The Effects of Context Switching on Branch Predictor Performance

Michele Co and Kevin Skadron
Department of Computer Science
University of Virginia
{mc2zk, skadron}@cs.virginia.edu

Abstract

This paper shows that context switching is not a significant factor to be considered when performing general branch prediction studies. Branch prediction allows for speculative execution by increasing available instruction level parallelism (ILP) and hiding the time required to resolve branch conditions. Accurate simulation of branch prediction is important because branch prediction strongly influences the behavior of processor structures. For this study, a timesharing framework was developed by modifying SimpleScalar's branch predictor simulator. A thorough characterization of the effects of branch predictor configuration, branch predictor area, and time slice length is provided. As further verification, branch predictor performance with and without flushing the predictor structures is compared. Experiments show that operating system context switches have little effect on branch prediction rate when using time slices representative of today's operating systems. Our findings show that this results from the fact that time slices are much larger than the training time required by the branch predictor structures. For all predictor configurations tested, the predictors train in under 128K instructions with or without flushing the branch predictor structures.

1 Introduction

Techniques for taking advantage of parallelism are imperative to realize the full potential of contemporary microprocessors. Branch prediction is one of the most powerful and widely-used techniques, which guesses the outcome of a branch before its condition is resolved. This avoids interruptions in instruction fetch and expands the window of instructions over which ILP can be exposed. Because the average size of basic blocks is relatively small (4-10 instructions), ILP within a block is limited. Therefore, accurate branch prediction is critical to higher performance, and it has been suggested that prediction accuracy is the biggest single architectural lever over performance in uniprocessors [11]. For these reasons, branch prediction techniques have been analyzed from many different perspectives in order to characterize branch behavior or evaluate new predictors. However, most branch prediction experiments have been performed in isolation from system-level features, in particular from the effects of context switching due to multiprogrammed workloads.

In a typical system, the branch predictor structures are time-shared between active processes. Between successive time slices for a particular process, intervening processes also use the branch predictor and therefore may affect branch predictor decisions for the process by mapping branches from multiple processes to the same predictor entries. This aliasing, similar to aliasing among branches in the same program, can have destructive, constructive, or neutral effects. Destructive aliasing causes the predictor to make an incorrect prediction, when originally a correct prediction would have been made. Constructive aliasing causes the predictor to make a correct prediction, when originally a misprediction would have been made. In neutral aliasing no change in prediction occurs. Context switching, by increasing the number of branches accessing individual entries, may increase the probability that branch predictor entries suffer destructive aliasing. This makes it important to understand the impact of context switching on branch prediction accuracy.

Current research has not systematically explored what effect realistic time slice lengths might have on branch predictor performance. Our work characterizes the effects context switching has on the performance of various branch predictor configurations and provides more comprehensive data on a wider range of predictor configurations and predictor areas than previous research. We explain the effects of context switching on branch predictor performance by showing that the training time required by all branch predictors is much smaller than current time slices. In this survey, we use a novel instruction-level simulation approach in which context switching can be modeled without requiring trace stitching. We find that:

- Branch predictor components train in under 128K instructions for CPU-bound workloads. This training time is shorter than realistic time slice lengths.
- For all but the shortest time slices, even the unrealistic behavior of flushing has negligible effect on predictor accuracy.
- Given the previous two findings, this means that branch predictor accuracy in the face of context switching with realistic time slices for CPU-bound processes is minimally affected. (This is not to say that context switching has zero effect, but rather, that with respect to general branch prediction studies, the effect is small and affects different predictors similarly.)
- With respect to general branch prediction studies, context switching is not as significant a factor as previously thought.

Taken together, the results in this paper provide the data necessary to help researchers choose simulation technique and accurately model the effects of branch prediction.

The paper is organized as follows. Section 2 describes necessary background information. Section 3 discusses related work. Section 4 describes our simulation technique. Section 5 discusses and analyzes experimental results. Section 6 summarizes our findings.

2 Background

For this study, four predictor configurations are studied: bimodal, GAs, PAs, and hybrid. The bimodal or "two-bit counter" scheme, proposed by Smith [22], contains a table of two-bit saturating counters or pattern history table (PHT), which is indexed by the branch address to provide the direction prediction. GAs and PAs are types of two-level predictors proposed by Yeh and Patt [27] and also Pan, So and Rahmeh [18]. Two-level predictors address the inability of bimodal predictors to identify branch pattern behavior or inter-branch correlation. The GAs prediction scheme uses global history to make direction predictions. It consists of a global history register (GBHR) which maintains a history of the outcomes of recent branches encountered and a table of two-bit saturating counters (the PHT). The GBHR is combined with the branch address to index the PHT and derive the direction prediction. The PAs prediction scheme uses per-branch history to make direction predictions. It consists of a table of per branch or local history patterns (branch history table or BHT) and a PHT. The branch address indexes the BHT. This yields a history pattern which is combined with the branch address to index the PHT and derive a direction prediction. Finally, hybrid prediction, proposed by McFarling [13], addresses the fact that different branches are predicted better with different predictor organizations. A hybrid predictor combines two predictor components from those previously described above and adds a selector component which selects the final direction prediction from the two predictions provided by the components.

