Wrong Path Events: Exploiting Unusual and Illegal Program Behavior for Early Misprediction Detection and Recovery

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

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

Abstract

Control and data speculation are widely used to improve processor performance. Correct speculation can reduce execution time, but incorrect speculation can lead to increased execution time and greater energy consumption.

This paper proposes a mechanism to leverage unexpected program behavior, called wrong-path events, that occur during periods of incorrect speculation. A wrong-path event is an instance of illegal or unusual program behavior that is more likely to occur on the wrong path than on the correct path, such as a NULL pointer dereference. When a wrong-path event occurs, the processor can predict that it is on the wrong path and speculatively initiate misprediction recovery. The purpose of the proposed mechanism is to improve the effectiveness of speculative execution in a processor by helping to insure that the processor remain "on the correct path" throughout periods of speculative execution.

We describe a set of wrong-path events which can be used as strong indicators of misprediction. We find that on average 5% of the mispredicted branches in the SPEC2000 integer benchmarks produce a wrong-path event an average of 51 cycles before the branch is executed. We show that once a wrong-path event occurs, it is possible to accurately predict which unresolved branch in the processor is mispredicted using a simple, novel prediction mechanism. We discuss the advantages and shortcomings of wrong-path events and propose new areas for future research.

1. Introduction

Accurate branch prediction is an important factor in achieving high performance in modern microprocessors. Branch predictors are used to keep the pipeline filled with instructions, before the control flow of all the in-flight instructions is known. The branch misprediction penalty can be broken down into two stages: the time it takes to discover that a branch was mispredicted and the time it takes to begin fetching instructions from the correct path. A great deal of previous work has focused on increasing the accuracy of branch predictors, i.e., reducing the number of branch mispredictions by improving the branch predictor [6]. Despite significant breakthroughs, branch predictors remain imperfect. Recognizing that future microprocessors will likely have to contend with branch mispredictions, this work proposes a mechanism to reduce the branch misprediction penalty by decreasing the time it takes to discover that a branch has been mispredicted.

We observe that when branches are mispredicted in an out-of-order machine, the wrong-path instructions following the branch may consume data values not properly initialized for the wrong-path instructions. This occurs when the mispredicted branch instruction is executed later than the wrong-path instructions that follow the branch– a scenario that arises when the mispredicted branch is data-flow dependent on a long-latency operation but the wrong-path instructions are not. When this occurs, the wrong-path instructions may exhibit illegal or unusual behavior. This behavior is interpreted to mean that a branch has been mispredicted before the mispredicted branch is executed. For example, if the instructions following a mispredicted branch consume an integer variable containing the value 0, but interpret this variable as a pointer, then dereferencing this variable on the wrong path causes a NULL pointer access. Using this hint, the processor can recognize that it has mispredicted a branch before the branch is executed.

This paper describes events that occur on the wrong path and that can be used to determine a branch was mispredicted *before* the branch is executed. We call these events "wrong-path events" (WPEs). We examine how these events can be used to increase processor performance. When a wrong-path event occurs in the processor, a recovery mechanism is used to determine which outstanding branch was mispredicted. If the recovery mechanism is unable to determine which branch was mispredicted, the processor can either continue on the wrong path or stop fetching wrong-path instructions. We propose and evaluate a recovery mechanism for branches that cause wrong-path events. Finally, we discuss the shortcomings of wrong-path events and propose topics for future research.

2. Motivation

The proposed mechanism addresses the branch resolution time. We demonstrate that focusing on this aspect of the branch misprediction penalty has potential to improve processor performance. Figure 1 shows the performance difference between a normal processor and an idealized processor running the SPEC2000 integer benchmarks. The idealized processor models the scenario where every mispredicted branch generates a WPE as early as possible which in turn triggers an early branch recovery. In the idealized processor, recovery is initiated for a mispredicted branch one cycle after it is placed in the instruction window (Section 4 describes the processor model in detail). Figure 1 shows that on average 11.7% IPC improvement is available.

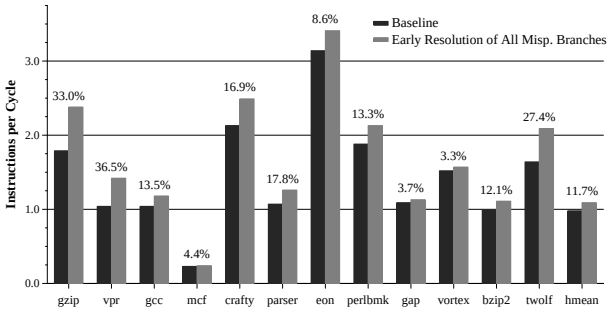

Figure 1. Performance potential when all mispredicted branches generate a WPE and resolve early.

3. Wrong Path Events

3.1. Overview

Wrong path events occur in an out-of-order machine when the instructions following a mispredicted branch are speculatively executed before the mispredicted branch instruction is executed. We evaluate wrong-path events that are caused by invalid memory accesses, mispredicted control flow operations and exception-generating arithmetic instructions. Wrong path events can be further broken down into two categories: hard and soft wrong-path events. A hard wrong-path event is an illegal operation, one that is allowed neither on the correct path nor on the wrong path. A soft wrong-path event is not an illegal operation, but is very unlikely to occur on the correct path of the program. Since a soft wrong-path event is unlikely to occur on the correct path of the program, when it does occur we guess that the processor is on the wrong path.

