

Understanding The Effects of Wrong-Path Memory References on Processor Performance

Onur Mutlu Hyesoon Kim David N. Armstrong Yale N. Patt

Department of Electrical and Computer Engineering
The University of Texas at Austin
{onur,hyesoon,dna,patt}@ece.utexas.edu

ABSTRACT

High-performance out-of-order processors spend a significant portion of their execution time on the incorrect program path even though they employ aggressive branch prediction algorithms. Although memory references generated on the wrong path do not change the architectural state of the processor, they can affect the arrangement of data in the memory hierarchy. This paper examines the effects of wrong-path memory references on processor performance. It is shown that these references significantly affect the IPC (Instructions Per Cycle) performance of a processor. Not modeling them can lead to errors of up to 10% in IPC estimates for the SPEC2000 integer benchmarks; 7 out of 12 benchmarks experience an error of greater than 2% in IPC estimates. In general, the error in the IPC increases with increasing memory latency and instruction window size.

We find that wrong-path references are usually beneficial for performance, because they prefetch data that will be used by later correct-path references. L2 cache pollution is found to be the most significant negative effect of wrong-path references. Code examples are shown to provide insights into how wrong-path references affect performance.

1. INTRODUCTION

High-performance processors employ aggressive branch prediction techniques in order to exploit high levels of instruction-level parallelism. Unfortunately, even with low branch misprediction rates, these processors spend a significant number of cycles fetching instructions from the mispredicted (i.e. wrong) program path. The leftmost bar in Figure 1 shows the percentage of total cycles spent fetching wrong-path instructions in the SPEC2000 integer benchmarks. The middle and rightmost bars of Figure 1 show the percentage of instructions fetched and executed on the wrong path¹. On average, even with a 4.2% conditional branch misprediction rate, the evaluated processor spends 47% of its total cycles fetching wrong-path instructions. 53% of all fetched instructions and 17% of all executed instructions are on the wrong path. 6% of all executed instructions are wrong-path data memory access instructions (loads and stores).

Although wrong-path data and instruction memory references do not change the architectural state of the machine, they can affect the arrangement of data in the memory hierarchy. In this paper, we examine the effect of wrong-path memory references on the performance of a processor. In

¹Machine configuration and simulation methodology are described in Section 2.

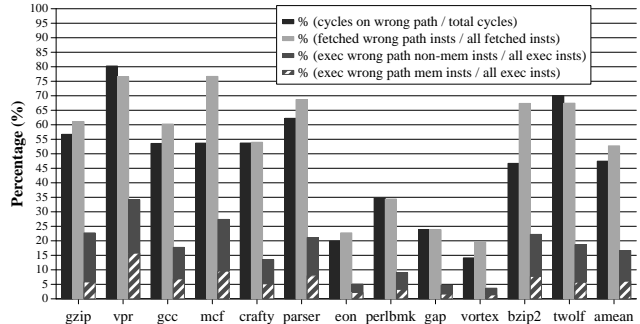


Figure 1: Percentage of total cycles spent on the wrong path, percentage of instructions fetched on the wrong path, and percentage of instructions (memory and non-memory) executed on the wrong path in the baseline processor for SPEC 2000 integer benchmarks.

particular, we seek answers to the following questions:

1. How important is it to correctly model wrong-path memory references? What is the error in the predicted performance if wrong-path references are not modeled?
2. Do wrong-path memory references affect performance positively or negatively? What is the relative significance on performance of prefetching, bandwidth consumption, and pollution caused by wrong-path references?
3. What kind of code structures lead to the positive effects of wrong-path memory references?

Our results indicate that wrong-path memory references significantly affect processor performance and not modeling them may lead to errors of up to 10% in IPC estimates. Although they have a positive effect on performance for most of the benchmarks due to prefetching, wrong path references negatively impact performance for a few others. We analyze the causes for the positive and negative performance impact. We identify pollution in the L2 cache as the dominant negative effect of wrong-path references and present code examples to illustrate the prefetching effects.

2. EXPERIMENTAL METHODOLOGY

We use an execution-driven simulator capable of accurately fetching and executing instructions on the wrong path

and correctly recovering from mispredictions that occur on the wrong path. The baseline processor we model is an 8-wide out-of-order processor with an instruction window that can hold 128 instructions. The conditional branch predictor is a hybrid branch predictor composed of a 64K-entry gshare [5] and a 64K-entry PAs [9] predictor with a 64K-entry selector along with a 4K-entry branch target buffer. The indirect branch predictor is a 64K-entry, 4-way target cache [4]. We model a deep pipeline with a 20-cycle branch misprediction latency. The data and instruction caches are 64KB, 4-way with 8 banks and a 2-cycle hit latency. The unified L2 cache is 1MB, 8-way with 8 banks and a 10-cycle hit latency. All caches have a line size of 64 bytes. We model bandwidth, port contention, bank conflicts, and queuing effects at all levels in the memory hierarchy.

