace sets analyzed (SimpleScalar with two trace lengths, and CBP). The branch footprint is the number of unique branch program counters (PCs) observed, including conditional branches and all unconditional branches such as indirect jumps and subroutine calls and returns. The 100 million instruction SimpleScalar traces (SS 100M) have a considerably larger branch footprint than the CBP traces. The longer trace covers a larger fraction of the static program address space, about 58% more unique branch addresses. The shorter 30M trace results in a footprint similar to the CBP traces, which provides evidence that the traces exercise a similar amount of code.

The second statistic measured is the average number of instructions per branch (either conditional or unconditional) which determines the average basic block size. The basic block size has a strong dependence on the ag-

gressiveness of the compiler, the algorithm implemented in the application, as well as the underlying instruction set architecture. The results in Figure 1(b) show that there is not much difference in average basic block sizes of the different trace sets.

The last two statistics deal with the predictability of the conditional branches in the traces. Figure 1(c) shows the average rate of taken conditional branches. This just illustrates that the traces all have an approximately even split between taken and not-taken branch outcomes. Figure 1(d) is an estimate of the information theoretic entropy in the global branch stream. The entropy provides us with a rough estimate of the randomness of the branches, and serves as an indicator of how difficult it should be to accurately predict the branch outcomes.

The exact entropy of a data stream is difficult to compute explicitly. To estimate the entropy of the branch traces, we instrumented the respective simulators to dump only the conditional branch outcomes (taken or not-taken) to a file. All branch address and other decode-level information was removed, with eight outcomes packed into each byte. We then attempt to compress this resulting outcome-dump file as much as possible. To perform this compression, we use both the gzip and bzip2 compression utilities with all possible compression settings (-1 to -9). The smallest resulting file provides a bound on the compressibility of the branch stream. By dividing the size of the smallest compressed file by the original outcome-dump file size, we obtain an upper bound on the bits of information conveyed per branch in the entire stream. The bzip2 utility always provided the best compression because at its most aggressive setting, it can search the entire outcome-dump file for patterns. The bzip2 utility is able to achieve a better overall compression on the SS 100M traces because

the larger trace provides more opportunities to search for repeated patterns.

The results in Figure 1(d) show that the branches in the integer traces convey at most about 0.12 to 0.15 bits of information per conditional branch. We observe that the relative randomness or difficulty of predicting the branches in these traces is close, indicating that branch prediction results derived from the SimpleScalar traces should translate to the CBP framework and vice versa. Overall, the statistics presented in Figure 1 provide some encouraging evidence that conclusions drawn from branch prediction studies conducted with the SimpleScalar framework should be generalizable to the CBP framework.

3.2. Predictor Accuracy

The statistics presented above hint that the branch predictors should behave similarly when evaluated with either the academic or industrial frameworks. We now present the comparative results of actually running several different predictors in the two frameworks. Figure 2(a) shows the misprediction rate of the five predictors used in this study averaged across the SimpleScalar 100M instruction traces, while Figure 2(b) shows the corresponding results for the CBP traces. The same experiments were conducted with the SS 30M traces as well, and the results are similar. The global branch history length for each predictor was chosen to minimize the average misprediction rate across the integer benchmarks. As a result, different algorithms make use of different history lengths, and the same algorithm may have different history lengths between the different simulation infrastructures.

The SimpleScalar results confirm past studies, showing that the neural predictors perform the best, followed by the interference reducing algorithms, and then finally gshare. For the CBP results, most of the predictors follow a similar trend. The only glaring exception is the perceptron predictor, which does not attain the same relative ranking until the hardware budget has been raised to 8KB. The exact differences in the prediction rates vary from one trace set to the other, but the overall trends are similar.

We also present the optimal global history lengths that we found for each of the predictor configurations, shown in Table 2. Even if the overall performance trends are similar between the SimpleScalar and CBP runs, the exact configuration details may lead to different design choices. For example, are long history lengths equally valuable in both contexts? Except for the path-based neural predictor, the CBP traces tend to favor predic-

tors with slightly shorter branch history lengths than the SimpleScalar traces.

