Restricted Dual Path Execution

Matthew Farrens

Computer Science
Department
University of California, Davis
Davis, CA
(farrens@cs.ucdavis.edu)

Timothy Heil
James E. Smith

Department of Electrical
and Computer Engineering
University of Wisconsin, Madison
Madison, WI
(heil@ece.wisc.edu)
(jes@ece.wisc.edu)

Gary Tyson

Department of Electrical
Engineering and Computer Science
The University of Michigan
Ann Arbor, MI
(tyson@eecs.umich.edu)

Technical Report No. CSE-97-18
Computer Science Department
University of California, Davis, CA 95616

tel: (530) 752-7002, fax: (530) 752-4767
farrens@cs.ucdavis.edu
{heil,jes}@ece.wisc.edu
tyson@eecs.umich.edu

November 24, 1997

Abstract

Restricted Dual Path Execution (RDPE) reduces branch misprediction penalties by selectively forking a second path and executing instructions from both paths following certain conditional branch instructions. Dynamically gathered confidence information is used to identify the branches most likely to be mispredicted (and thus most likely to benefit from dual-path execution). A branch forking policy defines the behavior of RDPE when a low confidence branch prediction is encountered while two paths are already being executed. RDPE restricts the number of simultaneously executing paths to two, because instruction fetch rates far exceed the capability of the pipeline to retire a given branch before others must be processed.

Cycle-level simulation is used to evaluate the effectiveness of RDPE with two different confidence mechanisms and 3 different pipeline lengths. Simulation results show RDPE can provide an approximate 7.5% reduction in overall execution time. However, it is also shown that both branch mispredictions and low confidence predictions tend to occur in clusters, limiting the effectiveness of RDPE.

Restricted Dual Path Execution

Abstract

Restricted Dual Path Execution (RDPE) reduces branch misprediction penalties by selectively forking a second path and executing instructions from both paths following certain conditional branch instructions. Dynamically gathered confidence information is used to identify the branches most likely to be mispredicted (and thus most likely to benefit from dual-path execution). A branch forking policy defines the behavior of RDPE when a low confidence branch prediction is encountered while two paths are already being executed. RDPE restricts the number of simultaneously executing paths to two, because instruction fetch rates far exceed the capability of the pipeline to retire a given branch before others must be processed.

Cycle-level simulation is used to evaluate the effectiveness of RDPE with two different confidence mechanisms and 3 different pipeline lengths. Simulation results show RDPE can provide an approximate 7.5% reduction in overall execution time. However, it is also shown that both branch mispredictions and low confidence predictions tend to occur in clusters, limiting the effectiveness of RDPE.

1. Introduction

Changes in control flow (branches) are a well-known impediment to achieving high performance in pipelined processors. Conditional branches, in particular, cause problems because the branch condition is resolved several pipeline stages after the instruction fetch stage. This can lead to holes in the pipeline and lost cycles unless special steps are taken.

The current method of choice for dealing with a conditional branch is to predict its outcome and speculatively execute instructions down the predicted path. If the prediction is correct, useful work is done, and performance losses are minimized. If the branch prediction is incorrect, however, cycles are wasted fetching and executing instructions that must be discarded. And there may be an additional penalty for restarting instruction fetching down the correct branch path. As pipelines deepen and issue widths increase, the penalty for fetching down the wrong path (the branch mispredict penalty) is becoming a substantial fraction of overall performance loss.

One way to avoid paying the branch mispredict penalty is to fetch instructions from both the fall-through and target locations, which guarantees the machine is executing the correct stream of instructions. When the conditional branch is eventually resolved, one of the two paths must be discarded.

Unfortunately, given the known distribution of branch instructions and the number of instructions that are often "in flight" in a current processor, it is inevitable that more branch instructions will be encountered on one (or both) of the streams before the initial branch condition is resolved. Providing sufficient resources to continue to split and fetch on each of these new branches can clearly lead to a cascading effect, requiring an exponentially increasing amount of hardware and potentially significantly impeding the progress of the correct stream.