The memory system we model is shown in Figure 2. At most 128 I-Cache and D-Cache requests may be outstanding. These requests may reside in any of the four queues in the memory system. Two of these queues, L2 Request Queue and Bus Request Queue are priority queues where requests generated by older instructions have higher priority. Such prioritization is fairly easy to implement on-chip and reduces the probability of a full window stall by servicing older instructions’ requests earlier. Memory Queue and L2 Fill Queue are modeled as FIFO queues. The bus is pipelined, split-transaction, 256-bit wide, and has a one-way latency of 100 processor cycles. At most two requests can be scheduled onto the bus every bus cycle, one from the Bus Request Queue and one from the Memory Queue. Processor frequency is four times the bus frequency. We model 32 DRAM banks, each with an access latency of 300 processor cycles. Hence, the round-trip latency of an L2 miss request is a minimum of 500 processor cycles (300-cycle memory access + 200-cycle round-trip on the bus) without any queuing delays and bank conflicts. On an L2 cache miss, the requested cache line is brought into both the L2 cache and the first-level cache that initiated the request. A store instruction request that misses the data cache or the L2 cache allocates a line in the respective cache. Write-back requests from D-Cache are inserted into the L2 Request Queue and write-back requests from the L2 Cache are inserted into the Bus Request Queue as bandwidth becomes available from instruction and data fetch requests.

The experiments were run using the 12 SPEC2000 integer benchmarks compiled for the Alpha ISA with the `-fast` optimizations and profiling feedback enabled. The benchmarks were run to completion with a modified test input set to reduce simulation time. The number of retired instructions along with branch misprediction and cache miss rates per 1000 instructions for each benchmark are shown in Table 1.

3. WRONG PATH: TO MODEL OR NOT TO MODEL

In this section, we measure the error in IPC if wrong-path memory references are not simulated. We also evaluate the overall effect of wrong-path memory references on the IPC (retired Instructions Per Cycle) performance of a processor. We investigate how the effects of wrong-path references change with memory latency and instruction window size. In order to isolate the effects of wrong-path memory references, we ensure that wrong-path execution can only affect

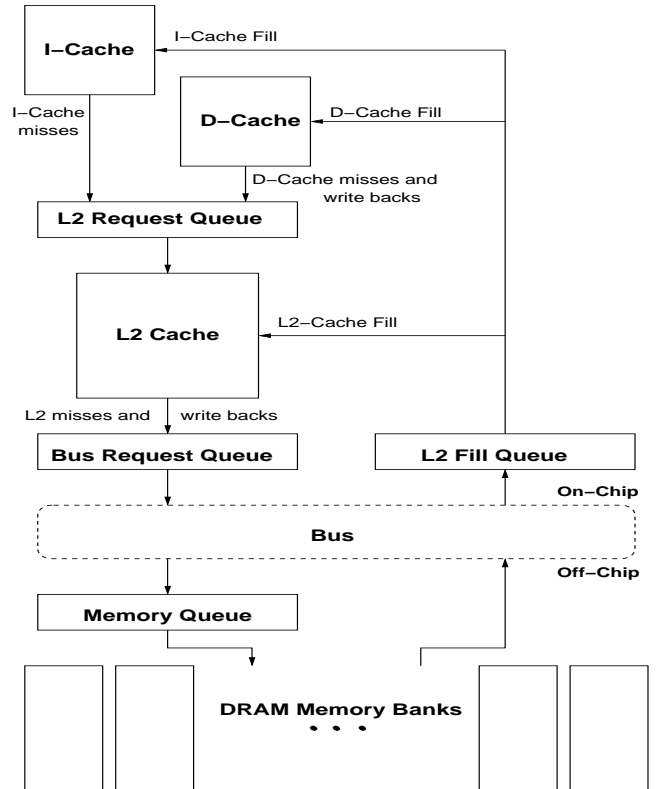


Figure 2: Memory system modeled for evaluation.

the execution on the correct path through changes in the memory system. All other state that is updated speculatively during wrong-path execution is restored upon recovery from misprediction.

Figure 3 shows, for reference, the IPC performance of the baseline processor for three different minimum memory latencies (250, 500, and 1000 cycles) when wrong-path memory references are correctly modeled. Figure 4 shows the percent error in IPC for the same three models when wrong-path memory references are not modeled at all². A positive error means that the IPC obtained when wrong-path references are not modeled is higher than the IPC obtained when they are modeled (i.e. a positive error implies wrong-path references are detrimental to performance)³.

Figure 4 shows that error in IPC estimates can be quite significant for some benchmarks if wrong-path memory references are not modeled. For instance, the IPC obtained for `mcf` without wrong-path references is 8% lower than the IPC obtained with wrong-path references, for a 250-cycle memory latency. Error in the average IPC⁴ can be as much as 3.5% for a 1000-cycle memory latency. Error in IPC generally increases as memory latency increases, which suggests that modeling wrong-path references will be even more im-

²Not modeling the wrong-path memory references is accomplished by stalling the fetch stage until a mispredicted branch is resolved and machine state is recovered.

³In effect, Figure 4 shows the difference in IPC when trace-driven simulation is used instead of the baseline execution-driven simulation.

⁴Rightmost set of bars in Figure 4 shows the error in the average IPC, not the average of the error.