For a fixed-size index, dedicating more bits for branch history increases the opportunities to correlate with past outcomes, while incorporating more bits from the program counter allows a predictor to distinguish between a larger number of branches. The results in Table 2 suggest that the CBP traces need more bits from the branch address rather than branch history. We believe that this is an artifact of the x86 architecture which allows instructions to not be aligned to natural word boundaries. While an instruction can start on any byte, this does not mean that the lower bits of PCs are evenly distributed. The result is that the utilization of predictor table entries is also not uniform, thus explaining the greater relative benefit of initial size increases for the predictors on the CBP traces as shown in Figure 2(b).

The increased interference due to the uneven distribution of the lower PC bits also provides an explanation for why the perceptron predictor performs much worse with the CBP traces than in the SimpleScalar case. The reason why the path-based neural predictor does not suffer the same fate is because each weight is indexed by a different PC, and so the predictor as a whole can tolerate some interference in some of the weights, so long as there are some other weights that are correctly trained. This may also explain why set-associativity is used in the branch predictor/BTBs of the Intel Pentium processors [28], while the PHTs used in the Alpha EV6 [17] and the proposed EV8 [27] did not use tagged structures.

4. SPECfp Comparison

This section repeats the experiments presented in Section 3 on the floating point benchmarks. While floating point applications are not frequently used in research studies of branch predictors, this portion of our study provides further data to help determine whether our academic simulation infrastructure has any correlation with the industrial setup. Furthermore, since these traces were actually distributed for the CBP competition, one might conclude that the performance of branch predictors on floating point applications is still of some interest to researchers and engineers in industry.

The results corroborate that the floating point (FP) applications are very predictable. Figure 3 shows the four statistics collected on the three sets of traces. The y-axes on the four charts are identical to the corresponding graphs from Figure 1 to allow direct comparisons between the integer and floating point applications. The FP applications exercise a much smaller number of unique branches for the same number of simulated instructions.

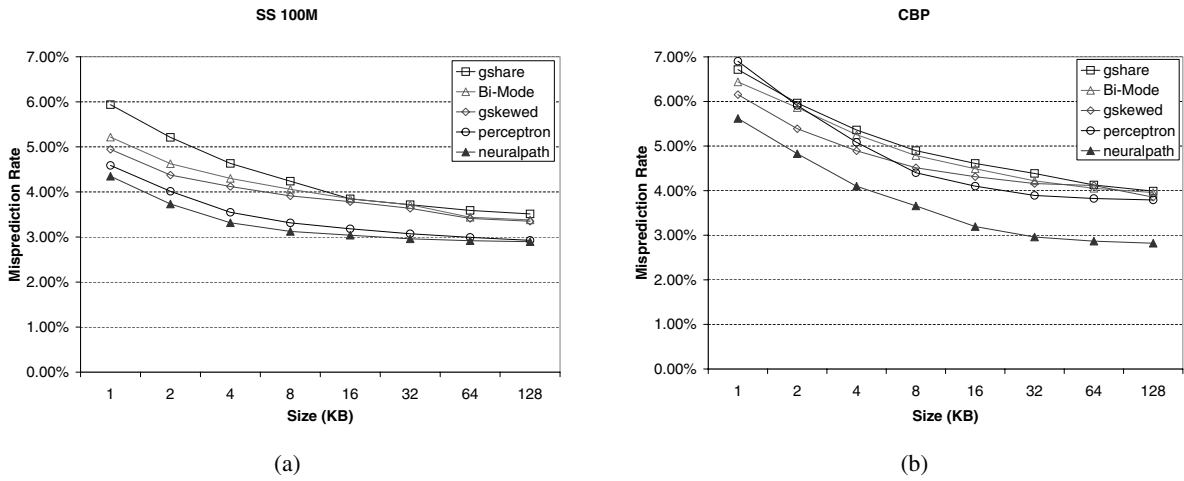


Figure 2. Misprediction rates on SPECint for the (a) SimpleScalar and (b) CBP frameworks. Lower is better.