3.2. Memory Instructions

Wrong path events that result from memory operations include: dereferences of a NULL pointer, reads or writes to an unaligned address[1], writes to a read-only page, data reads to the pages that contain the executable image, reads or writes to addresses that are outside of the segment range and reads or writes that are TLB misses. A TLB miss is the only soft wrong-path event generated by a memory access; all others are hard wrong-path events in the Alpha ISA. Since TLB misses are soft wrong-path events, we must be careful not to mistake a TLB miss on the correct path for a wrong-path event. In order to insure that a TLB miss originating from correct-path code is not considered a wrong-path event, we require that the number of outstanding TLB misses surpass a threshold of three or more, before the misses are considered a wrong-path event. Although this threshold reduces the number of wrong-path events caused by wrong-path TLB misses, it also prevents correct-path TLB misses from incorrectly generating a wrong-path event. The following two examples are taken from the SPEC2000 integer benchmarks and illustrate how incorrect memory accesses occur on the wrong path.

Figure 2 shows a code segment from the `mrSurfaceList::shadowHit` function in the eon benchmark. In this example, the loop-terminating branch is mispredicted at the end of the loop, and the loop is incorrectly entered for an extra iteration. During the extra iteration, the program reads past the boundary of the `surfaces` array and sets the pointer variable `sPtr` to a non-pointer value, which in this case happens to be 0. A wrong-path event, a NULL pointer access, occurs when `sPtr` is dereferenced using the non-pointer value.

```
for (int i=0; i < length(); i++) {  // mispred the exit branch
  mrSurface *sPtr=surfaces[i]; // set sPtr to 0
  if (sPtr->shadowHit(...))    // and then access sPtr
    // ...
}
```

Figure 2. A NULL pointer access from EON.

Figure 3 shows an example based on gcc's `move_operand` function. The wrong-path event is an unaligned access that occurs when a union structure, having been initialized with an integer value, is used as a pointer by the wrong-path code. The variable `op` is a pointer to an `rtx_def` structure that contains an array of unions, `fld[1]`, and an integer variable, `code`, that indicates how the union should be interpreted, i.e., keeps track of the type of the union data. The `if` statement is used to check the type of the data value held in the `op->fld[0]` union. When the `if` statement is mispredicted, the wrong-path code interprets `op->fld[0].rtx` to be a pointer and dereferences it to load a 4-byte word from memory. The value of `op->fld[0].rtx` is odd and therefore generates an unaligned access, a wrong-path event, when it is dereferenced.

3.3. Control Flow Instructions

Control flow instructions cause wrong-path events when successive mispredictions are resolved before the oldest branch instruction is executed. If three branches are exe-

[1] In the Alpha ISA, unaligned addresses require an unaligned load/store opcode.

```
typedef union rtunion_def {  // this union contains
  int rtint;                 // both an integer
  struct rtx_def *rtx;       // and a pointer
  // ...
} rtunion;

typedef struct rtx_def {
  unsigned int code; // This variable determines whether
  // ...             // fld is interpreted as ptr or int
  rtunion fld[1];
} *rtx;

int move_operand (rtx op) {
  // ...
  if (op->code == L0_SUM)                  // mispredict
    return ((op->fld[0].rtx)->code == REG) // wrong path
  return (op->fld[0].rtint < 64 && ...);   // correct path
}
```

Figure 3. An unaligned access from GCC.

cuted and resolved as mispredicts while there are older unresolved branches in the processor, we find it is almost certain that one of the older unresolved branches was mispredicted. Therefore, resolution of three mispredicted branches while there are older unresolved branches in the processor is considered a wrong-path event, which we call the "branch under branch" event. The insight behind this wrong-path event is that branch predictor accuracy decreases significantly on the wrong path. The average misprediction rate for the branch predictor we use is 4.2% on the correct path and 23.5% on the wrong path. For this reason, misprediction resolutions on the wrong path are more likely than misprediction resolutions on the correct path.

A branch under branch event is a soft wrong-path event because branch mispredictions are not illegal operations and successive branches could be executed and resolved as mispredicts while there are older unresolved branches on the correct path as well. Therefore, branch under branch events can also occur on the correct path. However, a threshold requiring three branches to be executed and resolved as mispredicts while there is at least one older, unresolved branch in the processor ensures that branch under branch events rarely occur on the correct path[2].

A call return stack (CRS) underflow is a soft wrong-path event. We find that a 32-entry CRS underflows on the wrong path and not on the correct path when executing the SPEC2000 integer benchmarks. Therefore we consider the CRS underflow condition a wrong-path event.

The Alpha ISA requires instruction addresses to be aligned. An unaligned instruction fetch address is illegal and therefore considered a hard wrong-path event.

3.4. Arithmetic Instructions

When arithmetic instructions consume uninitialized values, they too can cause wrong-path events. Examples of arithmetic exceptions include division by zero or taking the square root of a negative number.

[2]Less than a combined total of 150 events in all benchmarks.

4. Methodology

We use an execution-driven simulator capable of correctly fetching and executing instructions on the wrong path and correctly recovering mispredicted branches that occur on the wrong path. The simulator models an 8-wide out-of-order machine with an instruction window that can hold up to 256 in-flight instructions. Because a less accurate branch predictor would provide more opportunity for early recovery from wrong-path events, a large and accurate branch predictor is used in our experiments. The branch predictor is a hybrid branch predictor composed of a 64K-entry gshare [14] and a 64K-entry PAs [19] predictor with a 64K-entry selector. We model a deep pipeline with a 30-cycle branch misprediction latency. The first-level data cache is 64KB, direct-mapped with a 2-cycle hit latency. The second-level unified cache is 1MB, 8-way set associative with a 15-cycle hit latency. The instruction cache is 64KB and 4-way set associative. All caches use 64B line sizes. On a second-level cache miss, the latency to main memory is 500 cycles. The size of the unified TLB is 512 entries.

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.

5. Experimental Evaluation

5.1. Coverage and Timing

In order to have an impact on performance, wrong-path events must occur often and they must occur early on the wrong path. We measure both the frequency of wrong-path events and how far onto the wrong path, in cycles, wrong-path events occur. Figure 4 shows the percentage of mispredicted branches that lead to wrong-path events. At least 1.6% of the mispredicted branches in all of the benchmarks lead to a wrong path event and the greatest percentage is from gcc where 10.3% of mispredicted branches lead to a wrong-path event. Figure 5 expresses the relative significance of wrong-path events and branch mispredictions for each of the benchmarks by showing the rate of mispredictions and wrong-path events per 1000 instructions.