Context switches exist to permit multiprogramming. The processor time is divided between active processes; each process receives a maximum slice of time on the processor to perform computations after which it must relinquish the processor, regardless of whether the process has fully completed or not. This period of time is called the time slice. Context switches are important to branch prediction because branch predictor structures are shared between the running processes. Intervening processes can cause destructive aliasing in the branch predictor structures, and if the time slice is very small, the branch predictor might not have enough time to properly train before the next context switch. Context switches can be categorized into two main categories: voluntary and involuntary [10]. Voluntary context switches occur when a process willingly relinquishes the processor due to system calls, I/O, or page faults. Involuntary context switches occurs when the time slice has elapsed and the process is forced to relinquish the processor. In this study, we simulate both involuntary and voluntary context switches. Typical time slices are 25 ms for Windows NT and 50-200 ms for Unix varieties[1].

[1]Default time slice lengths for Windows NT on various platforms are listed in [23]. Unix time slice lengths are based on the output of Solaris 5.5.1 *dispadmin* utility [15] and examination of RedHat Linux 6.1 source code [19].

3 Related Work

Several previous branch prediction studies have considered context switching, but they have used very short time slices. Our work simulates a range of realistic time slices for a broad range of predictor configurations and predictor areas. We focus on benchmarks that are primarily CPU-bound and our results are directly useful to researchers using the SPEC benchmarks.

Gloy, *et al.* [7] analyzed dynamic branch prediction schemes on system workloads. They quantify the effects of kernel and user interactions on branch prediction accuracy. They argue that modeling context switching by flushing the branch predictor structures after each context switch is unrealistic because interactions with the kernel and other processes do not necessarily flush branch predictor state. In their study, kernel level branches were integrated with user level branches for each benchmark analyzed. They found that inclusion of kernel level branches significantly affected the branch predictor rate when the kernel level branches accounted for more than 5% of the total executed instructions. Time slices used in their experiments were small, however: 10K, 100K and 1M instructions. Our work more systematically studies the relationship among predictor organization, size, and time slice length, and uses more typical time slices. We also verify that flushing, although not representative of realistic behavior in CPU-bound workloads, minimally affects branch predictor accuracy in the face of longer, realistic time slice lengths.

Calder, Grunwald, and Emer [2] performed a system-level analysis of the performance of various branch architectures on the Alpha 21064 [3]. Their work demonstrated that branch performance metrics should include how overall system performance is affected. Our work, however, focuses on whether the modeling of multiprogramming affects predictor behavior. Since we find that it typically does not, it is unnecessary (and prohibitively expensive) to obtain IPC measurements.

Similar work by Mogul and Borg [16] has explored caching in the face of context switching. They examined how cache hit rate varies after a context switch and noted that the cost of context switches, in terms of how it affects cache performance, can guide cache design. Hwu and Conte [10] studied the worst-case susceptibility of programs to context switching. They study how voluntary and involuntary context switches affect various benchmarks with respect to cache miss ratio.

Many studies have evaluated specific branch predictor choices in light of context switching. Evers, Chang and Patt [5] proposed the multi-hybrid predictor, which contains at least one quick-training component (i.e., a bimodal component) which is used when context switches occur, giving other component structures of the multi-hybrid predictor time to train. They found the multi-hybrid to be more accurate than conventional hybrid configurations in the face of context switching. Context switching was modeled by flushing the predictor structures and setting the structures back to their initialized state at intervals of 16K, 64K, 256K, and 1M instructions. However, these intervals are unrealistically short by today's standards and exaggerate the impact of destructive interference. Our work shows that even if the branch predictor structures are flushed at every context switch, the impact is minimal for all but the smallest time slice lengths. We also present more realistic results in which we interleave the execution of several processes rather than perform single-benchmark simulations.

Juan, Sanjeevan and Navarro [12] explored dynamic history-length fitting (DHLF), where they showed that different types of two-level predictors perform differently depending on the code, input data and the frequency of context switches, and developed a method for dynamically selecting history length to adjust to the current workload. They cite the conventional wisdom that information in the PHT is periodically lost with every context switch and that larger PHTs and longer history lengths require longer warmup times as possible reasons why predictors with shorter training times have a higher prediction accuracy after a context switch. In their study, they used time slices of 8K through 256K dynamic conditional branches on the SPECint95 benchmarks. This translates to time slices of approximately 40K-1.25M instructions. They show that a *dhlf-gshare* scheme outperforms the corresponding same-area gshare predictor with a context switch distance of 70K conditional branches and a step value (the value at which history length may be adjusted based on observed behavior) of 16K conditional branches. Our work shows that 128K instructions is sufficient to properly train the PHT for CPU-bound processes.

