Mispredicted Path Cache Effects

Jonathan Combs, Candice Bechem Combs, and John Paul Shen

Carnegie Mellon Microarchitecture Research Team (CMuART)
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{jcombs, cbechem, shen}@ece.cmu.edu

Abstract. As superscalar pipelines become wider and deeper, the percentage of dynamic instructions fetched into the machine from the mispredicted path significantly increases. This paper discusses how a new cycle-accurate performance simulator is used to accurately measure mispredicted path effects on the cache hierarchy. Previously published results based on less accurate tools indicated that mispredicted path instructions have the serendipitous positive effect of doing memory prefetching. Our results show that while such prefetching does occur for some benchmarks, it does not occur consistently for all benchmarks. Furthermore the IPC impact varies widely among the benchmarks. SPECint95 benchmarks show IPC changes ranging from -8% to +12%.

1 Introduction

The aim in building a wider and deeper pipelined microarchitecture is to enable the processor to execute more instructions in parallel, and at the same time achieve higher clock frequencies. One major design challenge is to keep issue rates high despite the fact that basic blocks are small and memory latencies are high. A design that contributes to a more uninterrupted instruction fetching and reduces the memory latency will help increase issue and completion rates with an overall gain in IPC (average instructions per cycle).

To achieve high instruction fetch bandwidth, modern microprocessors employ branch predictors to speculatively fetch instructions beyond conditional branches. If these speculative instructions are determined to be on a mispredicted path, they must be invalidated and removed from the machine.

Mispredicted instructions can affect many parts of the machine, particularly the functional units, branch predictors [5], and caches. This research focuses on the effects mispredicted path instructions have on the cache hierarchy, due to the increased number of instruction and data references. Previous research in this area showed that mispredicted path references have a prefetching effect, but the methods that measured these effects had serious limitations. Using our improved fMW [1,2] performance simulation tool we found that the magnitude of the effect varies with each benchmark and is not always positive.

Before we can accurately differentiate previous methodologies from our own, it is important to have a background on simulators. Functional simulators model the

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 1322-1331, 1999.
Springer-Verlag Berlin Heidelberg 1999

instruction set architecture (ISA). The primary concern of functional simulators is functional correctness. Performance simulators model the *micro*architecture. They model the machine organization and are concerned with machine performance. Sometimes these simulators are also referred to as cycle-accurate simulators to reflect their concern with timing issues. Traditionally, performance simulators are implemented as trace-driven tools; i.e. their inputs are traces of dynamic instructions, without full-function simulation capability.

Some of the new microarchitecture features, such as aggressive branch prediction, value prediction, and multi-path execution, cannot be accurately simulated in a trace-driven tool. To address these shortcomings, we use a new performance simulator with full-function capability called fMW (functional microarchitecture workbench) [1]. The new fMW builds on MW by incorporating a customized version of the PSIM [3] functional simulator and by extending the capabilities of the original MW [2].

2 Previous Work

A handful of studies [6,8,9] have examined the effects of mispredicted paths; however each of these efforts is hampered by inadequate modeling techniques. The simulator used in [8,9] is trace driven leading to several inaccuracies. Since trace-driven simulators can not execute mispredicted path instructions, Pierce and Mudge injected a fixed number of instructions to emulate the mispredicted path [8]. However, the number of cycles a given machine spends on each mispredicted path depends on the aggressiveness of the branch predictor and the branch resolution latency. Fixing the branch resolution latency at a constant number of instructions introduces significant error. Nevertheless, using this method, Pierce and Mudge found the mispredicted path instructions tend to prefetch the data cache.

A continuation work [9], using the same tool, focused on the instruction cache and found the prefetching effects of mispredicted path instructions far outweigh the pollution effects caused by them. Lee et al. studied instruction cache fetch policies using a cache simulator and found mispredicted path instructions did not cause any degradation in performance over fetching the correct path only [6].

Previous studies generally show mispredicted path execution to be a beneficial prefetching mechanism for the I-cache [6,8,9]. Similar benefit for the D-cache was also suggested [6]. However, the methods that measured these effects had serious limitations and are inherently inaccurate. Using fMW such inaccuracies are removed by directly simulating the mispredicted path instructions in the machine model. The following section summarizes our experimental results.

3 Experimental Methodology

All data reported in this paper are generated using the new fMW [1] tool which integrates a functional simulator and a cycle-accurate timing simulator that is built based on a validated PowerPC 604 model [2]. The machine model is based on published reports [4,7,10] and accurately models all aspects of the microarchitecture.