Benchmark	Inst. count	BP miss rate	L2 miss rate	DC miss rate	IC miss rate
gzip	366 M	5.89	0.28	5.20	0.00
vpr	567 M	11.65	0.42	10.90	0.00
gcc	218 M	9.84	0.46	1.95	2.46
mcf	173 M	13.31	28.86	53.62	0.00
crafty	498 M	5.18	0.12	1.65	0.92
parser	412 M	8.89	0.87	5.48	0.08
eon	129 M	1.15	0.05	0.03	0.09
perlbmk	99 M	3.27	0.11	3.06	4.35
gap	404 M	1.46	4.64	4.76	0.03
vortex	165 M	1.42	3.47	5.60	2.01
bzip2	418 M	8.05	1.53	5.80	0.00
twolf	279 M	8.87	0.02	0.09	0.04

Table 1: The number of retired instructions, branch misprediction rate, and L2, D-Cache (DC), I-Cache (IC) miss rates per 1000 retired instructions on the baseline processor for the simulated benchmarks.

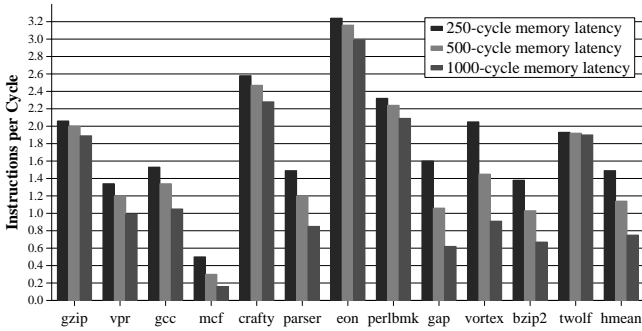


Figure 3: IPC of the baseline processor for three different memory latencies when wrong-path accesses are correctly modeled.

portant in future processors with longer latencies to memory. This is because, with increased memory latencies, the positive and negative effects of wrong-path memory operations become more pronounced in terms of their contribution to execution cycles. For instance, a wrong-path reference that generates a memory request that is later used by a correct-path reference, and thus saves 1000 cycles, affects the IPC more than one which saves only 250 cycles. Mcf, where error decreases as memory latency increases, is an exception. In this benchmark, long-latency cache misses caused by wrong-path references delay the servicing of correct-path misses by consuming bandwidth and resources. This bandwidth contention becomes more significant at longer memory latencies, therefore performance improvement due to wrong-path references reduces with increased memory latency.

Figure 4 also shows that wrong-path references have a positive effect on overall processor performance for many of the benchmarks, especially for mcf, parser, and perlbmk. The only benchmarks where wrong-path references have a significant negative affect on IPC are vpr and gcc.

Figure 5 shows that the percentage (and therefore, the number⁵) of executed wrong-path instructions does not significantly increase with increased memory latency. This is due to the limited instruction window size of 128. When the processor remains on the wrong path for hundreds of cycles

⁵Because the number of executed correct-path instructions is always constant for a benchmark.

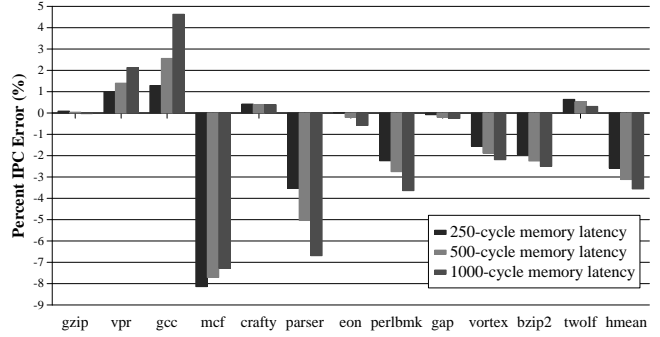


Figure 4: Error in the IPC of the baseline processor for three different memory latencies if wrong-path memory references are not simulated.

due to a mispredicted branch dependent on an L2 cache miss, the processor incurs a full window stall due to its limited window size. Hence, increasing the memory latency does not increase the number of executed wrong-path instructions. However, increasing the memory latency does increase the contribution wrong-path memory references make to the number of execution cycles, as explained above. To determine the effect of increased number of wrong-path instructions on performance estimates, we next evaluate processors with larger instruction windows that allow the execution of more instructions on the wrong path.

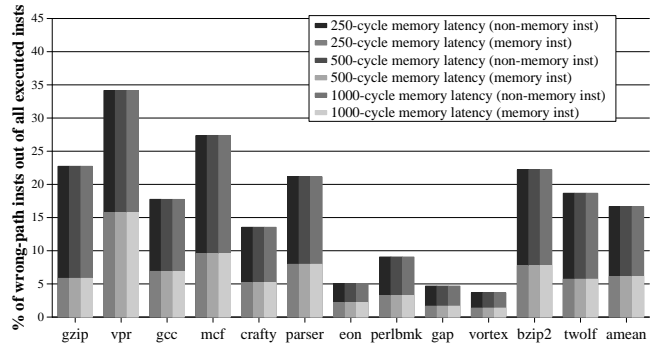


Figure 5: Percentage of executed wrong-path instructions out of all executed instructions for three different memory latencies.

3.1 Effect of Wrong-Path Memory References in Larger Instruction Windows

Future processors will have larger instruction windows to exploit even higher levels of instruction-level parallelism. A larger instruction window would change the effect of wrong-path memory references on performance in two major ways:

1. A larger window allows more wrong-path references to be executed by decreasing the number of full window stalls encountered on the wrong path. If references that occur later on the wrong path have positive effects, such as prefetching, a larger window could increase the positive impact of wrong-path references on IPC. On the other hand, if later wrong-path references

have negative effects, such as pollution, IPC could be negatively affected.