Eden and Mudge [4] show the performance of various branch predictor schemes compared to YAGS in the presence of context switches. For their experiments, they used a very short time slice of 60K instructions in order to demonstrate the worst-case severity of small time slices on various anti-aliasing branch predictor configurations. Their data shows that for YAGS, bimode, skew, and gshare, as the area of the predictor increases, the prediction rate for the various predictor types converges between predictor areas of 10 KB and 100 KB.

Gummaraju and Franklin [9] studied the problems involved with performing accurate branch predictions in single-program multi-threaded processors (SPMT). In SPMT, a compiler takes a single sequential program and divides it into groups of up to 8 parallel threads which can be executed in parallel. The focus of our research is processors with multiprogrammed workloads, which differs from SPMT processors. The authors note that for private predictors there is a "cold start" time for each thread, after which the prediction rate improves. This result matches our finding, that branch predictors train to a plateau in prediction performance within 128K instructions.

Nair [17] discussed dynamic path-based correlation and studied the robustness of branch prediction in the face of context switches. Context switches were simulated by periodically flushing the branch predictor structure. Nair found that as the time slice length increased, programs suffered less of a penalty for the flushing. This result was program-dependent but a plateau was reached at approximately 100K instructions, again matching our findings.

Yeh and Patt [27] explored alternative implementations of two-level adaptive branch prediction. They discuss the effect of context switching on the prediction performance of GAg, PAg, and PAp two-level predictors. The time slice used was 500K instructions. Context switches were simulated by flushing branch predictor structures. They showed that these predictor implementations suffered a degradation of less than 1% despite the effects of flushing.

Despite the fact that many studies have examined context switching or have considered other system-level effects, our work approaches context switching more systematically and completely. The simulator we developed allows dynamic interleaving of processes and thus allows us to study a variety of workloads and time slices. This allows us to thoroughly characterize the interaction between time slice length, predictor organization, and predictor area as it affects branch predictor accuracy. We show that with realistic time slices and a CPU-bound workload, all predictor types train within 128K instructions, which is much shorter than realistic time slice length. This explains our finding that for time slices longer than 781K instructions branch predictor accuracy is minimally affected.

4 Simulation Technique

Simulation Framework. Experiments were performed using the SimpleScalar 3.0 Toolkit [1] and MPI 2.0 (Message Passing Interface) [14]. SimpleScalar's branch predictor simulator, *sim-bpred*, was modified to use shared memory segments for all branch predictor structures, so that the predictor may be shared between the various processes. One copy of the simulator is spawned for each benchmark that is part of the workload to be simulated. Any number of processes may be interleaved. MPI was only used to control multiple processes for interleaving and to synchronize the sharing of the branch predictor. The active process requires ownership of a token to the branch predictor. Ownership of the branch predictor token is limited to one process at a time. During the initialization phase of *sim-bpred*, the first process spawned creates the shared branch predictor and the remaining processes attach to the predictor structure created. Context switching to the next process is accomplished by passing the token and is performed on the specified number of instructions corresponding to the time slice's expiration or upon system calls which require I/O. This means that number of instructions until a context switch is performed is potentially shorter than the specified time slice length for the experiment. When the first process has fully completed its execution, it detaches from the shared predictor, and subsequent processes each receive one more time slice before the simulation is terminated and the shared predictor is deallocated. This termination policy prevents the statistics from skew arising from one process executing much longer than the others.

Since CPU-bound benchmarks were selected for this study, round-robin scheduling was selected. Most popular operating systems, such as UNIX and Windows NT, use multilevel feedback queues or priority with aging for process scheduling. In both of these scheduling schemes, processes with the same priority are serviced in round-robin fashion [20].

Because different operating systems have differing average time slices, a time slice from the shorter end of the range was chosen. Longer time slices will only reduce the impact of context switching on branch predictor accuracy. Windows NT has an average time slice of 25 ms [23], while various flavors of Unix have a time slice which ranges from 50-200 ms [15, 19]. Some operating systems vary the time slice according to the process priority, but generally CPU-bound processes have time slices in the 100-200 ms range.

As processor clock speeds continue to increase, the number of instructions completed in an average context-switch interval will increase. For simplicity, our simulator specifies time slices in numbers of instructions. We assume a 25 ms time slice with an average IPC of 1.75-2 and a clock rate of 1 GHz. This corresponds to roughly 50M instructions per time slice, which we use as our baseline time slice. For our simulations, if a system call requiring I/O was encountered during the course of the time slice, a context switch was performed immediately. In all other cases, the full time slice length was used.

Benchmarks. Eight CPU-bound benchmarks were selected as the workload to be studied. Six benchmarks from SPECint95 [24] were chosen: compress, gcc, go, ijpeg, perl, and xlisp. In addition, radiosity from SPLASH-2 [26] and gnuchess from the IBS suite[25] were selected to make up the system workload. These benchmarks were started simultaneously and given access to the branch predictor in round-robin fashion. Equal time slices are given to each process, but a process is required to yield the time slice upon any system call requiring I/O. The static and dynamic branch footprints for the sections of the benchmark which were tested are shown in Figure 1. The benchmarks were compiled for SS PISA [1] using gcc v. 2.6.3. Linking is static, so the benchmarks include all library code.