Another way to reduce the branch mispredict penalty is to minimize the number of times fetching occurs from the wrong path. This fact has led to extensive work on improving the quality of branch predictors in order to allow the processor to speculatively execute past branches with a high degree of confidence. Average branch prediction accuracies have climbed substantially throughout the 1990's.

In this paper we present a technique that combines the above two approaches. "Restricted Dual Path Execution" (RDPE) allows the processor to fetch from both a target and fall-through path, but restricts the number of simultaneously executing paths to two. This restriction places reasonable limits on the amount of hardware necessary, and still provides substantial performance gains. These gains come from two factors - the growth in accuracy of branch predictors, and from studies of the characteristics of branch behavior [Tyso94] which indicate that certain branches are much more difficult to predict than others. The fact that different branches can be predicted with different accuracies allows us to divide conditional branch predictions into high and low confidence sets, and use the presence of a given branch in the low or high confidence set to indicate that path forking should or should not occur.

In this paper we will present an overview of previous work on fetching from multiple paths, then present RDPE in greater detail. This will be followed by a description of the simulation environment we used, the results of our cycle-level simulations measuring the effectiveness of RDPE, and a discussion of how to deal with branches that are encountered while fetching down dual paths. Finally we will outline some areas that need further study.

2. Prior Work

Processors have performed variants of dual path fetching for some time. Early IBM 3168 and 3033 mainframes could fetch instructions from both paths following a conditional branch, though instructions from only one path could be decoded and executed [CoFP79]. Similarly, the IBM POWER1 processor statically predicts all conditional branches not taken. However, the processor fetches some instructions from the taken path as a hedge, which results in a single cycle penalty when the branch is taken [WeSm94]. Also, the MIPS R10000 requires a one cycle delay to decode and predict branches. This cycle is used to fetch instructions sequentially following the branch. If the branch is predicted taken, the extra fetched instructions are stored in a Resume Cache [Gwen94] in a partially decoded state. If the branch is discovered to be not taken, then the sequential instructions are quickly recovered from the Resume Cache, eliminating one cycle from the misprediction penalty.

Several attempts have been made to expand the number of simultaneously executing paths. Uht referred to following both paths after every branch as "Eager Execution". Eager Execution requires following many tens of paths and is studied using an ideal model in [WaUh90]. Disjoint Eager Execution (DEE) [UhS95] restricts execution to the path that is the most likely to be correct out of all unexecuted possible paths. This involves calculating the cumulative probability that all preceding branches have been correctly predicted. Each cycle, DEE follows the paths that it considers most likely to be correct. DEE can follow multiple paths simultaneously, as long as hardware resources are available. The implementation studied uses a single fixed prediction probability for all branch predictions, resulting in paths being executed in a fixed pattern.

As part of an instruction level parallelism study, Wall considers branch fanout [Wall93], which also executes both paths following conditional branches. To avoid an exponential increase in the number of paths, a fanout limit is placed on the number of paths that are simultaneously executed. The branches chosen for fanout are selected using confidence based on static prediction probabilities assigned through program profiling. Branch fanout is studied with and without branch prediction up to a fanout limit of four.

3. Restricted Dual Path Execution (RDPE)

As outlined above, RDPE provides hardware to fetch down multiple paths, but restricts the number of simultaneously executing paths to two. It works as follows: As each branch is encountered in the dynamic instruction stream, the branch predictor is queried for a prediction direction. The branch confidence mechanism is also consulted, and if the confidence mechanism indicates that the branch prediction is in the high confidence set, then the predictor is considered correct and the branch is treated normally (no dual-path action takes place). However, if the confidence mechanism reports that the branch is in the low-confidence set, then an attempt is made to begin fetching from both the fall-through and the branch target (the dual-path hardware is enabled). Once the branch condition is resolved, the appropriate stream is invalidated and the dual-path hardware is freed up.

Since the number of simultaneous paths is limited to two, there must be a policy for what should occur when the dual-path hardware is in use and one of the two paths encounters another branch. The approach we used is to ignore the confidence information and use whatever prediction was generated directly. This approach requires two paths into the branch prediction logic, and the branch history table must not updated incorrectly. However, we do not expect this to be prohibitively expensive. There are several other approaches that can be used, such as stopping fetch when this situation is encountered, or stopping if the prediction is a low-confidence one but continuing if it is a high-confidence branch - a nice description of them is in [JaRS96].