Hardware Budget (KB)	gshare		Bi-Mode		gskewed		perceptron		path-neural	
	SS	CBP	SS	CBP	SS	CBP	SS	CBP	SS	CBP
1	11	7	10	9	9	7	S	S	S	S
2	12	10	11	10	11	11	M	M	M	S
4	14	11	12	12	12	12	M	M	M	M
8	15	13	13	13	13	13	L	M	M	L
16	16	14	14	14	14	14	L	M	M	L
32	16	16	15	15	15	15	L	M	L	L
64	17	18	16	16	16	16	L	M	L	L
128	18	18	17	17	17	17	L	M	L	L

Table 2. Optimal global history lengths on SPECint for different branch prediction algorithms. For the neural predictors, ‘S’ stands for a short history ($16 \pm$ bits), ‘M’ stands for a medium-length history ($32 \pm$ bits), and ‘L’ stands for a long history ($64 \pm$ bits).

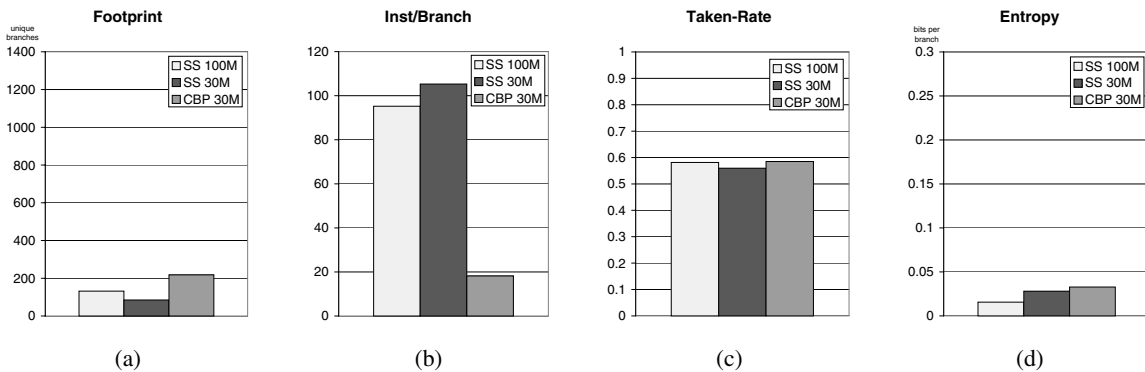


Figure 3. Basic statistics for the SPECfp traces: (a) number of unique conditional branch PCs, (b) number of instructions per conditional branch (c) conditional branch taken rate, (d) conditional branch entropy.

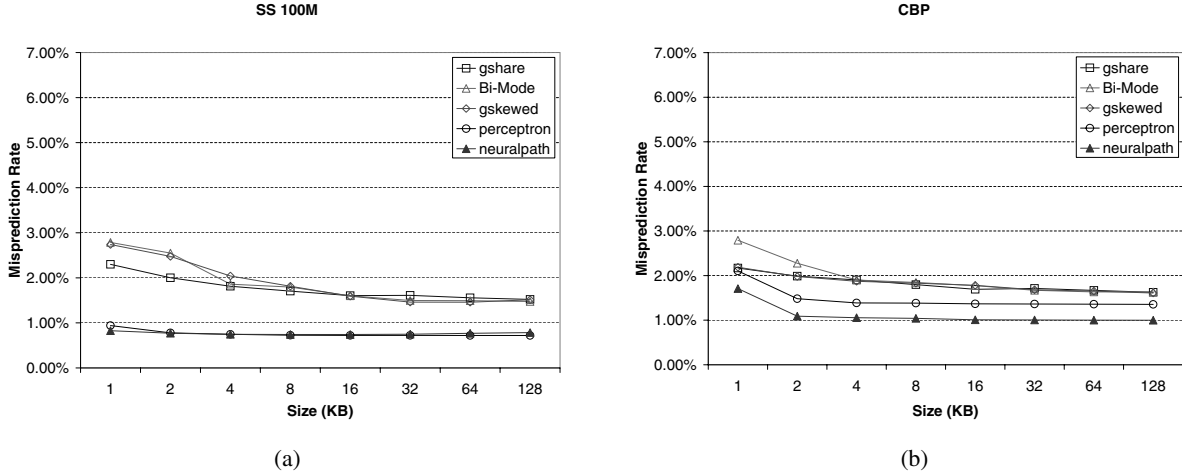


Figure 4. Misprediction rates on SPECfp for the (a) SimpleScalar and (b) CBP frameworks. Lower is better.