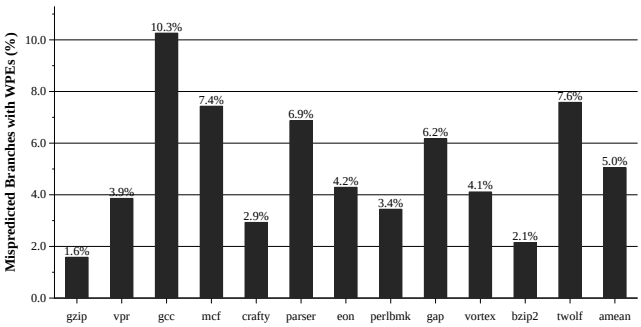

Figure 4. Percentage of mispred. branches with a WPE.

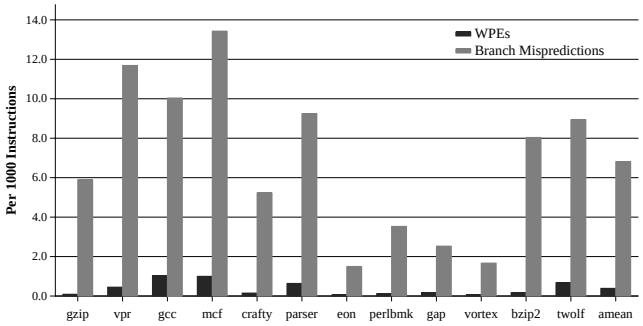

Figure 5. Branch misprediction and WPE rate.

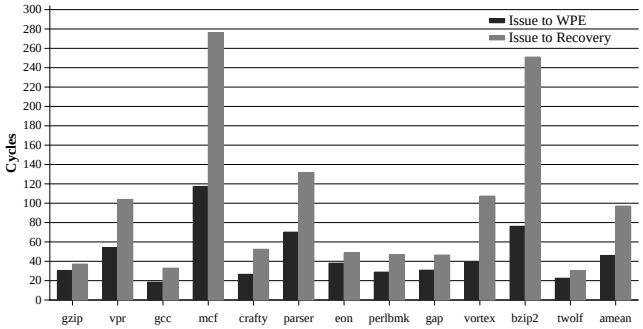

Figure 6. Wrong path cycles to WPE and recovery.

In order to find out when wrong-path events occur, we measure the average recovery time of all the branches that cause wrong-path events and the average number of wrong-path cycles before a wrong-path event occurs. Figure 6 shows the average time it takes to generate a wrong-path event and the average recovery time of the mispredicted branches that lead to wrong-path events. The leftmost bar for each benchmark in Figure 6 shows the average number of cycles from the time a mispredicted branch is issued[3] into the out-of-order window until the wrong-path event occurs; the rightmost bar shows the average number of cycles from the time the mispredicted branch is issued until the branch is resolved, which is when the recovery is initiated. After a mispredicted branch is issued into the window, the average time it takes to generate a wrong-path event is 46 cycles and the average time it takes to resolve the branch is 97 cycles. If recovery can be initiated for a mispredicted branch instantly when a wrong-path event occurs, that would yield a potential average savings of 51 cycles. The minimum potential savings is 7 cycles for gzip and the maximum potential savings is 176 cycles for bzip2. In bzip2 and mcf, the average time from issuing a mispredicted branch to its resolution is very large, because these two benchmarks have many mispredicted branches that depend on L2 cache misses.

The potential savings shown in Figure 6 by the average number of cycles between the wrong-path event and recovery time is significant and demonstrates the merit of leveraging wrong path events in a processor. However, the coverage of mispredicted branches that lead to wrong-path events is low as shown in Figure 5. That is, wrong-path events seem to occur fairly early[4] on the wrong path but do not occur very often. To utilize this scheme in a processor there is a need to increase the coverage of mispredicted branches by discovering additional wrong-path events.

Figure 7 shows the distribution of the different types of wrong-path events. "Branch under branch" events make up the majority of events in all benchmarks followed by NULL pointer accesses, unaligned accesses and accesses out of the segment range. On average, almost 30% of wrong-path events are generated by memory accesses. This fairly high percentage indicates that using register tracking [3] to compute load addresses early may aid in discovering wrong-path events earlier.

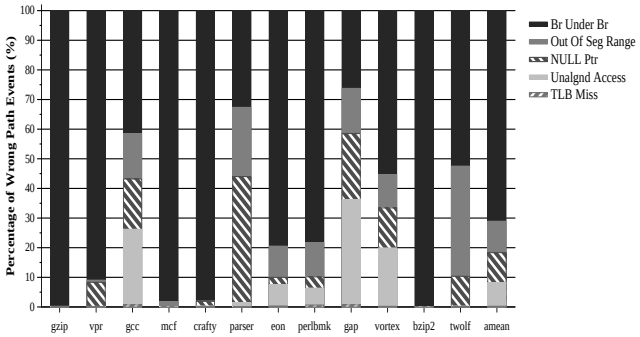

Figure 7. Distribution of wrong-path events.

5.2. Performance

In order to measure the performance potential of the available wrong-path events, we augmented the simulator enabling it to perfectly initiate recovery for a mispredicted branch as soon as a WPE occurs. Figure 8 shows the potential for performance improvement with the perfect recovery scheme as compared with the baseline processor, which does not recover when a wrong-path event occurs. Nine of the twelve benchmarks show some performance improvement. The maximum 1.7% IPC improvement is available from perlbmk and an average of 0.6% IPC improvement is observed over all the benchmarks.