All results are presented in terms of the prediction accuracy for conditional branch directions. The SPECint benchmarks run for many billions of instructions, which is prohibitive to simulate to completion. In addition, most benchmarks contain some initial behavior which is not representative of the general behavior of the program.

Jumping forward to the desired location in the program to begin collecting simulation statistics saves time. This is known as fastforwarding. Fastforwarding involves selecting a portion of the instructions which will be executed in "fast" mode in which some aspects of the simulation are left out in order to save time in reaching the desired instruction from which statistics will be collected. Work has been done by Skadron, Martonosi, and Clark [21] to determine reasonable fastforward intervals that skip unrepresentative behavior. The benchmarks used in this study were each fastforwarded according to the intervals shown in Table 1 before beginning to collect statistics. Branch predictors were always updated during the last 50M instructions of the fastforwarding period to provide a warmed up predictor. This is roughly equivalent to a 25 ms time slice.

Benchmark	Input	Stat. Br. Ct.	Dyn. Cond. Br. Ct.	I/O Calls	Fastfwd	#Inst Sim. after FF
compress	ref.in	234	441,043,245	0	1.648B	3,271,875,000
gcc	sim-outorder.i	19,596	467,603,714	27	220M	3,254,233,938
gnuchess	gnuchess.in2	805	306,147,429	65	150M	3,168,768,290
go	9stone21.in	5658	368,952,908	0	925M	3,271,875,000
ijpeg	vigr.ppm	1415	190,739,643	0	823M	3,271,875,000
perl	scrabble.pl	675	424,129,244	0	600M	3,271,875,000
radiosity	-p 1 -batch -room	183	302,859,859	0	300M	3,271,875,000
xlisp	9queens.lsp xit.lsp	316	504,101,948	0	270M	3,269,584,722

Table 1: Inputs, fastforward, and simulation numbers for benchmarks. (Fastforward distances are taken from [21]).

Experimental Parameters. The parameters used in the experiments were time slice length, predictor type and predictor area. Time slices of 390K, 781K, 3.125M, 12.5M, 50M, and 200M instructions were used. GAs, PAs, bimodal and hybrid predictor (bimodal-GAs hybrid) types were examined using areas of 2, 4, 8, 16, 32, 64, and 128 Kbits. Predictor area was calculated as the total number of bits used for the BHT, PHT, and selector. Specific predictor configurations are listed in Tables 2 and 3. Due to the large design space of hybrid branch predictors, only the configurations listed in Table 3 are studied.

Predictor Area	Bimodal L1 Ent.	GAs L2 Ent.	GAs Hist. Width	PAs L1 Ent.	PAs L2 Ent.	PAs Hist. Width
2 Kbits	1K	1K	5	512	512	2
4 Kbits	2K	2K	5	512	1K	3
8 Kbits	4K	4K	6	1K	2K	4
16 Kbits	8K	8K	6	1K	4K	8
32 Kbits	16K	16K	7	2K	8K	8
64 Kbits	32K	32K	7	4K	16K	8
128 Kbits	64K	64K	8	8K	32K	8

Table 2: Bimodal, GAs, PAs branch predictor configurations

Predictor Area	Hybrid Comp. 1 L1 Ent.	Hybrid Comp. 1 L2 Ent.	Hybrid Comp. 1 Hist. Width	Hybrid Comp. 2 Bimod. Ent.	Hybrid Selector Ent.
2 Kbits	128	128	2	512	256
4 Kbits	256	256	2	1K	512
8 Kbits	512	512	2	2K	1K
16 Kbits	512	1K	3	4K	2K
32 Kbits	1K	2K	4	8K	4K
64 Kbits	2K	8K	8	8K	8K
128 Kbits	8K	16K	8	8K	8K

Table 3: Hybrid branch predictor configurations

5 Experimental Results

Overview. Our experiments show that when using a contemporary time slice length, branch prediction direction rate is minimally affected compared to when very short time slices are used. Misprediction rate was chosen over IPC as a metric because it has been found that IPC simulations are highly sensitive to configuration parameters [6]. Our survey seeks to avoid tying the predictors to particular configuration parameters.

The baseline for comparison in our experiments is the prediction rate of each benchmark when executed as a single process and run to completion with no intervening processes. Each benchmark is executed for the same number of instructions (shown in Table 1) for all experiments. We always use the following order: gcc, compress, perl, go, ijpeg, xlisp, radiosity, gnuchess. Other orderings were tested but in all cases we found that this changes prediction rate by less than 0.15%. The only exception was ijpeg which performed up to 1.45% worse for some orderings due to the fact that it has a longer average basic block size and therefore requires more time to train; and obtains some constructive aliasing when following go.