2. With a larger window, the processor is better able to tolerate the negative effects caused by wrong-path memory references.

Figure 6 shows the error in IPC estimates for processors with three different instruction window sizes, when wrong-path memory references are not modeled⁶. Error in IPC is almost 10% in mcf for a window size of 512. Aside from a couple of exceptions, notably perlbnk and gcc, error in IPC generally increases with increasing window size if wrong-path memory references are not modeled. With a larger instruction window the processor is able to execute more memory operations on the wrong path as shown in Figure 7, which changes the impact of wrong-path memory references on IPC.

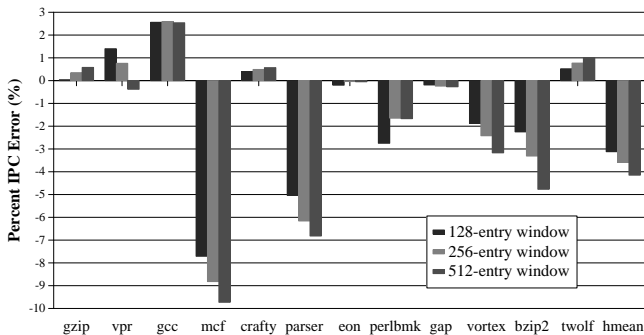


Figure 6: Error in the IPC of the baseline processor for three different instruction window sizes if wrong-path memory references are not simulated.

How the increase in the number of executed wrong-path references affects IPC depends on the usefulness of the extra references executed. In perlbnk, memory references executed further down on the wrong path start canceling out the positive prefetching effects of the operations executed earlier. Therefore, with a larger instruction window, wrong-path memory references have a less positive effect on IPC in perlbnk. On the other hand, we see the opposite effect in vpr, mcf, parser, vortex, and bzip2. Wrong-path references executed further down on the wrong path are useful for correct-path operations encountered after the processor resolves the mispredicted branch for these five benchmarks.

4. ANALYSIS

Wrong-path memory references affect processor performance significantly. Therefore, it is important to understand why that is the case. In this section, we analyze the reasons behind the positive or negative impact that wrong-path references have on performance.

4.1 Bandwidth and Resource Contention

Wrong-path references can use bandwidth and resources and thus get in the way of correct-path references by delaying the servicing of correct-path memory requests. To examine how much the bandwidth and resource contention caused by wrong-path references affects IPC, we simulated

⁶Memory latency is fixed at 500 cycles for these simulations.

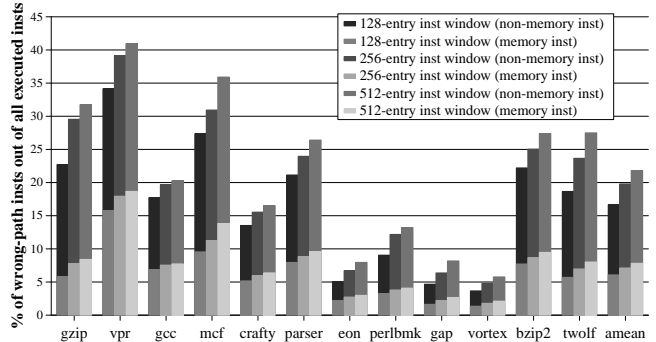


Figure 7: Percentage of executed wrong-path instructions out of all executed instructions three different instruction window sizes.

an idealized unrealistic processor which always gives lower priority to wrong-path references everywhere in the memory system. In this model, wrong-path references never get in the way if there are correct-path references outstanding. If a resource, such as a queue entry, is tied up by a wrong-path reference and a correct-path reference needs that resource, the model allocates the resource to the correct-path reference. We compared the performance of this idealized model to the baseline processor. We found that the performance difference between the two models is negligible for all benchmarks except mcf, whose IPC improves by 2.6% with the idealized model. Mcf, a benchmark with a very high L2 miss rate, generates many wrong-path references that miss in the L2 cache. These references keep the memory banks busy and delay the correct-path references that later try to access the same banks. In other benchmarks wrong-path references do not cause significant bandwidth and resource contention for correct-path references.

4.2 Usefulness of Wrong-path References

Wrong-path references can increase performance by prefetching data or reduce performance by polluting the caches. We explain the impact of these effects on performance by examining the accuracy of wrong-path data and instruction references. We categorize the misses caused by wrong-path references in three groups:

1. Unused wrong-path miss: caused by a wrong-path reference, but the allocated cache line is never used by a correct-path reference or it is evicted before being used.
2. Fully-used wrong-path miss: caused by a wrong-path reference and the allocated cache line is later used by a correct-path reference.
3. Partially-used wrong-path miss: initiated by a wrong-path reference and later required by a correct-path reference while the request is in flight.