The performance improvement is limited by both the low number of mispredicted branches with WPEs and an insufficient savings in cycles when a WPE occurs. Figure 6 shows that triggering the resolution of a mispredicted branch when a WPE occurs does have potential to reduce the average number of cycles on the wrong path. However, Figure 9

[3]In this paper, we use the term "issue" to indicate the placement of an instruction into the instruction window, i.e. insertion of an instruction into the reorder buffer. In our pipeline model an instruction gets fetched, decoded/renamed, issued, scheduled (when its source operands are ready), executed, and retired (when it is the oldest completed instruction in the window). Fetch-to-issue latency is 28 cycles, issue-to-execute latency is minimum 1 cycle, and the execute latency for a branch instruction is 1 cycle. Hence, the 30-cycle branch misprediction latency.

[4]More analysis on the time of occurrence of WPEs is provided in Section 5.2, which shows that WPEs do not occur early enough.

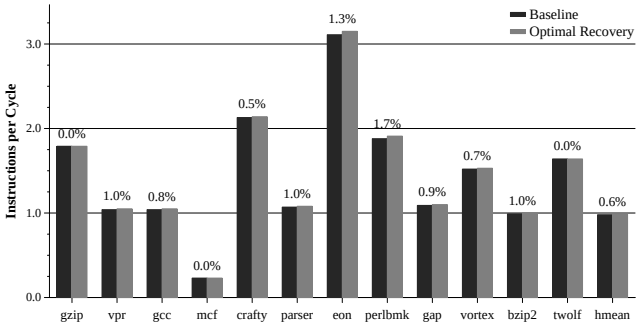

Figure 8. IPCs for baseline and optimal recovery.

provides additional insight for two benchmarks, mcf and bzip2, which are similar benchmarks in that they both expose the long memory latency used in our simulations. Figure 9 shows the cumulative distribution of the number of cycles after a WPE occurs until the associated mispredicted branch is resolved. The longer it takes to resolve the branch after a WPE occurs, the more potential savings there is for a WPE initiated recovery. Note that 30% of bzip2's mispredicted branches with WPEs save 425 cycles or more, as compared with only 8% of mcf's mispredicted branches with WPEs. This observation is reflected in the IPC results: bzip2 exhibits 1% IPC improvement and mcf exhibits no IPC improvement. In order to improve performance more significantly, additional wrong-path events must be discovered and they must be uncovered earlier on the wrong path.

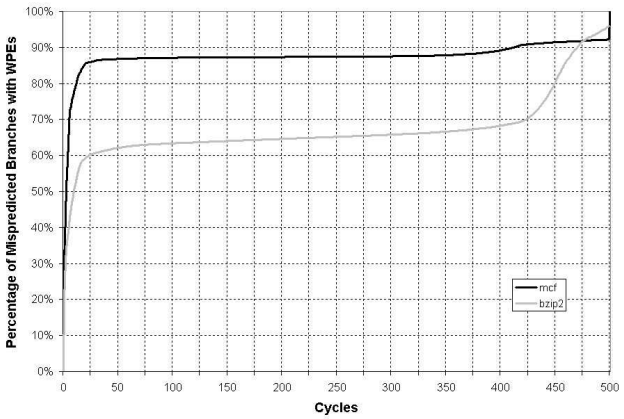

Figure 9. Cumulative distribution of the number of cycles between the occurrence of a WPE and the resolution of the associated mispredicted branch.

Another factor that limits the observed performance improvement is that some useful wrong-path prefetches are eliminated by initiating early misprediction recovery on the wrong path. As identified in previous work [5, 17, 16], the prefetching benefit of wrong-path instructions can and sometimes does result in important performance gains. We find that it is sometimes better to stay on the wrong path a little longer after the detection of a WPE in order to start the access of a load request that misses in the L2 cache and is needed later by a correct-path instruction, as opposed to initiating an early recovery and resuming execution on the correct path. This is especially true for mcf and bzip2, which experience more L2 cache misses than other benchmarks and also benefit considerably from prefetches generated by wrong-path instructions.

5.3. Gating Fetch

Another use for detecting wrong-path events is to prevent fetching new instructions once a wrong-path event occurs, i.e., gating fetch. Manne et al. report the benefits of gating pipeline stages when enough low confidence branches occur in the pipeline [13]. In a similar manner, when a wrong-path event occurs, the processor can stop fetching new wrong-path instructions for a potential energy savings.

6. A Realistic Recovery Mechanism

When a wrong-path event is detected by the processor, the processor needs to decide which instruction was mispredicted. The possible candidates for the mispredicted instruction are the branches (older than the WPE-generating instruction[5]) that are not yet executed when the WPE occurred[6]. If there is only a single unresolved branch in the processor, misprediction recovery is initiated for that branch. However, if there are multiple unresolved branches in the processor, we need a mechanism that decides which of these branches is mispredicted. To accomplish this, we propose a history-based predictor. The purpose of our proposal is to demonstrate that once a WPE occurs, it is possible to accurately predict which branch instruction was mispredicted[7]. We also discuss the issues involved in implementing a realistic recovery mechanism.

The proposed predictor memorizes the relationships between the instruction that generates the WPE and the instruction that is mispredicted. The design of this predictor is based on the following observations we make from our simulations (We do not include supporting data for these due to space constraints):

1. Many static instructions that cause WPEs do so repeatedly during program execution.
2. If an instruction A generates a WPE due to the misprediction of another instruction B that is N instructions older, then the next time instruction A causes a WPE, it is likely due to the misprediction of instruction B

[5]For the rest of the descriptions and discussion in Section 6, we only consider those branches that are older than the WPE-generating instruction.

[6]If there are no unresolved branches that are older than the WPE-generating instruction when a WPE occurs, then no action is taken, since the WPE must have occurred on the correct path.