The following aspects are central to determining the effectiveness of RDPE:

- The quality of the branch predictor. Increases in branch prediction accuracy will lead to fewer and fewer mispredictions, which decreases the likelihood that multiple incorrect predictions will be in the pipeline at the same time.
- The ability of the branch confidence mechanism to identify a high percentage of the incorrectly predicted branches without including a large number of correctly predicted branches in the low confidence set. This will allow the dual path execution resources to be allocated to those branches for which they are designed.[1]
- Pipeline depth. As pipeline depths (and issue widths) increase, there will be more and more instructions between the instruction fetch point and the branch condition resolution point, increasing the branch misprediction penalty.

In the simulation section, we examine the impact of some of these on the performance of RDPE, in an attempt to determine whether a DPE approach becomes more or less attractive as more sophisticated predictors and confidence mechanisms are employed. In the rest of this section, we discuss in more detail how branches are assigned to the high and low confidence sets.

3.1. Branch Confidence Mechanisms

Calculating the probability of a given branch prediction being correct is central to the idea of RDPE. In order to make this work, the branch confidence mechanism separates branch predictions into high and low confidence sets. In this work two different approaches to calculating branch confidence are modeled -- the Resetting Counters Confidence Mechanism [JaRS96] and the use of internal predictor state [LiTy96]. Each mechanism is briefly described below.

---

[1] Ideally, the set of low confidence branch predictions would exactly equal the set of all mispredictions. Of course, this can not be implemented in practice (or we could solve the branch prediction problem completely by reversing all predictions in the low confidence set!). Rather, we try to concentrate a large number of the mispredictions into the low confidence set -- realizing that some will be missed and some correct predictions will be included.

3.1.1. Resetting Counters Confidence Mechanism

This confidence mechanism consists of a table of resetting m-bit counters, and is indexed by the XOR of a truncated program counter and a global branch history register (GBHR). The program counter points to a conditional branch instruction that is being predicted and the GBHR indirectly indicates the control path followed before arriving at the branch in question. For a table of 2n entries, n low-order program counter bits are XORed with an n-bit GBHR; this is essentially the same mechanism used by the gshare branch predictor [McFa93].

For a given conditional branch, the confidence table's counter is incremented if the branch is predicted correctly, up to a maximum counter value (fifteen in the implementation used in this paper). The counter is reset to zero when the branch is mispredicted. If the counter is less than its maximum value, the prediction is considered to be a low confidence one; if it is equal to its maximum value, the prediction is assigned high confidence.

In [RoBS96] it is shown that this algorithm works nearly as well as more complex algorithms that keep exact histories of branch outcomes. The resetting counters work well because a mispredicted branch is more likely to be mispredicted in the near future. With a 4-bit resetting counter, a branch is considered low confidence for the next fourteen occurrences following a misprediction. This approach can be easily applied to any predictor model, since the confidence mechanism is separated from the predictor implementation.

3.1.2. Internal State Mechanism

A simpler approach to calculating branch confidence uses the internal state of the branch predictor to determine the quality of the prediction. For a two-level correlating predictor, for example, the pattern that comes out of the BHR is not sent directly to the predictor - instead, hardware checks to see if the pattern is in the "predicting set", which consists of those patterns which are anticipated to have good prediction accuracy. If it is, the branch is considered highly predictable, and the pattern is forwarded on to the second level of the predictor. However, if the pattern is not in the predicting set, then it is a branch that is considered difficult to predict accurately and it is treated as a low-confidence branch. Branch predictions marked as low confidence attempt to initiate a dual path fetch. If the second path is available, then DPE execution can begin; otherwise, the original, low confidence prediction is performed. A box diagram of the technique is presented in Figure 1.