Our survey of the effects of time slice length on branch predictor organization and branch predictor area are summarized in Figure 1, which plot the results for GAs, PAs, bimodal, and hybrid results respectively. The horizontal lines in each graph in Figure 1 gives the misprediction rate as a function of time slice length for one predictor area; 2, 4, 8, 16, 32, 64, and 128 Kbit predictor areas were surveyed. The flatter the line, the lesser the effect of context switching on branch predictor accuracy. The gaps between the lines show the improvement in misprediction rate due to increasing predictor area. Naturally, these gaps are the same when context switching has no effect, because under these conditions, each line is relatively flat. This does not mean that context switching has no effect. Rather, the small variances in the branch prediction rate between simulating context switching and not simulating context switching do not affect the overall relationship of how various branch predictor configurations perform relative to one another.

Graphs displaying flushing results show that for all predictor types, when using a realistic time slice such as 50M instructions or greater, flushing has negligible effect. Flushing shows more of an effect at 390K and 781K instructions (except bimodal), but these time slices are much shorter than contemporary time slices.

The general shape of the curves between predictor organizations is mostly similar; all the curves are nearly flat for time slices longer than 3.125M instructions. This shows that context switching has very little effect on branch predictor accuracy. At short time slice lengths (up to 3.125M instructions), flushing increases the misprediction rate on GAs, PAs, and hybrid predictors. Bimodal predictors do not suffer from any significant change in misprediction rate even with flushing. This result shows that for realistic time slices, flushing minimally impacts branch predictor accuracy.

Overall, for contemporary time slices (longer than 50M), the difference between the observed misprediction rate with context switching and the baseline misprediction rate is minimal(the lines are flat). Our training time experiments find that branch predictor structures train within 128K instructions, which is much smaller than these realistic time slices, and this explains the insensitivity due to context switching.

The remainder of this section discusses the following: 1) the results of giving one process a longer time slice than other processes, 2) the relationship between static branch footprint and branch predictor performance, 3) the number of branches at which branch prediction rate stabilizes, and 4) the upper bound of predictor training time for primarily CPU-bound workloads.

Larger Time Slice. Since predictors train rapidly, and we have already seen that branch prediction is insensitive to context switching. We would expect that giving different time slice lengths to different programs should have no effect. Experiments in which one process was given a longer time slice than the other running processes were conducted on 16 Kbit GAs, 16 Kbit PAs, and 64 Kbit hybrid predictors. In separate simulations, each process was given a time slice of 150M instructions, which was three times the time slice given to the rest of the processes. Generally for all predictor types, the process which was given more of a time slice had a prediction rate which was slightly better (less than 0.5%) than when it was given an equal time slice as all the others. Indeed, the longer time slice had minimal effect. These small differences are due to decreased constructive aliasing when receiving more of a time slice. Gcc encounters a 2% decrease in prediction accuracy when xlisp receives more of a time slice and a 1.8% decrease when gnuchess receives more of a time slice. This occurs because xlisp and gnuchess have relatively high dynamic conditional branch counts, so when they receive more of a time slice they cause more destructive interference for gcc's large static footprint. Curiously, gcc and go have no interaction.

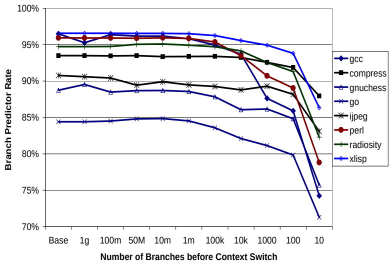

Figure 2: Context switching on a GAs predictor by number of branches

Large Static Branch Footprint versus Small Static Branch Footprint. The benchmarks used have very different static branch counts (see Table 1). For example, gcc has a very large static branch footprint of 19,704 branches, with go having an intermediately sized static branch footprint of 5094, and compress having a relatively small footprint of 233 static branches.

To isolate the effect of branch footprint size we conducted two experiments to compare a two-program workload in which both benchmarks have large static branch footprints to a workload in which one benchmark has a large static footprint and the other has a small static branch footprint. Each experiment consists of a workload of two processes with a time slice of 50M instructions run on 4 Kbit bimodal, 16 Kbit PAs, 16 Kbit GAs, 64 Kbit hybrid predictors. Resulting prediction rates are compared to baseline prediction rates. In the first experiment, a process with large static footprint is run concurrently with another process with a large static footprint (gcc and go). Gcc's prediction rate for all predictor types is between 0.01%-0.03% worse than the baseline. This is a result of destructive aliasing. Go's prediction rate improves for all predictor types between 0.29%-0.7%. This is a result of constructive aliasing between gcc and go. These variances in the prediction rate are tiny, of course.

In the second experiment, a process with a large static footprint is run concurrently with another process with a small static footprint (gcc and compress). Gcc's prediction rate matches its baseline prediction rate, while compress's prediction rate decreases by 0.08%-0.1%. This means that compress suffers from a tiny amount of destructive aliasing due to gcc's branches.

Overall, as with varying time slice, the size of the static branch footprint is also not important.