[7]The potential performance improvement shown in Figure 8 perhaps does not currently justify the implementation cost of the proposed predictor and the recovery mechanism, but we hope that the predictor will become more important as future research increases the performance potential by discovering additional WPEs.

which is N instructions older. In other words, the "distance in instructions" between the WPE-generating instruction and the mispredicted instruction is persistent and predictable. The concept of distance is demonstrated in Figure 10a.

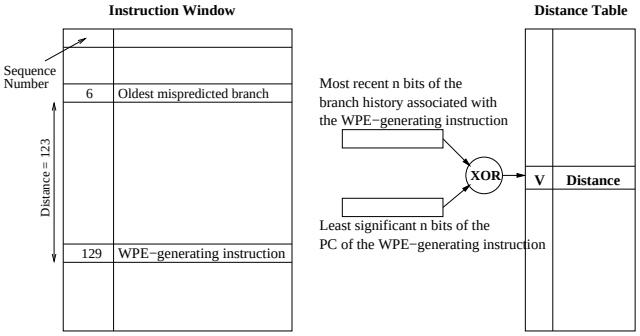

Figure 10. Distance concept and distance predictor.

A simple and initial implementation of this predictor, which we call the "distance predictor," is shown in Figure 10b. The predictor is indexed using a hash of the global branch history and the address of the WPE generating instruction. Each entry in the prediction table (distance table) has a valid bit and a distance field, which is a $log_2(instruction - window - size)$-bit value. The valid bits are initialized to 0 when a process starts execution. An entry is updated as follows: when the oldest mispredicted branch instruction in the machine retires after being resolved as a mispredict, the processor checks if a WPE occurred on the wrong path. If no wrong-path event is detected, the distance table is not updated. If there is a WPE, the address of the oldest WPE-generating instruction and the global branch history associated with it are used to index the distance table. The valid bit of the entry is set to 1, meaning this entry caused a WPE, and the distance between the WPE-generating instruction and the mispredicted branch is recorded in the distance field. The distance is calculated using the circular sequence numbers associated with each instruction used in modern processors [12]. The update of the distance table is very latency-tolerant and can take multiple cycles. Note that this update mechanism requires the processor to record the PC and the sequence number of the oldest WPE-generating instruction. Although multiple wrong-path events may occur under the same misprediction, we only record the oldest WPE for simplicity.

When a WPE is detected, the distance table is accessed using the PC of the instruction that generated the WPE and the global branch history associated with that instruction. If the valid bit of the accessed entry is not set, no prediction is made. In this case, the processor can gate fetch to save energy. If the valid bit is set, the processor uses the distance field of the entry to determine which instruction was mispredicted. The processor first checks the status of the instruction A that is N instructions older than the WPE-generating instruction, where N is the value in the distance field. Two cases are possible:

1. Instruction A is not a branch instruction, or it is a branch instruction that is already resolved, or Instruction A already retired (Instruction A may have already retired if predicted distance N is too large). Therefore, the predicted distance was incorrect. No recovery can take place, but the processor can gate fetch to save energy. We call this case Incorrect-No-Match.
2. Instruction A is an unresolved branch instruction. The processor initiates misprediction recovery for A. There are three distinct cases:
   (a) The prediction is correct if A is the oldest mispredicted branch in the processor.
   (b) If A is younger than the oldest mispredicted branch, the prediction was incorrect and the processor initiates recovery for a branch that should not have been executed anyway (Incorrect-Younger-Match).
   (c) If A is older than the oldest mispredicted branch, the prediction was incorrect and the processor flushes instructions that are on the correct path along with those that are on the wrong path (Incorrect-Older-Match). Therefore, this last case should be avoided by a good predictor. When the distance table is updated for this case, the valid bit of the entry that generated the prediction is set to 0 so that Incorrect-Older-Matches and possible deadlock are avoided in the future.

6.1. Evaluation of the Distance Predictor

The predictor is accessed when a WPE is detected and if there is at least one older unresolved branch in the instruction window. There are seven possible outcomes of the prediction:

1. Correct-Only-Branch (COB): There is only one unresolved branch in the window when the WPE is detected and this branch is the mispredicted branch. In this case, the output of the distance table is ignored. But the distance table is still updated when the single unresolved branch is retired.
2. Correct-Prediction (CP): The mispredicted branch is correctly identified by the predictor.
3. No-Prediction (NP): The predictor did not produce a prediction, because the valid bit was 0.
4. Incorrect-No-Match (INM): Described above.
5. Incorrect-Younger-Match (IYM): Described above.
6. Incorrect-Older-Match (IOM): Described above. If a recovery is initiated when the processor is on the correct path due to the detection of a wrong-path event on the correct path, the outcome is also considered IOM.
7. Incorrect-Only-Branch (IOB): There is only one unresolved branch in the window when the WPE is de-

tected, but this branch is not mispredicted and the processor is not on the wrong path. This case can occur if the detected WPE is a soft WPE[8] and the processor incorrectly decides that it is on the wrong path. Although this case is possible, we did not see an instance of it in our simulations. Therefore, we do not discuss results related to this case in this section.

Outcomes 1 and 2 (COB and CP) would correctly initiate early misprediction recovery. Outcomes 3 and 4 (NP and INM) would gate fetch until the mispredicted branch is resolved, and outcomes 5 and 6 (IYM and IOM) would initiate recovery on the incorrect branch, with outcome IOM being potentially the most harmful. Figure 11 shows that, on average, we can correctly initiate recovery for 69% of the mispredicted branches that result in WPEs. For 18% of these branches (outcomes 3 and 4) we can gate fetch from the time we discover the WPE until the mispredicted branch is resolved. Only for 4% of these branches does the predictor incorrectly identify an older branch as mispredicted. Hence, the potentially harmful mispredictions are rare. Figure 12 shows that a 1K-entry predictor achieves a correct prediction 63% of the time, while it incorrectly identifies an older branch only 4% of the time. Reducing the size of the predictor reduces the occurrence of CP and increases the occurrence of INM without significantly increasing the IOM and IYM outcomes. This indicates that the smaller predictor favors gating fetch instead of correctly initiating recovery.