The machine model and the simulation framework used in this paper are discussed in the next two subsections.

Table 1. SPECint95 benchmarks

Name	Input Set	Instruction Count
compress	10000 e 2231	39,719,131
gcc	-f<all optimizations> -O regclass.I –s regclass.s	257,670,349
go	5 9	79,544,303
ijpeg	tinyrose.ppm	92,054,217
li	queen6.lsp	56,572,774
m88ksim	dhry.big.100iter, cache off	106,900,787
perl	trainscrabbl.in	50,039,056
vortex	tiny.in	153,084,257

3.1 The Machine Model

The SPECint95 benchmark suite is used for all experiments. The input sets and run lengths of each benchmark are summarized in Table 1. To focus the current study on the effects of speculative execution and to emphasize the effect of mispredicted path instructions in the pipeline, the PowerPC 604 microarchitecture is extended to remove resource constraints, and widened to allow a greater number of in-flight instructions. The instruction window is limited to 512 instructions with an unlimited number of functional units and rename registers. Instruction fetch and dispatch widths are increased to 16 instructions per cycle. There is no limit to the number of instructions that may complete in each cycle[1]. A 64-entry fully associative branch target address cache (BTAC) and a 512-entry branch history table (BHT) handle branch prediction.

The memory hierarchy includes a perfect main memory, a 32KB 4-way set associative Level-1 instruction cache (IL1), a 32KB 8-way set associative Level-1 data cache (DL1), and a 512KB 8-way set associative unified Level-2 cache (UL2). All caches use a write-back, write-allocate scheme. Access latencies are 1, 3, and 100 cycles for the L1, L2, and main memory respectively. An unlimited load-miss queue and an unlimited store queue handle all load and store execution. The store queue performs data forwarding, and load/store instructions execute out-of-order if no address aliasing is detected.

In terms of mispredicted branches and cache accesses made by mispredicted path instructions, there are three key stages that determine the effects of wrong path instructions. These stages are where different types of branch outcomes resolve in a PowerPC machine. Unconditional branches resolve in decode, branches to count or link registers resolve in dispatch, and conditional branches resolve in the branch execution unit within execution. The later the stage that a branch resolves, the greater number of mispredicted path instructions that are allowed into the machine. As soon as a mispredicted branch is resolved, however, the mispredicted path instructions are flushed from the machine. Since most branches are resolved before reaching the

[1] The authors are not proposing this as a realistic machine design, but a model that increases the effects of mispredicted path execution while enforcing register and memory data dependencies.

execution core, nearly all mispredicted path load accesses are flushed from the machine without ever affecting the caches. For those branches that do not resolve until reaching the execution unit, there is a limited window of opportunity for mispredicted path load accesses to execute and thus affect the DL1 and UL2.

The cache fetch policies process all cache misses. Given the limit study nature of this work, when mispredicted path loads are allowed to access the caches any complications due to memory mapped I/O are ignored. Also, the IL1 cache must become nonblocking. This is because correct path instruction fetching is allowed to continue immediately even if an I-cache miss is outstanding due to instruction fetches down the wrong path. This is referred to as the "Resume" policy as discussed in [6]. The Resume policy was determined to have the lowest miss ratio during speculative execution; thus one can infer this policy to have the most beneficial impact on IPC gain. In [6], however, only a single outstanding miss was allowed. The policy in this study is to allow unlimited outstanding misses.

3.2 Simulation Framework

To determine the exact effects mispredicted path instructions have on the cache hierarchy, there needs to be a record of what the accesses would have been if mispredicted path instructions were not simulated. To accomplish this, we duplicated the cache hierarchy in the simulation model. One set of caches is accessed by all instructions, including mispredicted path instructions, and is used for the timing simulation. The other set is accessed only by correct-path instructions. The delays of the correct-path-only copies are for accounting purposes only and do not affect the flow of instructions through the machine. When the delay of a cache access differs from the correct-path-only cache delay, that difference is caused by mispredicted path instructions. By recording the delay difference (in cycles) and the direction of the difference, we can accurately measure the exact prefetching and/or polluting count as well as the average number of cycles gained or lost due to the difference. Separate tallies are kept for the prefetch and pollution accesses for both the instruction cache hierarchy (fetch) and the data cache hierarchy (load). Given that individual accesses can have very different delay differences (i.e. one access can prefetch from the L2 while another prefetches from main memory), the average cycles gained or lost is a crucial metric.