This is primarily due to the large loop iteration counts typical of many FP programs. Figure 3(b) shows that the average number of instructions per branch does differ significantly between the SimpleScalar and CBP traces. The main reason for this difference is the availability of floating point vector instructions (SIMD) in the x86 architecture, which allows a single instruction to encode what would take several instructions in a RISC architecture. The SS footprints are also biased by the applu and mgrid programs which both have very large blocks of matrix operations with no intervening branches. Loop unrolling can further increase the observed basic block sizes. Both the taken-rates and entropy estimates are relatively consistent across the trace sets. The entropy is much less than for the integer benchmarks, which is consistent with the common belief that the branches in FP benchmarks are easier to predict.

The results in Figure 4 corroborate the notion that floating point benchmarks are “easy” for branch predictors. Even a 1KB gshare predictor achieves a misprediction rate close to 2% for both SimpleScalar and CBP trace sets. Similar to the integer benchmarks, the neural techniques provide the best prediction rates. The smaller footprint of the floating point applications allows the perceptron predictor to perform better relative to the non-neural predictors when compared to the CBP integer traces. The results also demonstrate that branch path history has a much greater impact for the x86 CBP traces than for the Alpha SimpleScalar runs.

Thus far, the comparisons of the SPEC applications for both SimpleScalar and CBP infrastructures indicate that results from one can be generalized to the other to a reasonable extent. So long as the SPECcpu bench-

mark suite is what industry is interested in, the academic methodologies for evaluating branch predictors are sufficiently sound to draw meaningful conclusions. While the general trends correlate well, subtle differences such as the impact of using path history may lead to different design decisions.

5. Is SPEC Representative?

Apart from the SPECcpu benchmarks, the CBP competition also provides a limited set of traces from multimedia and server workloads. This provides evidence that the design of real microprocessors involves the evaluation of microarchitectures running more than just the SPECcpu benchmark suite. While this fact may already be obvious to people who are in contact with industry researchers, these traces provide us with some opportunity to gauge just how representative or limiting the current SPECcpu benchmarks really are.

In this section, we repeat the previous experiments with the four application classes of the CBP trace set: integer (INT), floating point (FP), multimedia (MM) and server (SERV). Because Sections 3 and 4 have already demonstrated the strong correlation between SimpleScalar and CBP traces for the SPECcpu applications, we do not reproduce the SimpleScalar data in this section.

5.1. Basic Trace Statistics

We again start by analyzing the basic branch statistics independent of any particular branch prediction algorithm. Figure 5 shows the collected statistics across the four CBP application groups. The first chart in Fig-

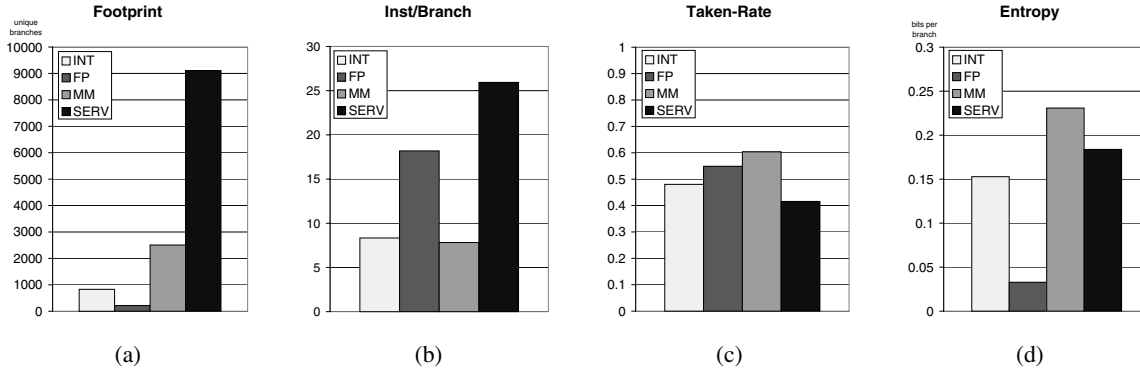


Figure 5. Basic statistics for the CBP traces: (a) number of unique conditional branch PCs, (b) number of instructions per conditional branch (c) conditional branch taken rate, (d) conditional branch entropy.