For other types of branch predictors, differences in the internal state of the predictor will lead to differing mechanisms to identify the predicting set. For instance, in a combining predictor employing 2 sub-predictors, the predicting (high-confidence) set might consist of those predictions in which the sub-predictors agree, and the low-confidence set would consist of those

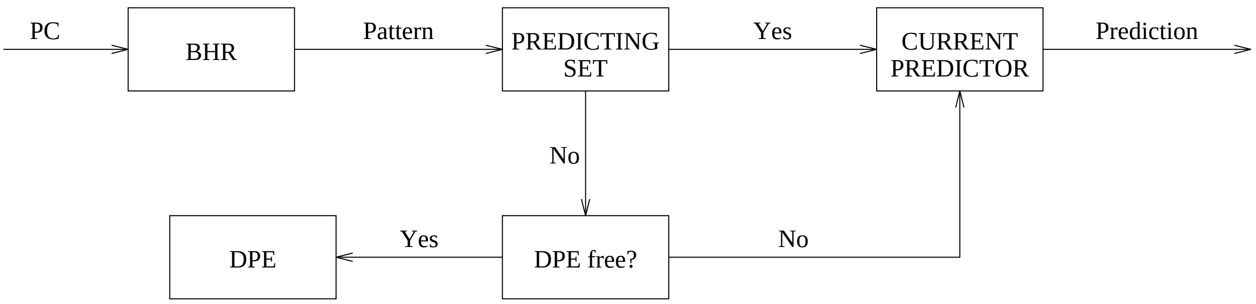

Figure 1: Internal State Mechanism

where the sub-predictors disagree.

When this approach was applied to a two-level PAg predictor model with a history depth of 8, a predicting set consisting of 4 history states (out of 256) was shown to identify over 85% of all branches as high-confidence while correctly marking 55% of all mispredictions as low confidence.

4. Evaluating the Effectiveness of RDPE

Fetching down two paths can cause contention for many different processor resources. Therefore, a detailed cycle-level simulation was performed in order to measure the true effectiveness of RDPE. The simulator used was a modified version of one of the simulators in the SimpleScalar [BuAu97] toolset, performing out-of-order issue, execution and completion on a derivative of the MIPS instruction set architecture. The benchmarks used in the study were the SPEC95 integer programs.

4.1. The Processor Model

As mentioned above, the processor modeled in this study supports out-of-order issue, execution, and completion. It was modified to support a fetch width of 8 instructions, which when executing in dual-path mode is shared by the two fetch paths; each path can fetch up to the next branch encountered. The decode bandwidth is 8 instructions each cycle, again shared by both paths when executing in dual-path mode. The number of functional units was increased slightly from what would likely be chosen for a conventional 8-way superscalar in order to accommodate the increased instruction flow while executing from two paths; this may seem unnecessary at first glance because the fetch and decode rates remain unchanged, but the fact that dependencies do not exist between the paths enables the actual fetch/decode rates to approach the limit of 8 per cycle. Table 1 lists the model parameters used for all simulations (unless otherwise stated.)

Table 1. Simulation Parameters used in Study

Superscalar Processor		**Memory System**	
Fetch Rate (max)	8 instructions	**L1 Data Cache**	64K 4-Way LRU
Issue Rate (max)	4 ALU, 2 Load/Store, 1 Mult/Div	**L1 Instr. Cache**	64K 2-Way LRU
Predictors		**Confidence Mechanisms**	
2-Bit Bi-modal	8K counters (16Kbit)	**Resetting Counter**	Counter Size 2 bits
			Counters have 8K entries (16Kbit)
PAg	History Depth 8	**Internal State**	Confident BHR Patterns -
	Level 1 Table 1024 entries (8Kbit)		All Taken, All Not Taken
	Level 2 Table 256 entries (512 bit)		Confident PHT States 0,3

4.2. Benchmarks

The benchmarks used in this study were the eight SPEC95 integer benchmarks. The integer benchmarks were chosen because they tend to be more irregular and have higher misprediction rates than the floating point benchmarks. To allow most benchmarks to run to completion, the inputs were trimmed to yield runs of less than 200 million instructions. m88ksim and vortex run longer, but only the first 200 million instructions were simulated. The benchmark characteristics are listed in Table 2.