The simulator can be instructed regarding whether or not the machine model should fetch down the mispredicted path. This allows the IPC difference caused by mispredicted path to be determined. Also, the simulator has the ability to stop mispredicted path loads from accessing the caches. This enables the effects on the data cache to be separated from those on the instruction cache. If the overall net effect of mispredicted path cache accesses is to prefetch data, then this should result in a lower average memory latency and a higher IPC.

Finally, to simulate the effects of the deeper pipelined machines, variable delay stages (VDS) are placed between pipe stages prior to the execute stage in the timing simulation model of the machine (Figure 1). This allows increases in the average branch latency and, proportionally, the number of mispredicted path instructions fetched into the machine. To view these effects, we allow the variable delay stages to be set to 0, 1, 2, or 3. Note that because there are three pipe stages prior to the execute stage, a setting of D=1 adds 3 additional cycles to the front end, a setting of D=2 adds

6 cycles, and so on. These modifications allow us to determine the general trend in IPC as the number of front-end stages increases. [8] did similar studies by noting the effects of injecting more mispredicted path instructions into the instruction trace.

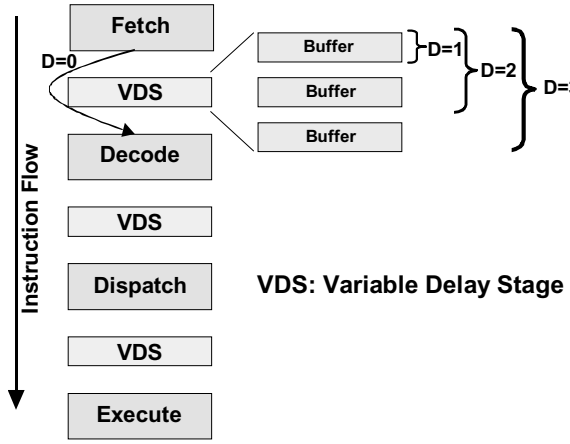

Figure 1. Variable delay stages for simulating deeper front end.

4 Experimental Results

Table 2 shows the cache access discrepancies on the instruction cache hierarchy for running the benchmarks to completion. For this part of the experiment we ignore the data cache accesses by load instructions in the mispredicted path. The amounts of prefetch and pollution accesses are presented along with the average number of cycles gained or lost. The "Net Change" column records the overall cycle change in cache delay cycles caused by the mispredicted path accesses. A positive number indicates a decrease in cycle count (good), whereas a negative number indicates an increase in the cycle count (bad).

Table 2. Cache access discrepancies caused by mispredicted path instructions.

Benchmark	Pollution Accesses	Avg. cycle Loss/access	Prefetch accesses	Avg. cycle gain/access	Net Change (cycles)	Projected % IPC Change
compress	207	1.00	110	24.44	2481	0.02%
gcc	2356036	14.92	543954	34.35	-16479381	-6.67%
go	349958	3.95	188388	50.51	8133593	16.23%
ijpeg	108252	2.88	42341	38.56	1320485	4.12%
li	659	7.96	203	34.36	1730	0.01%
m88ksim	857	6.28	361	34.25	6983	0.02%
perl	96656	29.60	18885	45.63	-1999238	-9.26%
vortex	278049	10.16	99737	34.84	650993	1.16%

Since the machine model used is designed to put pressure on the front-end, a delay caused by a missed fetch access should correspond directly to an increase in the total execution time of the benchmark. If, however, during benchmark execution the machine is not bottlenecked due to the latency of instruction fetching, the cycles gained or lost during this period will not impact the IPC. Assuming that the latter is not the usual case, a projected % IPC change is calculated and included in the table. Taking the execution time of the benchmarks with the mispredicted path disabled and changing it by the "Net Change" value allows us to compute a projected IPC change when the mispredicted paths are enabled. This column is based on the assumption that for each delay cycle difference, this cycle directly changes the total execution cycles of the benchmark. This projection is included to give some rough indication of the potential impact on IPC due to the net changes in total cycle count. Actual IPC changes based on cycle-accurate simulation are presented later.

The results in Table 2 show that most benchmarks have a positive net change due to mispredicted path cache accesses, however the extent varies greatly. Only *perl* and *gcc* show negative net changes. Examining the average cycles gained/lost for the two types of accesses, it can be seen that most gains on prefetch accesses are from main memory prefetches because the average cycles gained per prefetch access is in the range of 25-50 cycles. On the other hand most of the polluting accesses are hitting in the L2. This means the polluting wrong path accesses contaminate mainly the first level of the cache. *Perl* and *gcc* are the exceptions; both exhibit significant number of penalty cycles per pollution access, indicating significant number of misses to the main memory. For these benchmarks, polluting accesses have a tendency to not only remove data from the L1 cache, but from the L2 cache as well.