Transition Point in Branch Predictor Performance. In order to determine the boundary point at which predictor performance reaches a plateau or begins to

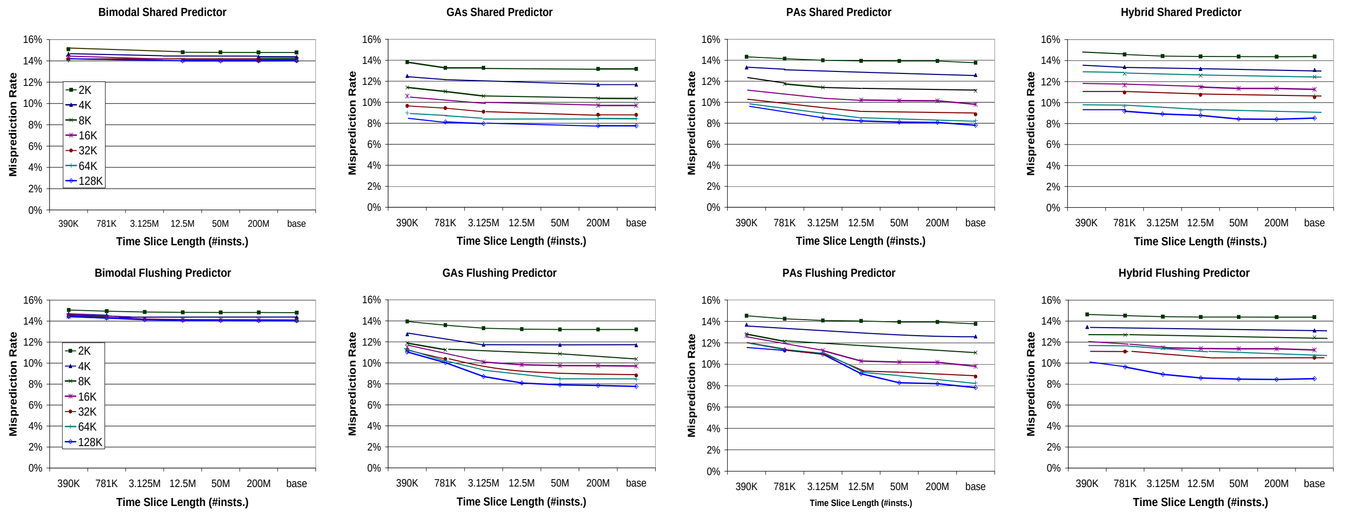

Figure 1: Misprediction rate as a function of time slice length and predictor area w/out flushing (top) and w/flushing (bottom). Left to right: Bimodal, GAs, PAs, hybrid.

degrade, we conducted experiments using context switching based on the number of branches encountered. Context switches were made every X branches on a base-10 exponential scale (10, 100, ... 1G) and the branch prediction rate was noted for each process. Figure 2 shows the change in prediction accuracy by context switching on various numbers of branches for each benchmark for the 16 Kbit GAs predictor. The results for other predictor types were similar. For most of the benchmarks there is a distinct "knee" in the curve at 100 branches after which the prediction rate quickly plateaus and approaches the baseline prediction rate.

The plateau is reached between 1K and 100K branches, which is far shorter than realistic time slices. A basic block is 4-10 instructions on average, so this graph shows that the training time necessary for branch predictor structures is between 4K-1M instructions. The following section narrows the range found here and shows that the training time for most predictor structures studied to be no more than 128K instructions.

Branch Predictor Training Time Evaluation. Experiments to capture the training time of the branch predictor structures were performed. For each time slice length, predictor type (GAs, PAs, and bimodal only), and predictor size, misprediction rates and PHT occupation by each process were tracked during the course of the context switch interval. These statistics were gathered on an increasing base-2 log scale starting from 1 through 128K instructions. Results show that for the largest predictor size (hence potentially the longest training time) and smallest time slice for all predictor types (hence maximum impact of context switching), the time to reach steady state for conditional branch misprediction rate is approximately 128K instructions. These results are shown for the largest predictor because a large predictor has more entries to fill, it may take longer to train, since there will be less aliasing than occurs in a smaller predictor. Results for 2K predictor area for a time slice length of 390K were similar to the results for the 128K predictor area, with all benchmarks' misprediction rates reaching steady state by 128K instructions (less than 1K instructions for compress, which has a very small static branch footprint).

This result explains the results of some small pilot experiments with flushing carried out at the University of Maryland by Gummaraju [8] which found that flushing has very little effect on prediction rate. Our experiments were run both with and without flushing to determine whether inter-process aliasing affects results. When flushing was used, the branch predictor's PHT and BTB were re-initialized to random values at the beginning of each time slice. Results show that prediction rates without flushing are slightly better, due to retained state and due to constructive aliasing. Data is shown for gcc and compress for the 128Kb GAs predictor and 390K instruction time slice.