Figure 5(a) shows that there are considerable differences in the branch footprints between the SPEC benchmarks and the multimedia and server workloads. (Note that Figure 5(a) and (b) have different scales on the y-axis than the corresponding graphs from Figures 1 and 3.) Depending on the importance of the multimedia and especially the server workloads, making branch predictor tradeoff decisions that reduce the number of entries in the predictor tables could be a very bad choice. Using the SPEC benchmarks alone, one would arrive at a different conclusion and make different tradeoffs.

The average number of instructions per branch for the server workloads is very different than the other application groups. Figure 5(b) shows that while the multimedia codes have similar basic block sizes as compared to the integer programs, the server workloads process many more instructions between branches. While the basic block size is generally not a concern for most branch predictors, there are a few algorithms that could potentially be impacted by this difference. The dataflow-based branch predictor classifies branches based on which registers are modified within a basic block [32]. The combination of large basic blocks and a small architected register set in the x86 ISA would make it likely that almost every register is modified in every basic block. This would result in every branch getting the same dataflow “signature”, which would eliminate the benefits of the technique on the server workloads.

The taken rates of the branches for the multimedia and server traces are not too different from the SPEC applications. Figure 5(c) shows that the average is still within 10% of being evenly split. This result is not surprising given the fact that an overly strong taken or not-taken bias would tend to make the branches much easier to predict.

The entropy estimates for the multimedia and server traces are on average greater than the SPECcpu benchmarks. This indicates that these traces should be more difficult to predict. A point that should be observed is that the entropy estimates for applications with larger footprints are less accurate than those for smaller applications. This is because our entropy estimation methodology discards the branch addresses which may actually contain a significant amount of information. For example, consider a hypothetical program where every branch address with an odd number of ones corresponds to a taken branch, and every branch address with an even number of ones is not-taken. If these branches are traversed in a seemingly random fashion, then the branch outcome trace will produce a bit-stream with high entropy. Given the branch addresses, it is clear that the overall predictability is 100%.

5.2. Predictor Performance

The performance of the branch predictors on the multimedia and server workloads is shown in Figure 6. The statistics presented in Figure 5 for the multimedia traces are actually fairly similar to the integer traces. As a result, it is not surprising to see that the predictor accuracy trends are also similar. While the trends show a reasonably strong correlation, the entropy estimates of Figure 5(d) suggest that the multimedia traces should be more difficult to predict than the integer traces. The misprediction saturation point on the multimedia benchmarks for the algorithms evaluated appears to be about 4%, whereas this saturation point is near 3% for the integer applications.

The server workloads are very interesting because of their considerably larger branch footprints. Not surprisingly, both of the neural approaches perform much

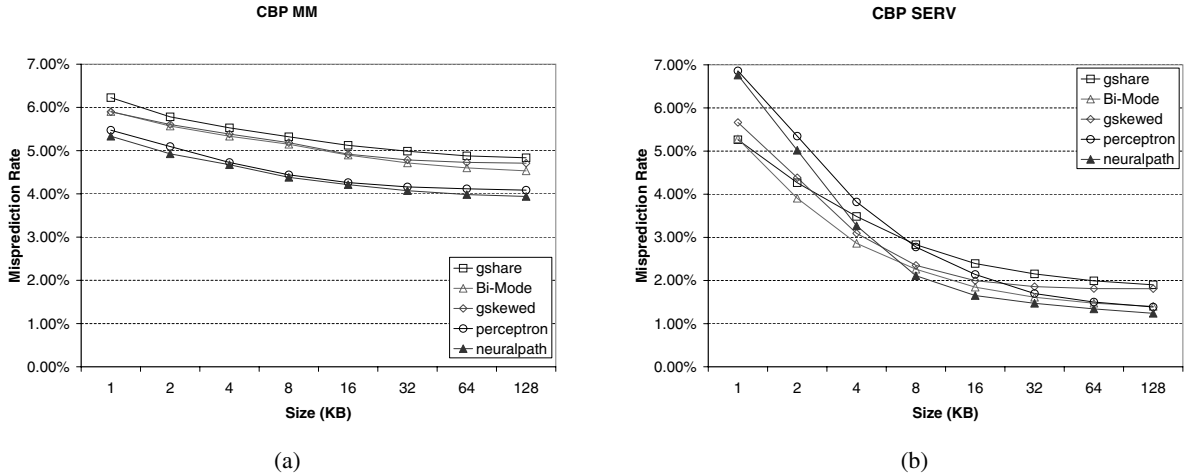


Figure 6. Misprediction rates for the non-SPEC CBP traces: (a) Multi-Media, (b) Server.

worse at small hardware budgets where the amount of inter-branch aliasing is very high. For such workloads, the ability to effectively deal with the capacity problem is more important than being able to mine subtle correlations from a deep branch history.