Figure 8 shows the number of data cache misses for two processor models. The leftmost stacked bar for each benchmark shows the number of data cache misses for the baseline model that executes wrong-path memory references. The rightmost bar shows the number of data cache misses for a

simulator that does not model wrong-path references. We show the raw number of misses in this figure to illustrate the impact data cache misses can have on performance. We observe that the number of correct-path data cache misses are reduced by 13% on average when wrong-path references are modeled correctly, which hints at why most benchmarks benefit from wrong-path references. This reduction is most significant in *vpr* (30%) and *mcf* (26%). In *mcf*, this reduction affects the IPC positively (as was shown in Figure 4) because most (90%) of the wrong-path data cache misses are fully or partially used. Wrong-path data cache misses that also miss in the L2 cache provide very accurate long-latency prefetches in *mcf*, and this positively impacts the IPC. However, in *vpr*, many unused wrong-path data cache misses cause significant pollution in the L2 cache, as we show in section 4.3. Therefore, *vpr*'s performance is adversely affected by wrong-path data references. On average, 76% of the wrong-path data cache misses are fully or partially used.

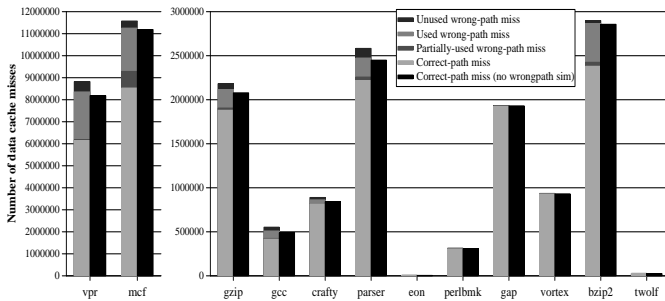


Figure 8: Number of data cache misses for the baseline (leftmost bar for each benchmark) and a model that does not execute wrong-path references (rightmost bar for each benchmark). Note that the y-axis for *vpr* and *mcf* is on a different scale due to the large number of misses they experience.

Figure 9 shows the number of instruction cache misses for the same two processor models. We observe that only *gcc*, *crafty*, *perlbmk*, and *vortex* can be affected by wrong-path instruction references, because only these benchmarks incur a significant number of instruction cache misses. The accuracy of wrong-path instruction requests is lower than that of wrong-path data requests. On average, 69% of the wrong-path instruction cache misses are fully or partially used. Only 60% of wrong-path instruction cache misses are fully or partially used in *gcc* and the pollution caused by the other 40%, which are unused, is the reason why *gcc* loses performance due to wrong-path references. On the other hand, used wrong-path instruction cache misses in *perlbmk* and *vortex* provide significant performance increase.

We find that, in *gcc*, many unused instruction cache misses also miss in the L2 cache and evict useful L2 cache lines. Since L2 cache miss latency is very high, these unused wrong-path misses decrease performance significantly. In contrast, in *crafty*, which also has a large number of unused wrong-path instruction cache misses, most of these misses are satisfied in the L2 cache. Therefore, these misses do not evict useful lines from the L2, they only cause pollution in the instruction cache. That’s why the IPC of *crafty* is not significantly reduced due to unused wrong-path references, as was shown in Figure 4. Unused wrong-path instruction cache misses do not cause significant pollution in *perlbmk*, as we

show in section 4.3. The prefetching benefit of used wrong-path instruction cache misses outweighs the pollution caused by unused ones in *vortex*. Hence the performance increase in these two benchmarks.

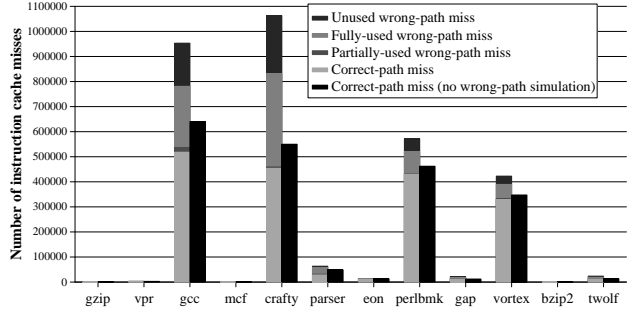


Figure 9: Number of instruction cache misses for the baseline (leftmost bars) and a model that does not execute wrong-path references (rightmost bars).

4.3 Understanding the Pollution Effects

In order to understand the performance impact of cache pollution caused by wrong-path references, we eliminate wrong-path pollution from the three caches. We hypothesize that pollution caused by wrong-path references in the first-level instruction and data caches would be less detrimental to performance than pollution in the L2 cache, due to the very high miss penalty of the L2 cache. We evaluate four different idealized models to test this hypothesis: three models in which wrong-path requests do not cause pollution in I-cache, D-cache, and L2 cache, respectively; and a model in which wrong-path requests do not cause pollution in any of the caches⁷. We model “no pollution” by storing lines fetched by wrong-path references in a separate buffer rather than the respective cache and moving those lines to the respective cache only when they are used by a correct-path request. These models are idealized because a real processor does not know whether or not a reference is a wrong-path reference until the mispredicted branch is resolved.

Figure 10 shows the IPC improvement over baseline of these four idealized models. Eliminating the pollution caused by wrong-path references from the first-level instruction and data caches does not affect performance except in *crafty* and *vortex*. In contrast, eliminating the pollution in the L2 cache increases performance for half of the benchmarks, including *gcc* and *vpr* where wrong-path references are detrimental for overall performance. In *gcc*, eliminating L2 cache pollution increases the baseline performance by 10% and thus makes wrong-path references beneficial for overall performance. In *mcf* and *parser*, eliminating L2 cache pollution increases IPC by 6% and 4.5% respectively, further increasing the usefulness of wrong-path references in these benchmarks.