Table 2. Detailed Benchmark Information

Program	Input File	Total # of Instruction Executions (in millions)	Total Branch Executions (in millions)
compress	40000 e	103.2	20.7
gcc	gcc.i	203.5	40.6
go	9 9	132.9	20.2
ijpeg	specmun.ppm	129.3	11.8
li	test.lsp	202.2	47.7
m88ksim	ctl.raw	200.0	45.9
perl	scrabbl.in	41.5	8.0
vortex	vortex.in	200.0	31.8

5. Simulation Results

The first set of experiments run were to measure the effectiveness of a given implementation of RDPE using different confidence prediction mechanisms. In these simulations, the branch predictor selected was a PAg predictor, and the number of pipeline stages in the machine was 5 (4 before the branch resolution stage) Five different configurations were simulated:

(1) Configuration 1 (the base case), which used only branch prediction (Pag) and speculative execution (no dual path execution)

(2) Configuration 2, which used a Pag branch predictor and the Internal State Confidence Mechanism (ISCM) in conjunction with dual path execution

(3) Configuration 3, which used a Pag branch predictor and the Resetting Counters Confidence Mechanism (RCCM) in conjunction with dual path execution

(4) Configuration 4, which used an oracle (perfect) confidence mechanism, modeled by placing all correct predictions in the high-confidence set and all incorrect predictions in the low-confidence set, in conjunction with dual path execution

(5) Configuration 5, which used an oracle (perfect) branch predictor (no dual path execution required).

Figure 2 presents the results of these simulations. Several things are of interest in figure 2. The overall performance improvement can be seen to be about 7.5% for the Resetting Counters Configuration, with a slightly smaller gain seen using the Internal State Configuration. In general, there is little difference between the two confidence mechanisms. However, in several cases the Resetting and Internal State Configurations actually provide slightly better overall system performance than the Perfect Confidence Configuration. We believe this unexpected result is due to the fact that mispredictions can at times fetch useful information into the caches, thus giving a slight overall improvement in IPC. In addition, as discussed in the next subsection, the clustering of mispredictions in the program interferes with even the perfect confidence predictor's ability to eliminate all mispredictions. This fact, in addition to the positive effects of DPE on cache prefetch when no misprediction occurs, leads to this surprising conclusion.

Each configuration provides improved performance when compared to the base case, but each also falls well short of the improvements possible (shown by the perfect predictor configuration). This is in part because mispredicted branches tend to occur in clusters (or clumps).