Another interesting observation is that the server traces greatly benefit from increases in the predictor hardware budget. Figure 6(b) shows that given enough state, the neural predictors can achieve a misprediction rate that is close to 1%. This suggests that the branches are not intrinsically difficult to predict, but it is merely the sheer volume of unique branches that makes it challenging.

The different characteristics of the multimedia and server traces compared to the SPECcpu applications make it likely that the optimal history lengths also differ. Table 3 shows the optimal history lengths for the gshare, gskewed and path-based neural predictors. The results for the other two predictors were omitted in the interest of space. The strongest trend is that the optimal history length is heavily correlated with the branch footprint. Traces that have smaller footprints (such as FP) need fewer bits from the branch address to distinguish between different branches, and as a result can more effectively make use of the global branch history. These results again highlight the fact that the choice of benchmarks can have a profound impact on engineering decisions and tradeoffs.

6. Conclusions

This paper has presented a comparative study of two different simulation frameworks: one representative of a

typical academic infrastructure, and the other is a simulator provided by industry. Given very different instruction set architectures and applications, we can actually produce similar results in both environments with the SPECcpu benchmark suite. While we cannot jump to the conclusion that all academic processor simulators provide meaningful results, this study provides a first piece of evidence that all hope is not lost.

The bad news is that SPECcpu2000 is not likely to be representative of all application classes that are of interest when designing a modern microprocessor. The design decisions and the importance of different tradeoffs vary considerably when the application mix is changed. Unfortunately, many of the applications that are interesting for industry pose logistical problems for academic researchers. Some applications are only available for specific architectures or even operating systems (e.g. any of the Microsoft applications, many multimedia programs and games). Other applications, such as realistic enterprise-scale databases, are simply beyond the financial means of most academic researchers to obtain. Attempts have been made to develop new benchmark suites to capture different program behaviors such as MediaBench [18] and MiBench [12]. Even if industrial traces were available, the sheer number of traces presents a heavy simulation burden on many researchers. Currently, simulating only the SPECint applications on a cycle-accurate simulator requires many CPU hours. A recent branch prediction study published in part by researchers from Intel used 110 different benchmarks [11]. Many researchers, particularly at smaller institutions without significant simulation resources, simply could not complete enough simulations for a meaningful mi-

Hardware Budget (KB)	gshare				gskewed				path-neural			
	INT	FP	MM	SERV	INT	FP	MM	SERV	INT	FP	MM	SERV
1	7	11	8	2	7	10	8	4	S	M	S	S
2	10	12	9	3	11	11	9	5	S	L	S	S
4	11	13	8	4	12	12	9	8	M	L	S	S
8	13	15	9	8	13	13	12	9	L	L	M	S
16	14	16	14	10	14	14	14	12	L	L	M	S
32	16	17	16	12	15	15	15	15	L	L	L	M
64	18	18	15	15	16	16	15	15	L	L	L	M
128	18	18	16	16	17	17	15	15	L	L	L	L

Table 3. Optimal history lengths on the CBP traces for three branch prediction algorithms.

croarchitecture design space exploration.

Even if companies in industry made their simulators and traces available to academia, there is the possibility that widespread academic adoption still would not occur. The distribution of the industrial simulators may only be limited to certain institutions with strong relationships with the companies. This in turn may bias conferences and journals to publish a disproportionate number of articles from these selected schools. Even if a company releases code to an academic researcher, it is likely that it is only a specific part of the simulator or that the code has been sanitized or stripped down in some fashion. This reduces the usefulness, and possibly accuracy, of the simulator to the academic researcher, and makes it less likely for the simulator to be adopted.