The graphs in Figure 3 plots the average instantaneous misprediction rate over the course of a 390K instruction time slice. The misprediction rate plot is divided into mispredictions made by PHT entries owned by the process (self), PHT entries owned by another process (other), and PHT entries which are untouched by any process (empty). Data was collected at an exponentially (base 2) increasing number of instructions from the beginning of each time slice and was also collected at the end of the time slice. The data was then averaged across time slices to create summary graphs.

From Figure 3 it is clear that even for the largest predictor area studied, GAs has misprediction rates that reach a plateau within 128K instructions for the gcc and compress. Even for flushing, the misprediction rate for gcc and compress also stabilizes by 128K instructions. Without flushing, the graphs show that the misprediction rate stabilizes sooner, by 64K instructions for gcc and 32K instructions for compress. Compress trains the predictors more quickly due to its small static branch footprint. The falling curve for gcc for *empty* predictions in flushing and *other* predictions in non-flushing is due to its changing branch footprint. PAs and bimodal have graphs that are very similar, and also stabilize by 128K instructions. In terms of flushing or not flushing the predictors, there is no significant difference between flushing and not flushing in terms of total misprediction rate (summed across the three curves). This is because any interference in the PHT caused by other processes basically behaves the same as an entry that is initialized to a random value.

The PHT occupation graphs in Figure 4 track percentage of PHT entries by ownership (self, other, empty) during the course of a time slice. The slowly rising curve for *self* entries is again due to gcc's changing footprint. For gcc with flushing, over the course of a time slice the number of PHT entries starts at 0% and rises to 5% by the end of the time slice. For compress, since it has such a small static branch footprint, the percentage of the PHT occupied is negligible throughout the course of the time slice. Without flushing, gcc starts with 41% of the PHT entries and over the course of the time slice, rises to 42%. Compress starts with 8% of the PHT and remains steady at this level throughout the time slice. The large difference in PHT occupation between flushing and not flushing results from retained state. Despite context switching, some state in the PHT is being retained due to the large area of the branch predictor. However, much of this state is generally unused by gcc during the course of its execution. Furthermore, the difference in total misprediction rate between flushing and not is negligible.

The combination of the misprediction rate and PHT occupation graphs in Figures 3 and 4 shows that regardless of the size of a benchmark's branch footprint, the predictor structures train quickly, within 128K instructions.

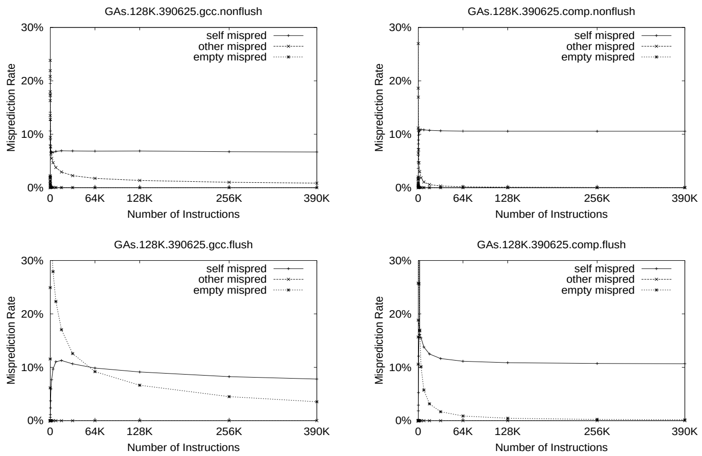

Figure 3: Misprediction rates for 128K GAs predictor w/out flushing (top) and w/flushing (bottom) using a 390K instruction time slice for gcc (left) and compress (right)

6 Conclusions and Future Work

This work provides a thorough characterization of the interaction between branch predictor configuration, branch predictor size, and time slice length. Experimental results show that for time slices common in contemporary operating systems and contemporary hardware, context switching has little effect on branch predictor performance. Although prediction accuracy improves with increasing predictor area, the improvement is not related to context switching, but rather a matter of alias reduction during each process' time slice.

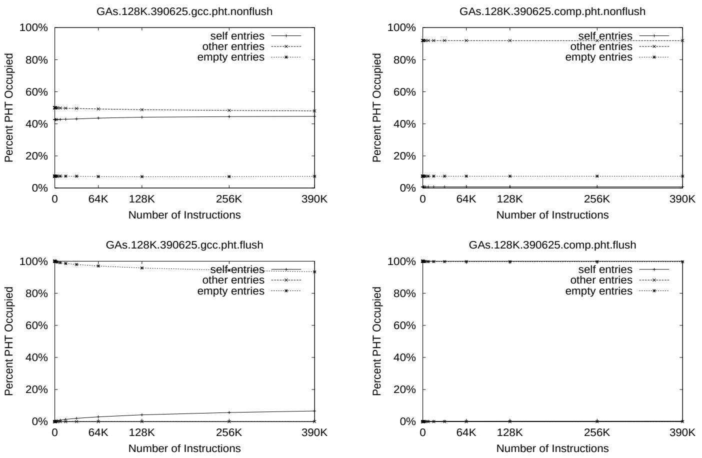

Figure 4: PHT occupation for 128K GAs predictor w/out flushing(top) and w/flushing (bottom) using a 390K instruction time slice for gcc (left) and compress (right)