4.1 Mispredicted Path Impact on IPC

Figure 2 shows the actual percent change in IPC caused by mispredicted path instructions. The data in this figure are obtained by actually simulating the mispredicted path instructions using the fMW cycle-accurate simulator. The values range from the greatest increase in *go* of 12.0%, to the *perl* decrease of -7.93%. The average across all benchmarks is around 1.0%. These values correlate strongly with the projected IPC changes of Table 2 suggesting that, indeed, the machine is bottlenecked mainly in the front-end of the pipeline. It is important to notice the negligible change in IPC for *compress*, *li*, and *m88ksim*. This is because the data and instruction working sets for these benchmarks are small enough to be contained entirely within the caches.

Although the net cycles gained is positive for most benchmarks except for *perl* and *gcc*, the IPC changes vary significantly from benchmark to benchmark. Therefore, the findings of [9] seem to have been on the right track in terms of instruction prefetching, but the conclusion that the mispredicted path instructions always perform instruction prefetching is not substantiated by our results. [8,9] did not actually simulate any benchmark whose accesses caused more pollution than prefetching and thus concluded by saying that the cache effects of mispredicted path instructions would always be beneficial. It is also important to note that whether prefetching or polluting becomes the dominant factor in changing the total cycle count, depends not only on the number of accesses of each type but also on the cycle count impact by each access of each type.

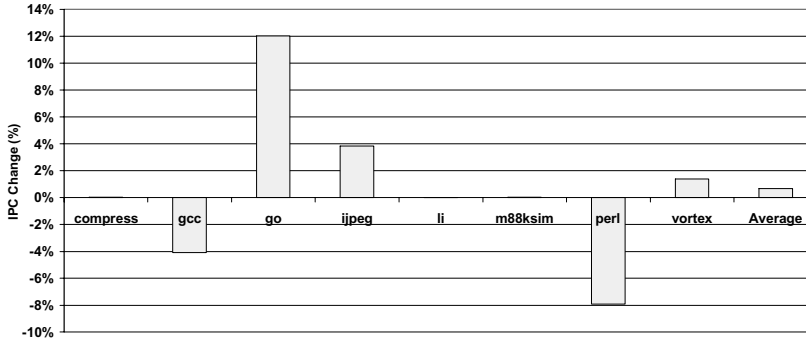

Figure 2. Percent IPC change due to mispredicted path instructions

Mispredicted path load accesses and their impact on the data cache hierarchy (studied in [8]) are also studied here by running the same benchmarks to completion, but allowing the mispredicted path load instructions to access the cache hierarchy if given the opportunity. The results show these loads to have no impact on the IPC of the machine. Of course, for a machine bottlenecked in the front-end, such results are expected. Studying data similar to Table 2 but for the load accesses only, it seems that if the mispredicted path loads could influence the machine performance, the overall trend tends to be polluting rather than prefetching as [8] suggested. Only *go* and *perl* have positive net cycle changes caused by the mispredicted path, and these numbers are negligible. However, it must be noted that such cache delay results are not as pronounced when dealing with the load accesses due to the nature of the out-of-order execution core coupled with an aggressive nonblocking cache hierarchy. Unlike the fetch unit, which must stall when the next predicted instruction misses in the cache, the execution core can be processing other instructions in parallel with outstanding load misses. Only accurate simulation of data dependency relationships for each load can determine the actual impact on IPC.

From our results, it can be seen that the amount and type of cache effects caused by mispredicted path instructions are strongly benchmark dependent. The actual IPC impact is not only a function of the relative frequency of prefetching vs. polluting accesses, but also the number of machine cycles gained or lost by each of such accesses. Applications should be analyzed individually to determine the cache effects of executing mispredicted path instructions. It is interesting to note that *go*, one of the most troublesome benchmarks for branch predictors, shows the best IPC gain due to such serendipitous prefetching. We surmise that this is due to the control flow structure of *go* such as the presence of many short "hammocks" in the code, the traversal of which is difficult to predict. With such control flow structure, a long mispredicted path is likely to converge with the correct path and/or the instructions fetched by the mispredicted path are soon after fetched as part of a correct path. It also appears that for *go,* significant number of such prefetches are prefetching from the main memory, while most of the polluting misses are between L1 and L2.