While this study used an Intel x86-based infrastructure for an “industrial” simulator, an increase in the depth of interaction and collaboration between academia and other companies is also needed to facilitate better technology transfer. In fact, many semiconductor companies provide generous financial and hardware support to a large number of academic research efforts. To improve the return on their academic investments, we believe it is in their best interest to more closely engage the academic community in the evaluation of current methodologies and the development of new ones if necessary.

There are several possible models for this engagement between industry and academia. One approach, which we are currently taking, is for researchers from academia and industry to independently conduct identical microarchitecture studies in their respective simulation environments, and then compare the results. This will help to determine the amount of correlation between the different methodologies, and answer whether results based on academic cycle-level processor simulators are useful to researchers and practitioners in industry. At the same time, this approach avoids the difficult issues related to publicly releasing proprietary simulation tools and traces.

There are other approaches to measuring the simulation gap between academia and industry. Companies may privately conduct comparative studies between their own proprietary simulators and academic infrastructures, and then disclose their findings (but not their simulators) in a public forum such as a workshop or conference. If such results show that there exists a high correlation between academic and industry studies, then it would validate the methodology and simulators already in use. The problem of getting access to industrial simulators then becomes a moot point. No matter what the model is, improving the value and usefulness of academic research in microarchitecture requires more studies in comparing, correlating, and quantifying the abilities and weaknesses of our simulation infrastructures.

Acknowledgments

We would like to thank Jared Stark and Chris Wilkinson of Intel for organizing the CBP contest and making the CBP simulation infrastructure publicly available. The reviewer comments were very helpful in strengthening the paper and provided several different opinions and ideas about possible interactions between industry and academia. Milos Prvulovic of Georgia Tech also provided useful feedback. Some equipment and funding support was provided by Intel Corporation.

References

- [1] The 1st JILP Championship Branch Prediction Competition (CBP-1). <http://www.jilp.org/cbp>.
- [2] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Micro Magazine*, pages 59–67, February 2002.
- [3] B. Black and J. P. Shen. Calibration of Microprocessor Performance Models. *Computer*, 31(5):59–65, 1998.
- [4] B. Calder, D. Grunwald, and J. Emer. A System Level Perspective on Branch Architecture Performance. In *Proceedings of the 28th International Symposium on Microarchitecture*, pages 199–206, Ann Arbor, MI, USA, November 1995.
- [5] P. Chang and U. Banerjee. Profile-Guided Multi-Heuristic Branch Prediction. In *Proceedings of the International Confer-*