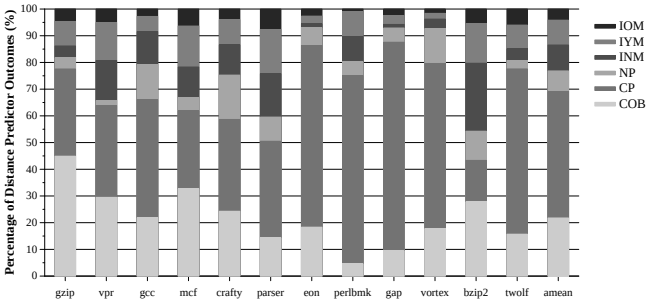

Figure 11. Accuracy of the 64K-entry predictor.

Using a 64K-entry distance predictor, the processor can correctly initiate early recovery for 3.6% of all mispredicted branches[9], averaged over SPEC2000 integer benchmarks. Recovery is initiated an average of 18 cycles before a mispredicted branch is executed. The resulting IPC improvement is 1.5% for perlbmk, 1.2% for eon, and 0.5% for gcc. IPC is not degraded for any benchmark. If instruction fetch is gated when the predictor outcome is NP or INM, the number of fetched wrong-path instructions decreases by 1% on

[8]Or if it is a hard WPE and the application architecturally generates illegal behavior.

[9]Recovery is incorectly initiated for an older correctly-predicted branch for 0.15% of the branch mispredictions (IOM outcome). We see no incorrect recoveries initiated (due to soft WPEs) when there are no mispredicted branches in the processor.

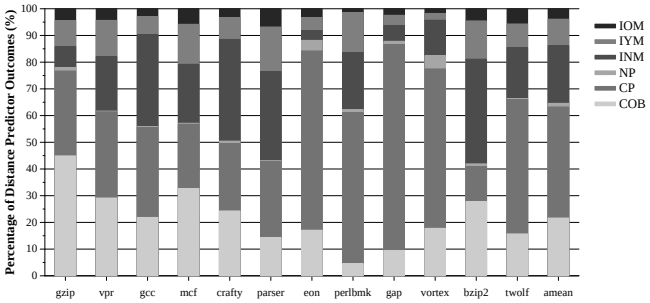

Figure 12. Accuracy of the 1K-entry predictor.

average compared to the baseline processor; 3% for eon, 4% for perlbmk, 1% for gcc, mcf, vortex, gap, and bzip2.

6.2. Avoiding Deadlock

A useful recovery mechanism should guarantee forward progress when a WPE is detected on the correct path and recovery is incorrectly initiated for a correctly-predicted branch. We avoid possible deadlock situations by invalidating the distance table entry that causes an IOM outcome.

An example deadlock situation is as follows: let's say an arithmetic exception occurs on the correct program path and the distance predictor initiates recovery for a branch that is not mispredicted. The branch eventually executes and the processor finds out that it incorrectly initiated recovery due to an IOM outcome from the distance predictor. The processor initiates recovery on the incorrectly-recovered branch and resumes execution on the correct path. Eventually, it encounters the instruction that generated the exception. As this instruction is on the correct program path, it generates the same exception again. If the valid bit of the entry that caused the IOM outcome was not reset the first time, the distance predictor would again use this entry and incorrectly initiate recovery on the same branch. As long as the valid bit is not reset, this situation could happen over and over and the program could be deadlocked.

Another example deadlock situation related to fetch gating is as follows: The processor detects a WPE on the correct path and incorrectly predicts that it is on the wrong path and gates fetch to save energy. However, because there are no mispredicted branches in the processor, no misprediction recovery will be initiated and instruction fetch will not be redirected. To avoid deadlock in such a case, the processor should un-gate fetch when all branches in the window are resolved.

6.3. Avoiding Performance Loss

If the distance predictor outcome is IOM, the processor overturns a correct prediction on the correct program path. Hence, our mechanism causes the processor to take the wrong path. Another wrong-path event may occur on this path, which may lead to another IOM outcome, which can put the processor on the wrong path at an even earlier point in the program. We did not see an instance of this

case in our simulations, but it can be very costly if it occurs. Therefore, a good implementation should prevent it from happening.

To avoid this case, we allow only one outstanding distance prediction at any given time. If a wrong-path event occurs and the distance predictor decides to initiate recovery on a branch, no other distance predictions are allowed until that branch executes and verifies both the branch prediction and the distance prediction.

6.4. Early Recovery for Indirect Branches

If the recovery initiated by the distance predictor is for a direct branch (i.e. a branch with only one possible target address), it is trivial to determine what address the program counter will be set to on recovery. However, if the recovery initiated by the distance predictor is for an indirect branch, the processor needs a way of determining the new target address to fetch from. To solve this problem, we extend the distance table entry to record the target address of an indirect branch. This entry is updated with the correct target address when a mispredicted indirect branch that leads to a WPE executes. When the distance predictor decides to initiate early recovery for an indirect branch using the same distance table entry, the recorded target address is used as the new fetch address. On average, with a 64K-entry distance predictor, this mechanism correctly predicts the target addresses of 84% of the indirect branches for which the distance predictor initiates recovery. With a 1K-entry distance predictor, 75% of these indirect branches are correctly predicted. Although costly in terms of area, this mechanism may be worthwhile to implement, because our simulations show that 25% of all branches that lead to wrong-path events are indirect branches.

7. Discussion and Future Research Directions

This paper presented the idea of wrong-path events, evaluated their occurrence and performance potential for branch prediction, and proposed a realistic recovery mechanism that can take advantage of the idea. We have analyzed both the advantages and shortcomings of wrong-path events in previous sections. In this section, we propose several research ideas that address the shortcomings. We also propose other areas of research that can take advantage of wrong-path events.