4.2 Effects Throughout Benchmark Execution

To get a better understanding of how the mispredicted path accesses affect performance, it is necessary to see how the cycle impact changes over the lifetime of the benchmark's execution. To do this, the benchmarks are run for incrementally increasing instruction counts beginning at 1 million and until the end of benchmark execution. We then study the percent IPC changes for runs with and without mispredicted path instruction effects. Temporal IPC changes can be seen in Figure 3. From this figure one can observe that the effects of mispredicted path cache accesses continue throughout the lifetime of the benchmark execution. Conventional wisdom says that these extra cache accesses would be the most beneficial during early part of execution because of compulsory misses. Such behavior is observed for most benchmarks, but its impact is smaller than expected.

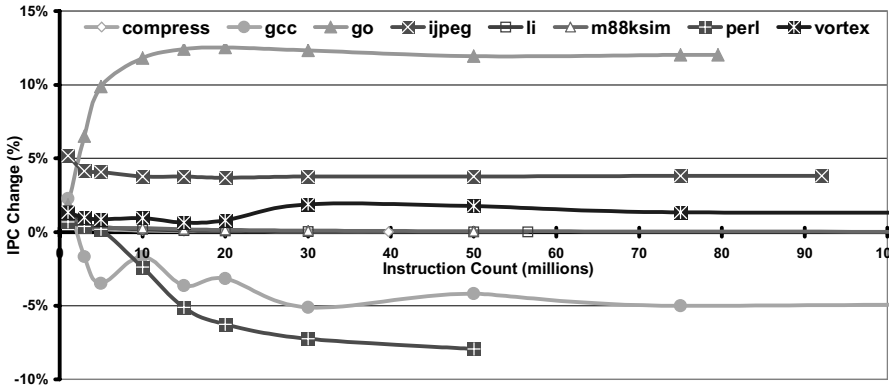

Figure 3. IPC change over benchmark lifetime

4.3 Sensitivity to Memory Latency and Pipeline Depth

Experiments on varying main memory latencies are done on all benchmarks running to an instruction count of 50 million. The results can be seen in Figure 4. As expected, *compress*, *li*, and *m88ksim* display no variation in IPC due to memory latency changes since these benchmarks basically only access main memory for compulsory misses (these three overlap on the 0% axis). The rest of the benchmarks show a linear progression of more dramatic changes in IPC as the latency to main memory is increased, and the slope of each progression is dependent on the specific benchmark. This slope variation is probably caused by the differing prefetch to pollution ratios of the mispredicted path accesses. Similar experiments are also conducted on variations of L2 latency. These results show negligible impact on the percent IPC change for delays varying from 3 cycles to 10 cycles.

Our other sensitivity analysis is done by adding stages in the front-end. This causes mispredicted branches to take longer to resolve, resulting in more fetch cycles being spent fetching instructions down the mispredicted path and increasing the number of mispredicted path instructions being brought into the machine. Figure 5 shows the resulting changes in IPC as the VDS parameter D is varied from 0 to 3. The

experiments on varying front-end pipeline depths are done on all benchmarks and run to an instruction count of 30 million. In general, if the trend is for mispredicted path instructions to cause more prefetching than pollution, then increasing the number of mispredicted path instructions will simply cause more prefetching to occur. [8] showed similar results. If the accesses are more polluting (as it is for *perl* and *gcc*), then more instructions will cause more pollution. Yet this trend is not universal, as can be seen most dramatically in *vortex*. For some benchmarks, it seems that going down the mispredicted path to a certain extent causes prefetching effects. Going too far down the mispredicted path, however, starts polluting the caches more than prefetching.

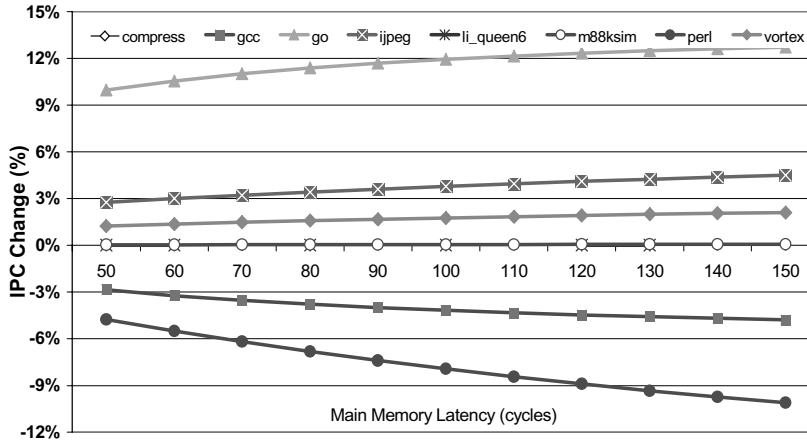

Figure 4. Memory latency effects.