We investigate whether pollution in the first-level caches has a more pronounced effect on IPC when using smaller first-level caches. Figure 11 shows the IPC improvement of the four idealized models when 16KB instruction and data

⁷We also examined a model where wrong-path requests do not cause pollution in both the I-cache and the D-cache, but cause pollution in the L2 cache. The results obtained using this model were negligibly different from the results obtained using the model which eliminates only I-cache pollution.

caches are used. We can see that pollution in especially the instruction cache becomes more significant for performance with smaller instruction and data caches. Data cache pollution is still not significant, because the relatively short-latency misses it causes are tolerated by the 128-entry instruction window. Instruction cache pollution due to wrong-path prefetches affects performance significantly in gcc, crafty, perlbnk, vortex, and twolf, four of which have significant numbers of unused wrong-path instruction cache misses (shown in Figure 9). However, even with smaller first-level caches, removing pollution in the L2 cache is more important than removing pollution in either of the first-level caches.

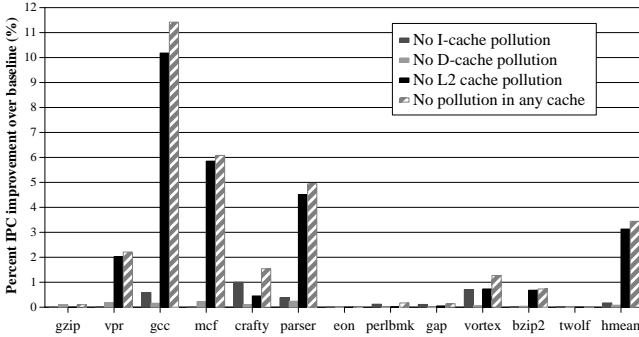


Figure 10: IPC improvement over the baseline processor if pollution caused by wrong-path references is eliminated from caches.

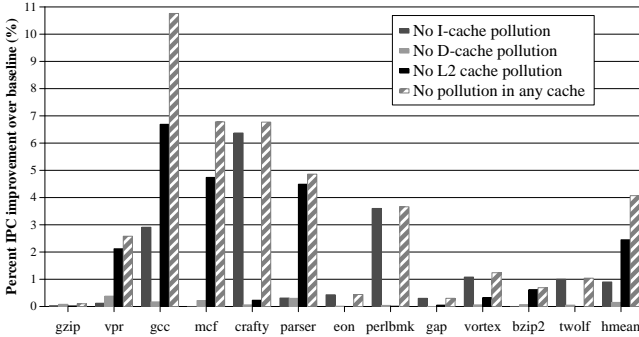


Figure 11: IPC improvement over the baseline processor with 16KB instruction and data caches if pollution caused by wrong-path references is eliminated from caches.

Figure 12, which shows the normalized number of L2 cache misses in the baseline model and a model that does not simulate wrong-path references, provides insight into why L2 cache pollution degrades performance in vpr and gcc when wrong-path references are modeled. For these two benchmarks, the number of L2 cache misses suffered by correct-path instructions (correct-path miss + partially-used wrong-path miss in Figure 12) increases significantly when wrong-path references are modeled, due to the pollution caused by unused wrong-path L2 cache misses. On the other hand, the number of L2 cache misses suffered by correct-path instructions either decreases or stays the same for other benchmarks when wrong-path references are modeled, which explains why wrong-path references are beneficial for the performance of most benchmarks.

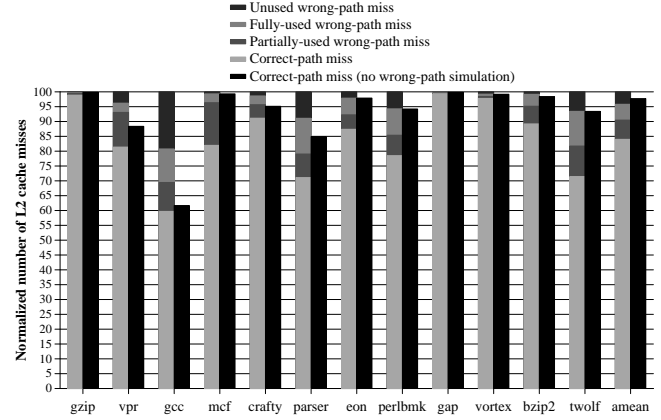


Figure 12: Normalized number of L2 cache misses for the baseline (leftmost bars) and a model that does not execute wrong-path references (rightmost bars).

We conclude that pollution in the L2 cache is the most significant negative effect of wrong-path memory references. In order to reduce the negative impact of wrong-path references or to increase their positive effects, high-performance processors should adopt policies to reduce the L2 cache pollution caused by wrong-path references.

4.4 Understanding the Prefetching Effects

Previous sections have shown that, in general, the prefetching benefits of wrong-path references outweigh their negative effects, such as bandwidth demands or cache pollution. In this section we present code examples to provide insights into why wrong-path memory references can be beneficial for correct-path execution.

4.4.1 Prefetching Data for Later Loop Iterations

We find that wrong-path execution of a loop iteration can prefetch data for the correct-path execution of the same iteration. This can happen when a conditional branch inside the loop is mispredicted and the processor continues to execute the next iteration(s) on the wrong path. Our analysis shows that most of the useful wrong-path data cache misses in mcf and bzip2 are generated in this fashion.