7.1. Addressing the Shortcomings of WPEs

Results and analysis discussed in Section 5 indicate that there are three major shortcomings that limit the gains obtained from a WPE-based mechanism:

1. WPEs do not occur very frequently. Hence, the percentage of mispredicted branches with WPEs is low.
2. WPEs do not occur early enough in the wrong path.
3. Staying on the wrong path a few more cycles is sometimes more useful than recovering early.

To address the first shortcoming, future research in wrong-path events should focus on discovering additional WPEs. The set of WPEs proposed in this paper provides a starting point and is by no means definitive. To come up with the "silver bullet set of WPEs," a detailed analysis of events occurring on the wrong path may be necessary. Microarchitectural behavior that is statistically more likely to occur on the wrong path than on the correct path can be exploited as WPEs. A "branch under branch event" described in this paper is an example of such behavior. Future research can target finding similar behavior using a comprehensive statistical analysis of the occurrence of various events on the wrong path vs. on the correct path.

Another avenue of research, which we believe is promising, would leverage the compiler to increase the occurrence of WPEs. The compiler can insert special, non-binding[10] instructions into the program that would generate a wrong-path event if an older branch is mispredicted. For example, the compiler can insert a special, non-binding load instruction that causes a NULL pointer dereference only if it is executed on the wrong path. If this special instruction is executed on the correct path, it computes a legal address and thus does not generate a WPE. The ISA needs to be augmented with these special instructions. Care must be taken to make sure that these instructions do not cause code bloat and do not tie up machine resources that are valuable for other instructions.

To address the second shortcoming, future research can focus on methods of rapidly executing wrong-path instructions. For example, using register tracking [3] to compute load addresses early may aid in discovering wrong-path events earlier. Having the compiler insert instructions that generate WPEs on the wrong path can also reduce the time it takes to detect a WPE on the wrong path.

Addressing the third shortcoming requires a better understanding of what makes a particular "wrong-path period"[11] useful for correct path execution. If the usefulness of a wrong-path period is predictable, perhaps a WPE-based recovery mechanism should not be employed or should be employed more carefully on a wrong-path period that is predicted to be useful. Previous research [17, 11, 16] focused on identifying the general effects of wrong-path execution on the correct-path execution, but, to the best of our knowledge, there is no body of research on distinguishing useful wrong-path periods from useless/harmful ones. We suggest that the usefulness of a particular wrong-path period is an important area of research in view of a WPE-based mechanism that can take advantage of the results.

7.2. Other Areas of Future Research

An orthogonal area of research is the exploration of situations other than early branch recovery where wrong-path events can be employed. We have explored the applicability of WPEs to value prediction in [2]. The proposed

[10] A non-binding instruction does not stall instruction retirement.

[11] Time elapsed from the fetch of a mispredicted correct-path branch until the resolution of the same branch and initiation of recovery.

idea of wrong-path events may apply to other methods of speculation, including but not limited to cache hit speculation [20], memory dependence speculation [15], and thread-level speculation [18]. The discovery of additional WPEs may be critical for the application of the idea of wrong-path events to other methods of speculation. If applicable to other forms of speculation, wrong-path events have the potential to be a generalized feedback mechanism used for the detection of misspeculation.

8. Related Work

8.1. Related Schemes

Glew proposes the use of "bad memory addresses" and illegal instructions as strong indicators of branch mispredictions [8]. Glew poses the question "What fraction of branch mispredictions lead to a bad memory address?" as a research topic. We extend and generalize Glew's notion of bad memory addresses to various instances of unusual and illegal program behavior (wrong-path events) including but not limited to bad memory addresses. We also evaluate the frequency of occurrence of these generalized set of events and analyze when they occur on the wrong path. Glew points out that it is not clear how to take advantage of wrong-path events, because a recovery mechanism needs to determine which unresolved branch in the processor is mispredicted. We describe a recovery mechanism that can take advantage of wrong-path events and show that a simple predictor can accurately predict which unresolved branch in the processor is mispredicted, once a wrong-path event is detected.

Jacobsen et al. explore branch confidence, or the likelihood that a branch is mispredicted, based on past program behavior [9]. They propose and evaluate mechanisms to determine branch confidence statically and dynamically. Confidence mechanisms can be used to conserve processor resources when there is little confidence that the processor is on the correct path. Manne et al. use branch confidence to gate low confidence speculative instructions from the early stages of the pipeline in order to save energy [13]. Their mechanism is called pipeline gating. The authors report a significant energy savings with negligible performance degradation. A low confidence branch in Manne et al. is analogous to a highly speculative wrong-path event. These previous mechanisms use information from past program behavior, the branch history information, to guess that a branch has been mispredicted. In contrast, a mechanism based on wrong-path events monitors the results from speculative instructions and, based on this feedback, determines whether the processor is on the correct path.

Jimenez et al. observe that access time is an important design point in a branch predictor [10]. To address this design point, they propose an overriding predictor scheme, where a larger, slower, and more accurate predictor overrides the prediction from a smaller, faster, less accurate predictor. Wrong path events provide a mechanism to override a previous prediction similar to the larger, more accurate predictor. However, in the case of wrong-path events, the overriding prediction is made based on program behavior exhibited after the branch is predicted, and is not based solely on branch history information.

Falcón et al. propose the use of predictions made for younger branches to re-evaluate the prediction made for an older branch [7]. Initially, a branch is predicted using a traditional branch predictor, the "prophet", which uses past history. Later, bits from the branch's global history and predictions made for younger branches are together used to index an overriding "critic" predictor to generate a more accurate prediction for the branch. This mechanism uses future speculation information (predictions made for younger branches) to refine the prediction made for an older branch. In contrast, the mechanism we propose utilizes many pieces of wrong-path information (called wrong-path events), including the results of the wrong-path instructions, to identify that the processor is on the wrong path.