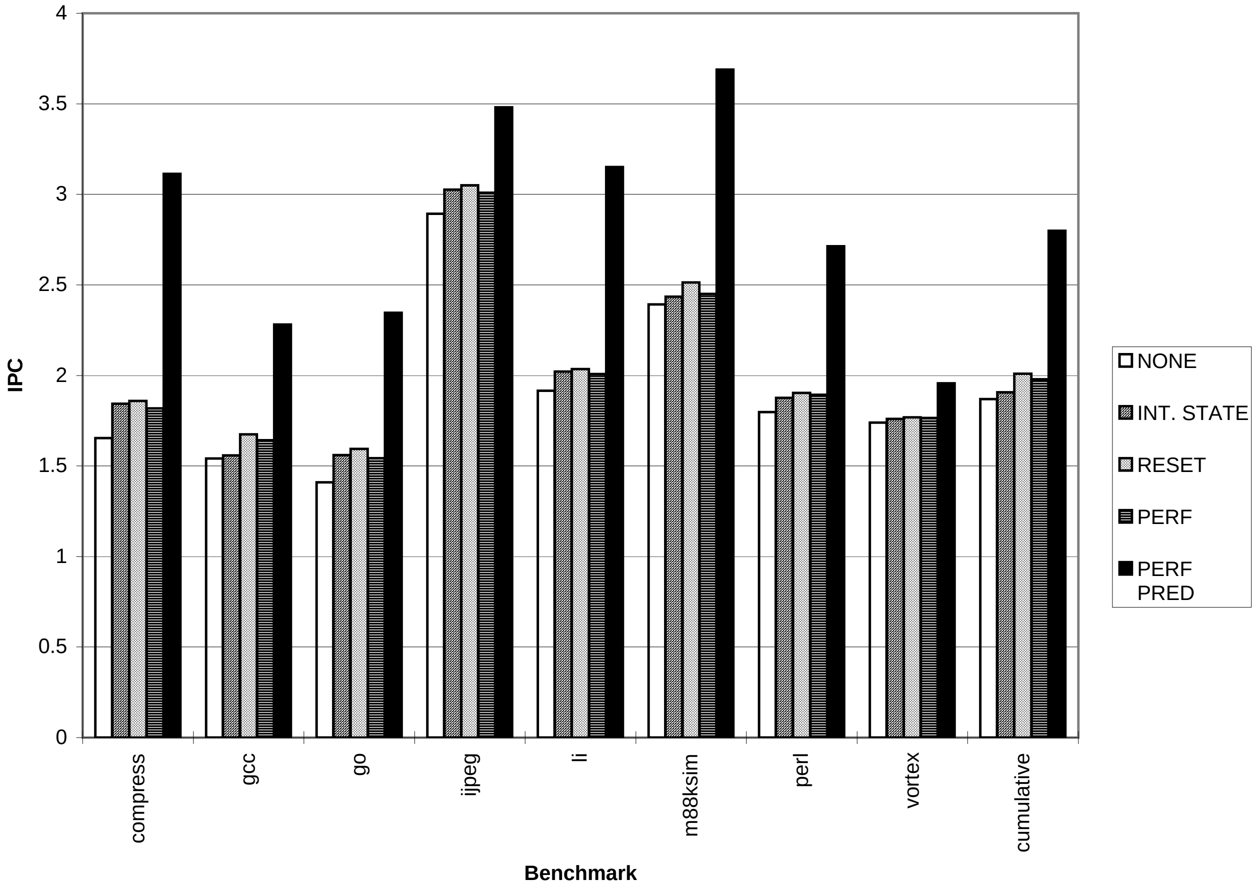

Figure 2: Performance of 5 Different Configurations

5.1. Limits to Effectiveness of RDPE

The impact of the forking policy on performance depends on the distance between mispredicted branches, or how often mispredicted branches are "clumped" together. Previous work [JaRS96] indicated that branch mispredictions showed a high degree of clumping, thereby requiring more advanced forking policies. However, the authors only looked at a single branch prediction strategy when doing that study, so the question of whether or not clumping is inherent to programs or dependent on the quality of the branch predictor was not addressed. One could expect that as branch predictors increase in quality the distance between low-confidence branches could decrease as well. In order to attempt to answer this question, another set of simulations was done to measure the "clumping" for 3 different branch predictors.

The 3 predictors simulated were a simple bi-modal predictor (as used in the Alpha 21164), a correlating predictor (as used in the Pentium Pro) and a combination predictor (as proposed in the Alpha 21264 processor). Figure 3 shows the results of the clumping experiments for the bi-modal predictor, figure 4 the correlating predictor results, and figure 5 the results for the combination predictor.