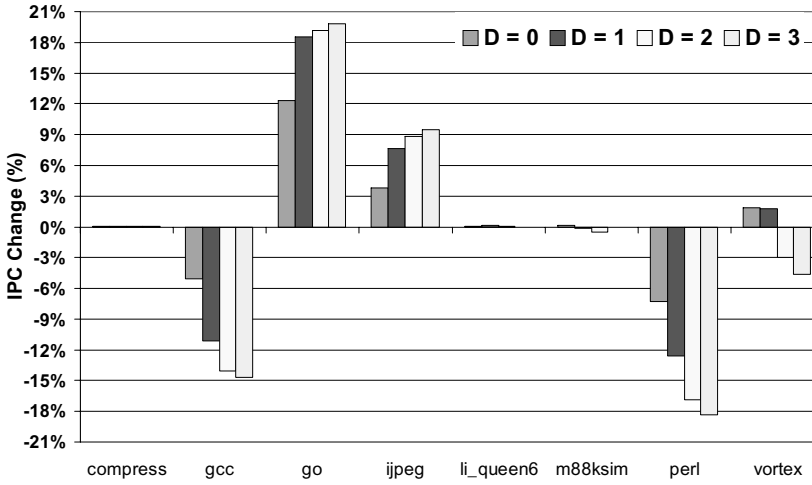

Figure 5. Variable delay stage effects.

5 Conclusion

Mispredicted path instruction fetches can serendipitously prefetch for subsequent correct path instructions. However this potential is highly benchmark dependent; for example *perl* and *gcc* show significant losses in IPC due to pollution. Furthermore, the overall performance impact is a function of the relative numbers of prefetching vs. polluting accesses as well as the actual cycle count impact by each of such accesses. The mispredicted path data accesses seem to display a more uniform trend of pollution over prefetching. Finally we note that the impact of mispredicted path instructions is likely to increase in the future, as shown in our sensitivity experiments. An interesting area of future research is to develop microarchitecture that can adapt the aggressiveness of processing mispredicted path instructions depending on whether the anticipated impact is prefetching or polluting.

Acknowledgement and Footnote

This work benefited from a cluster of simulation machines donated by Intel. We also benefited from discussions with Prof. Trevor Mudge at the University of Michigan. Jonathan Combs is now with Intel's Texas Development Center, and Candice Bechem Combs is now with Motorola in the Engineering Rotation Program.

References

1. C. Bechem, J. Combs, N. Utamaphethai, B. Black, R.D. Blanton, J.P. Shen. "An Integrated Functional Performance Simulator." IEEE Micro, May-June 1999, pp 2-11.
2. B. Black and J.P. Shen. "Calibration of Microprocessor Performance Models." COMPUTER, May 1998, pp. 59-65.
3. A. Cagney. PSIM User's Guide. ftp://cambridge.cygnus.com/pub/psim/index.html August 1996.
4. K. Diefendorf and E. Silha. "The PowerPC User Instruction Set Architecture." IEEE Micro, October 1994, pp. 30-41.
5. S. Jourdan, T. Hsing, J. Stark, and Y. Patt. "The Effects of Mispredicted-Path Execution on Branch Prediction Structures." Conference on Parallel Architectures and Compilation Techniques (PACT), October 1996.
6. D. Lee, J-L. Baer, B. Calder, and D. Grunwald. "Instruction Cache Fetch Policies for Speculative Execution." International Symposium on Computer Architecture, June 1995.
7. IBM Microelectronics Division, PowerPC 604 RISC Microprocessor User's Manual 1994.
8. J. Pierce and T. Mudge. "The Effect of Speculative Execution of Cache Performance." Proceedings of the International Parallel Processing Symposium, April 1994.
9. J. Pierce and T. Mudge. "Wrong Path Instruction Prefetching." Technical Report, University of Michigan, 1994.
10. S. Song, M. Denman, and J. Chang, "The PowerPC 604 RISC Microprocessor." IEEE Micro, October 1994, pp. 8-17.