Figure 13 shows a code section from mcf’s primal_bea_mpp function, which performs an optimization routine. The shown for loop traverses an array of pointers to arc_t structures and performs operations on a single arc_t structure in each iteration. The branch in line 4 is dependent on the pointer load arc->ident, and is mispredicted 30% of the time. In some iterations the processor mispredicts this branch and does not execute the body of the if statement and starts executing the next iteration on the wrong path. This wrong-path execution of the next iteration initiates a load request for the next arc->ident. When the mispredicted branch is resolved, the processor recovers, executes the body of the if statement and starts the next iteration on the correct path. Back on the correct path, the processor generates a load request for arc->ident, which has already been prefetched into the data cache by the previous execution of this iteration on the wrong path. We find that the load of arc->ident frequently misses the data cache and sometimes the L2 cache. Therefore, the wrong-path execution of later

iterations of this `for` loop prefetches data that will later be used by the correct-path execution of the same iterations. The instruction that loads `arc->ident` causes 63% of the wrong-path data cache misses in `mcf` and 99% of these are fully or partially used by later correct-path references. In this example, because the body of the `if` statement contains a lot of instructions, not executing the body of the `if` statement in the iterations executed on the wrong path parallelizes the misses caused by the load of `arc->ident` in different iterations. The parallelization of these misses may not be achieved if the processor remains on the correct path and correct path requires the execution of the body of the `if` statement in each iteration, because the instruction window would be filled with instructions in the body of the `if` statement instead of instructions that load `arc->ident`.

```

1 : arc_t *arc; // array of arc_t structures
2 : // initialize arc (arc = ...)
3 :
4 : for ( ; arc < stop_arcs; arc += size) {
5 :     if (arc->ident > 0) { // frequently mispredicted branch
6 :         // function calls and
7 :         // operations on the structure pointed to by arc
8 :         // ...
9 :     }
10: }
```

Figure 13: An example from `mcf` showing wrong-path prefetching for later loop iterations.

4.4.2 One Loop Prefetching for Another

Although less common and less accurate than wrong-path prefetching within iterations of the same loop, two different loops can prefetch data for each other if they are both working on the same data structure.

Figure 14 shows a code example from `mcf`, a sorting routine which exhibits wrong-path prefetching behavior. The two `while` loops in lines 5 and 7 traverse an array of pointers to structures, `perm`, and compare a member of each structure to the value `cut`. It is important to note that the first traversal begins from a lower memory address and works up, while the second traversal begins at a higher memory address and works down. Both `while` loops branch based on a data-dependent condition. We find that when the first `while` loop mispredicts its branch-terminating condition and continues executing loop iterations, its accesses to `perm[1]->abs_cost` continue to load data from the upper part of the `perm` array and, in the process, serve to prefetch data elements for the second `while` loop.

4.4.3 Prefetching in Control-Flow Hammocks

If a hammock branch is mispredicted, the loads executed on the mispredicted path in the hammock may provide useful data for the loads that are later executed on the correct path in the hammock. This happens if both paths of the hammock need the same data.

The `while` loop from `mcf` benchmark’s `refresh_potential` function, shown in Figure 15, demonstrates this kind of wrong-path prefetching. This function traverses a linked data structure. Depending on the `orientation` of the `node` visited, a `potential` is calculated for the `node`. Note that the values used to calculate the potential are the same regardless of the `orientation` of the node. In other words, instructions in the `if` block and instructions in the `else` block

```

1 : l = min; r = max;
2 : cut = perm[ (long)( (l+r) / 2 ) ]->abs_cost;
3 :
4 : do {
5 :     while( perm[l]->abs_cost > cut )
6 :         l++;
7 :     while( cut > perm[r]->abs_cost )
8 :         r--;
9 :
10:    if( l < r ) {
11:        xchange = perm[l];
12:        perm[l] = perm[r];
13:        perm[r] = xchange;
14:    }
15:    if( l <= r ) {
16:        l++; r--;
17:    }
18: } while( l <= r );
```

Figure 14: An example from `mcf` showing prefetching between different loops.

use the same data. Therefore, if the branch of the `if` statement is mispredicted, wrong-path load instructions will generate requests for `node->basic_arc->cost` and `node->pred->potential`. Once the mispredicted branch is resolved, correct-path load instructions will generate requests for the same data, which would already be in the cache or in flight. Our analysis shows that wrong-path cache misses caused by the `if` block and the `else` block of this hammock constitute 6% of the wrong-path data cache misses in `mcf` and more than 99% of them are fully or partially used by instructions on the correct path.

```

1 : node_t *node;
2 : // initialize node
3 : // ...
4 :
5 : while (node) {
6 :
7 :     if (node->orientation == UP) { // mispredicted branch
8 :         node->potential = node->basic_arc->cost
9 :             + node->pred->potential;
10:    } else { /* == DOWN */
11:        node->potential = node->pred->potential
12:            - node->basic_arc->cost;
13:        // ...
14:    }
15:
16:    node = node->child;
17: }
```

Figure 15: An example from `mcf` showing prefetching in control-flow hammocks.