8.2. Schemes to Reduce the Misprediction Penalty

Bondi et al. propose a misprediction recovery cache for deep, superscalar pipelines [4]. The misprediction recovery cache reduces the recovery time of mispredicted branches by caching the decoded instructions that follow the most frequently mispredicted branches. When a misprediction is discovered, the pipeline is flushed. While the fetch and decode stages of the pipeline are warmed with correct-path instructions, the execution stage of the pipeline draws decoded instructions from the misprediction recovery cache. Whereas a misprediction recovery cache addresses the time it takes to recover from a misprediction after the branch is executed, a wrong-path event mechanism addresses the time it takes to discover the misprediction.

Aragón et al. propose a mechanism that fetches, decodes and renames, but does not execute instructions from the alternative paths of low-confidence branches [1]. Once a misprediction is detected, the instructions from the correct path are immediately available to the execution core. This scheme is also intended to reduce the penalty after the mispredicted branch is discovered. This mechanism is different from that of a wrong-path events mechanism, which is intended to reduce the time it takes to discover that the branch is mispredicted. Therefore, an approach based on wrong-path events and the schemes proposed by Bondi et al. and Aragón et al. are orthogonal and can be combined for greater performance gains.

9. Conclusion

In this paper, we propose and evaluate a novel mechanism to resolve mispredicted branches before they are executed in the processor. The purpose of this mechanism is to improve processor performance by helping to insure that the processor remain "on the correct path" as as much of the time as possible. We observe that wrong-path instruc-

tions can exhibit unexpected or illegal behavior. We call an instance of this behavior a wrong-path event and use it as a trigger to initiate a branch resolution before the mispredicted branch executes. We show that wrong-path events affect an average of 5% of the mispredicted branches in the SPEC2000 integer benchmarks and occur an average of 51 cycles before the mispredicted branch is executed. This mechanism has potential for performance benefit for nine of the twelve SPEC2000 integer benchmarks, but is limited by three factors: the coverage of mispredicted branches affected by WPEs, how far onto the wrong path the WPE occurs and finally by negating potentially beneficial prefetching effects generated by wrong-path instructions. In light of these limitations, we propose new ideas to explore.

We propose a recovery mechanism that utilizes the concept of distance between a wrong-path event and the mispredicted branch and achieves a low misprediction rate. We summarize the shortcomings of wrong-path events and propose future areas of research to address these shortcomings in order to exploit the potential of the idea. Although we have not found the "silver bullet" set of wrong-path events, our results show that the idea of wrong-path events does provide a significant opportunity for performance improvement, which we hope will be exploited with future research.

10. Acknowledgements

We thank Andy Glew and Mike Fertig for their comments on earlier drafts of this paper. We also thank members of the HPS research group for their comments and suggestions, and the fertile environment they provide.

11. References

[1] J. L. Aragon, J. Gonzalez, A. Gonzalez, and J. E. Smith. Dual path instruction processing. In *Proceedings of the 2002 International Conference on Supercomputing*, 2002.

[2] D. N. Armstrong, H. Kim, O. Mutlu, and Y. N. Patt. Wrong path events: Exploiting unusual and illegal program behavior for early misprediction detection and recovery. Technical Report TR-HPS-2004-002, HPS Technical Report, 2004.

[3] M. Bekerman, A. Yoaz, F. Gabbay, S. Jourdan, M. Kalaev, and R. Ronen. Early load address resolution via register tracking. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 306–315, 2000.

[4] J. O. Bondi, A. K. Nanda, and S. Dutta. Integrating a misprediction recovery cache (MRC) into a superscalar pipeline. In *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 14–23, Dec. 1996.

[5] M. G. Butler. *Aggressive Execution Engines for Surpassing Single Basic Block Execution*. PhD thesis, University of Michigan, 1993.

[6] M. Evers and T.-Y. Yeh. Understanding branches and designing branch predictors for high-performance microprocessors. *Proceedings of the IEEE*, 89(11):1610–1620, Nov. 2001.

[7] A. Falcón, J. Stark, A. Ramirez, K. Lai, and M. Valero. Prophet-critic hybrid branch prediction. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, 2004.

[8] A. Glew. Branch and computation refi nement. Unpublished Manuscript, University of Wisconsin, Jan. 2000.

[9] E. Jacobsen, E. Rotenberg, and J. E. Smith. Assigning confi dence to conditional branch predictions. In *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 142–152, 1996.

[10] D. A. Jiménez, S. W. Keckler, and C. Lin. The impact of delay on the design of branch predictors. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, pages 67–76, 2000.

[11] S. Jourdan, T.-H. Hsing, J. Stark, and Y. N. Patt. The effects of mispredicted-path execution on branch prediction structures. In *Proceedings of the 1996 ACM/IEEE Conference on Parallel Architectures and Compilation Techniques*, pages 58–67, 1996.

[12] R. E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[13] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 132–141, 1998.

[14] S. McFarling. Combining branch predictors. Technical Report TN-36, Digital Western Research Laboratory, June 1993.

[15] A. Moshovos, S. E. Breach, T. N. Vijaykumar, and G. S. Sohi. Dynamic speculation and synchronization of data dependences. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997.

[16] O. Mutlu, H. Kim, D. N. Armstrong, and Y. N. Patt. Understanding the effects of wrong-path memory references on processor performance. In *Third Workshop on Memory Performance Issues*, 2004.

[17] J. Pierce and T. Mudge. The effect of speculative execution on cache performance. In *Proceedings of the International Parallel Processing Symposium*, pages 172–179, 1994.

[18] G. S. Sohi, S. Breach, and T. N. Vijaykumar. Multiscalar processors. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 414–425, 1995.

[19] 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.

[20] A. Yoaz, M. Erez, R. Ronen, and S. Jourdan. Speculation techniques for improving load related instruction scheduling. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 42–53, 1999.