- ence on *Parallel Processing Vol. I*, volume 1, pages 215–218, Urbana-Campaign, IL, USA, August 1995.
- [6] P.-Y. Chang, E. Hao, and Y. N. Patt. Alternative Implementations of Hybrid Branch Predictors. In *Proceedings of the 28th International Symposium on Microarchitecture*, pages 252–257, Ann Arbor, MI, USA, November 1995.
- [7] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. N. Patt. Branch Classification: a New Mechanism for Improving Branch Predictor Performance. In *Proceedings of the 27th International Symposium on Microarchitecture*, pages 22–31, San Jose, CA, USA, November 1994.
- [8] R. Desikan, D. Burger, and S. W. Keckler. Measuring Experimental Error in Microprocessor Simulation. In *Proceedings of the 28th International Symposium on Microarchitecture*, pages 266–277, Göteborg, Sweden, June 2001.
- [9] A. N. Eden and T. N. Mudge. The YAGS Branch Prediction Scheme. In *Proceedings of the 31st International Symposium on Microarchitecture*, pages 69–77, Dallas, TX, USA, November 1998.
- [10] M. Evers, P.-Y. Chang, and Y. N. Patt. Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches. In *Proceedings of the 23rd International Symposium on Computer Architecture*, pages 3–11, Philadelphia, PA, USA, May 1996.
- [11] A. Falón, J. Stark, A. Ramirez, K. Lai, and M. Valero. Prophet/Critic Hybrid Branch Prediction. In *Proceedings of the 31st International Symposium on Computer Architecture*, pages 250–261, München, Germany, June 2004.
- [12] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *Proceedings of the 4th Workshop on Workload Characterization*, pages 83–94, Austin, TX, USA, December 2001.
- [13] D. A. Jiménez. Fast Path-Based Neural Branch Prediction. In *Proceedings of the 36th International Symposium on Microarchitecture*, pages 243–252, San Diego, CA, USA, December 2003.
- [14] D. A. Jiménez and C. Lin. Dynamic Branch Prediction with Perceptrons. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture*, pages 197–206, Monterrey, Mexico, January 2001.
- [15] D. A. Jiménez and C. Lin. Neural Methods for Dynamic Branch Prediction. *ACM Transactions on Computer Systems*, 20(4):369–397, November 2002.
- [16] T. Juan, S. Sanjeevan, and J. J. Navarro. Dynamic History-Length Fitting: A third level of adaptivity for branch prediction. In *Proceedings of the 25th International Symposium on Computer Architecture*, pages 156–166, Barcelona, Spain, June 1998.
- [17] R. E. Kessler. The Alpha 21264 Microprocessor. *IEEE Micro Magazine*, 19(2):24–36, March–April 1999.
- [18] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems. In *Proceedings of the 30th International Symposium on Microarchitecture*, pages 330–335, Research Triangle Park, NC, USA, December 1997.
- [19] C.-C. Lee, I.-C. K. Chen, and T. N. Mudge. The Bi-Mode Branch Predictor. In *Proceedings of the 30th International Symposium on Microarchitecture*, pages 4–13, Research Triangle Park, NC, USA, December 1997.
- [20] G. H. Loh and D. S. Henry. Predicting Conditional Branches with Fusion-Based Hybrid Predictors. In *Proceedings of the 11th International Conference on Parallel Architectures and Compilation Techniques*, Charlottesville, VA, USA, September 2002.
- [21] S. Manne, A. Klauser, and D. Grunwald. Branch Prediction using Selective Branch Inversion. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pages 81–110, Newport Beach, CA, USA, October 1999.
- [22] S. McFarling. Combining Branch Predictors. TN 36, Compaq Computer Corporation Western Research Laboratory, June 1993.
- [23] P. Michaud, A. Seznec, and R. Uhlig. Trading Conflict and Capacity Aliasing in Conditional Branch Predictors. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 292–303, Boulder, CO, USA, June 1997.
- [24] R. Nair. Dynamic Path-Based Branch Correlation. In *Proceedings of the 28th International Symposium on Microarchitecture*, pages 15–23, Austin, TX, USA, December 1995.
- [25] E. Perelman, G. Hamerly, and B. Calder. Picking Statistically Valid and Early Simulation Points. In *Proceedings of the 2003 International Conference on Parallel Architectures and Compilation Techniques*, pages 244–255, New Orleans, LA, USA, September 2004.
- [26] S. Reches and S. Weiss. Implementation and Analysis of Path History in Dynamic Branch Prediction Schemes. In *Proceedings of the 1997 International Conference on Supercomputing*, pages 285–292, Vienna, Austria, July 1997.
- [27] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides. Design Trade-offs for the Alpha EV8 Conditional Branch Predictor. In *Proceedings of the 29th International Symposium on Computer Architecture*, Anchorage, AK, USA, May 2002.
- [28] J. P. Shen and M. H. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw Hill, 2005.
- [29] K. Skadron, M. Martonosi, and D. W. Clark. Alloyed Global and Local Branch History: A Robust Solution to Wrong-History Mispredictions. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pages 199–206, Philadelphia, PA, USA, October 2000.
- [30] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt. The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 284–291, Boulder, CO, USA, June 1997.
- [31] J. Stark, M. Evers, and Y. N. Patt. Variable Length Path Branch Prediction. *ACM SIGPLAN Notices*, 33(11):170–179, 1998.
- [32] R. Thomas, M. Franklin, C. Wilkerson, and J. Stark. Improving Branch Prediction by Dynamic Dataflow-based Identification of Correlated Branches from a Large Global History. In *Proceedings of the 30th International Symposium on Computer Architecture*, pages 314–323, San Diego, CA, USA, May 2003.
- [33] M. Yousif, W. Dally, Y. Patt, J. Rattner, S. Scott, and A. Gonzalez. Bridging the Research Gap Between Academia and Industry. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, Madrid, Spain, February 2004.