5. RELATED WORK

Butler compares the performance of trace-driven versus execution-driven simulation for the SPEC89 integer benchmarks on a machine with a 10-cycle memory latency [3]. Butler shows that in general, execution-driven simulation performs worse than trace-driven simulation and the main reason for this is pollution of the branch prediction structures. However, a performance improvement is observed on one benchmark, `gcc`, when the instruction and data caches are speculatively updated, but the branch prediction structures are not speculatively updated. We confirm the result that speculative memory references can improve performance and find that the beneficial effects are even more

pronounced with longer memory latencies.

Moudgill et al. investigate the effect wrong-path memory accesses have on IPC and data cache miss rates [6]. Their objective is to determine whether, in light of speculative execution, trace-driven simulators can accurately inform the design decisions made during processor development. They compare the IPC of a processor running the SPEC95 integer benchmarks with and without wrong path memory accesses; their memory latency is 40 cycles. They find that the IPC difference is negligible in all but one case (an unexplained outlier with a difference of 4% in IPC occurs for the benchmark compress). We show that processor performance is less sensitive to wrong-path memory accesses when using low memory latencies, hence the negligible differences in IPC reported by Moudgill et al.

Pierce and Mudge study the effect of wrong-path memory references on cache hit rates [7]. They develop a tool used to simulate wrong-path memory accesses and use this tool to show that wrong-path memory accesses allocate useful data and instruction cache blocks 50% of the time on the SPEC92 C benchmarks.

Pierce and Mudge introduce an instruction cache prefetching mechanism, which leverages the usefulness of wrong-path memory references to the instruction cache [8]. Their mechanism fetches both the fall-through and target addresses of conditional branch instructions, i.e., they prefetch both the correct-path and wrong-path instructions. They find that wrong-path prefetching improves performance by up to 4% over a next-line prefetching mechanism. Pierce and Mudge observe that the effectiveness of wrong-path prefetching increases as the memory latency is increased.

Bahar and Albera investigate a method of capturing the beneficial aspects of speculative memory references while avoiding the pollution effects of wrong-path memory accesses [1]. They assume *a priori* that wrong-path references degrade performance. They use a branch confidence predictor to indicate when the processor is likely on the wrong path in which case, the results of all memory accesses are placed into a separate fully-associative 16-entry buffer. A maximum performance improvement of 3.4% is observed when the results of all wrong-path references are stored in the separate buffer. However this performance improvement is due primarily to the additional associativity provided by the separate buffer. We refute Bahar and Albera's assumption that wrong-path references always degrade performance and show that wrong-path references do benefit performance in many cases.

Bhargava et al. observe that the major disadvantage of trace-driven simulation is its inability to model the fetch and execution of speculative instructions [2]. They propose a method for augmenting a trace-driven simulator to model the effects of speculative execution, including most memory references from the wrong path.

6. CONCLUSION

In this paper, we evaluate the effects wrong-path references have on processor performance. Our evaluation reveals that:

1. Modeling wrong-path memory references is important, since not modeling them leads to errors of up to 10% in IPC estimates.
2. Modeling wrong-path memory references will be more

important in future processors with longer memory latencies and larger instruction windows.

3. In general, wrong-path memory references are beneficial for processor performance because they prefetch data into processor caches.
4. The dominant negative effect of wrong-path memory references is the pollution they cause in the L2 cache. Pollution in the first-level caches or bandwidth and resource usage of wrong-path references do not significantly impact performance.
5. The prefetching benefit of wrong-path references can be caused by different code structures. For the benchmarks examined, the main benefit comes from wrong-path prefetching of the data used by a loop iteration before that iteration is executed on the correct-path.

In light of these results, to increase the performance of processors, designers should focus on eliminating the L2 cache pollution caused by wrong-path memory references. Perhaps compilers should also structure the code such that wrong-path execution always provides prefetching benefits for later correct-path execution, especially for references that have a high probability of cache miss and around branches that are frequently mispredicted.

7. REFERENCES

- [1] R. I. Bahar and G. Albera. Performance analysis of wrong-path data cache accesses. In *Workshop on Performance Analysis and its Impact on Design, 25th Annual Intl. Symposium on Computer Architecture*, 1998.
- [2] R. Bhargava, L. K. John, and F. Matus. Accurately modeling speculative instruction fetching in trace-driven simulation. In *Proceedings of the IEEE Performance, Computers and Communications Conference*, pages 65–71, 1999.
- [3] M. G. Butler. *Aggressive Execution Engines for Surpassing Single Basic Block Execution*. PhD thesis, University of Michigan, 1993.
- [4] P.-Y. Chang, E. Hao, and Y. N. Patt. Predicting indirect jumps using a target cache. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 274–283, 1997.
- [5] S. McFarling. Combining branch predictors. Technical Report TN-36, Digital Western Research Laboratory, June 1993.
- [6] M. Moudgill, J.-D. Wellman, and J. H. Moreno. An approach for quantifying the impact of not simulating mispredicted paths. In *Workshop on Performance Analysis and Its Impact in Design Workshop, 25th Annual Intl. Symposium on Computer Architecture*, June 1998.
- [7] J. Pierce and T. Mudge. The effect of speculative execution on cache performance. In *Proceedings of the Intl. Parallel Processing Symposium*, pages 172–179, 1994.
- [8] J. Pierce and T. Mudge. Wrong-path instruction prefetching. In *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 165–175, 1996.
- [9] 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 124–134, 1992.