Specifically, this paper shows that:

- Branch predictor components train in under 128K instructions for CPU-bound workloads. This training time is shorter than realistic time slice lengths.
- For all but the shortest time slices, even the unrealistic behavior of flushing has negligible effect on predictor accuracy.
- Given these two findings, this means that branch predictor accuracy in the face of context switching with realistic time slices for CPU-bound processes is minimally affected.
- Even when context switching changes branch prediction rate slightly, it does not affect the impact of changing branch predictor area or organization, which behaves like an offset. This does not apply for the smallest time slices, but these are unrealistically small.
- Despite the assumption of an eight-process workload, the effect of context switching is negligible. For a fewer-process workload, the prediction accuracy will approach that of a single-process workload.
- Whether or not flushing is used, the impact is negligible for realistic time slices.

Taken together, the results in this paper provide the data necessary to help researchers choose simulation technique and accurately model the effects of branch prediction.

Processor clock speeds will continue to increase, therefore the number of instructions completed per time slice will increase. Therefore, context switching in multiprogrammed environments with CPU-bound processes is not likely to have a significant effect on the prediction accuracy of today's branch predictors and need not be a factor included in general branch prediction studies.

As part of our future work, we hope to measure the amount of conflict mispredictions that occur in multiprogrammed workloads.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation under grant no. CCR-0082671. The authors would also like to thank Douglas Clark, John Haskins, Jason Hiser and the anonymous reviewers for their valuable feedback and insights.

References

[1] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997. http://www.simplescalar.org.

[2] B. Calder, D. Grunwald, and J. Emer. A system level perspective on branch architecture performance. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 199–206, Dec. 1995.

[3] Digital Semiconductor. *DECchip 21064/21064A Alpha AXP Microprocessors: Hardware Reference Manual*, June 1994.

[4] A. N. Eden and T. Mudge. The YAGS branch prediction scheme. In *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, pages 69–77, Dec. 1998.

[5] 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 Annual International Symposium on Computer Architecture*, pages 3–11, May 1996.

[6] J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich. FLASH vs. (simulated) FLASH: Closing the simulation loop. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[7] N. Gloy, C. Young, J. B. Chen, and M. D. Smith. An analysis of dynamic branch prediction schemes on system workloads. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 12–21, May 1996.

[8] J. Gummaraju. Personal communication, Oct. 2000.

[9] J. Gummaraju and M. Franklin. Branch prediction in multi-threaded processors. In *Proceedings of the 2000 International Conference on Parallel Architectures and Compilation Techniques*, pages 179–188, Oct. 2000.

[10] W. W. Hwu and T. M. Conte. The susceptibility of programs to context switching. *IEEE Transactions on Computers*, 43(9):994–1003, Sept. 1994.

[11] N. P. Jouppi and P. Ranganathan. The relative importance of memory latency, bandwidth, and branch limits to performance. In *The Workshop on Mixing Logic and DRAM: Chips that Compute and Remember*, June 1997. http://ayer.CS.Berkeley.EDU/isca97-workshop.

[12] 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 Annual International Symposium on Computer Architecture*, pages 156–66, June 1998.

[13] S. McFarling. Combining branch predictors. Tech. Note TN-36, DEC WRL, June 1993.

[14] Message Passing Interface Forum. *MPI-2: Extension to the Message-Passing Interface*, July 1997.

[15] Sun Microsystems. Solaris 7 reference manual collection. http://solaris.license.Virginia.EDU:8888/. dispadmin -c TS -g.

[16] J. C. Mogul and A. Borg. The effect of context switches on cache performance. Tech. Note TN-16, DEC WRL, Dec. 1990.

[17] R. Nair. Dynamic path-based correlation. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 15–23, Dec. 1995.

[18] S.-T. Pan, K. So, and J. T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 76–84, Oct. 1992.

[19] J. Regehr. Personal communication, June 2001.

[20] A. Silberschatz and P. B. Galvin. *Operating System Concepts.* John Wiley and Sons, Inc., 5th edition, 1999.

[21] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. Branch prediction, instruction-window size, and cache size: Performance tradeoffs and simulation techniques. *IEEE Transactions on Computers*, 48(11):1260–81, Nov. 1999.

[22] J. E. Smith. A study of branch prediction strategies. In *Proceedings of the 8th Annual International Symposium on Computer Architecture*, pages 135–48, May 1981.

[23] D. A. Solomon. *Inside Windows NT.* Microsoft Press, 2nd edition, 1998.

[24] Standard Performance Evaluation Corporation. SPEC CPU95 Benchmarks. http://www.specbench.org/osg/cpu95.

[25] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Instruction fetching: Coping with code bloat. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 345–56, June 1995.

[26] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.

[27] T.-Y. Yeh and Y. N. Patt. Alternative implementations of two-level adaptive branch prediction. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 124–34, May 1992.