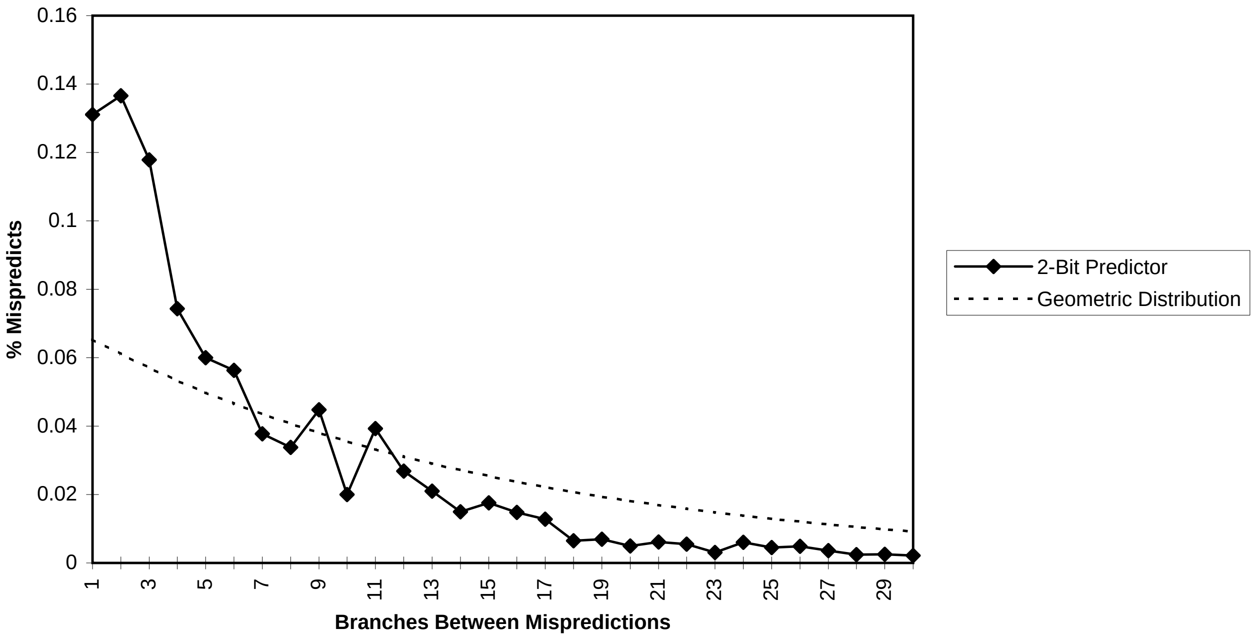

Figure 3: Branch Mispredict Distribution for bi-modal predictor

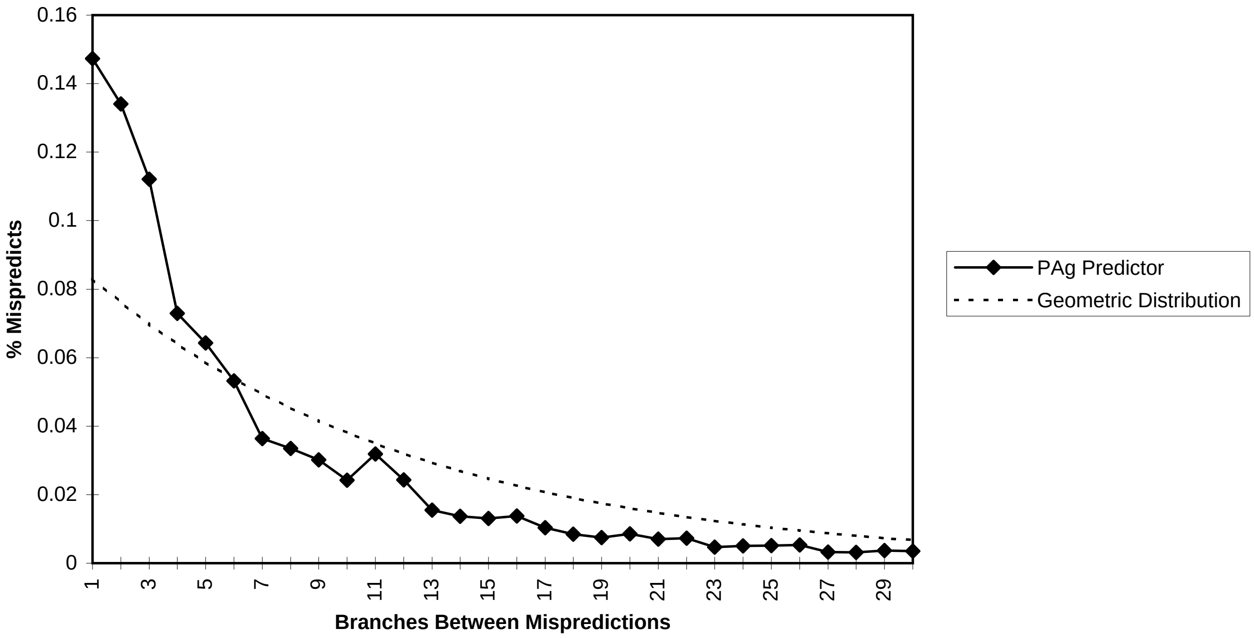

Figure 4: Branch Mispredict Distribution for correlating predictor

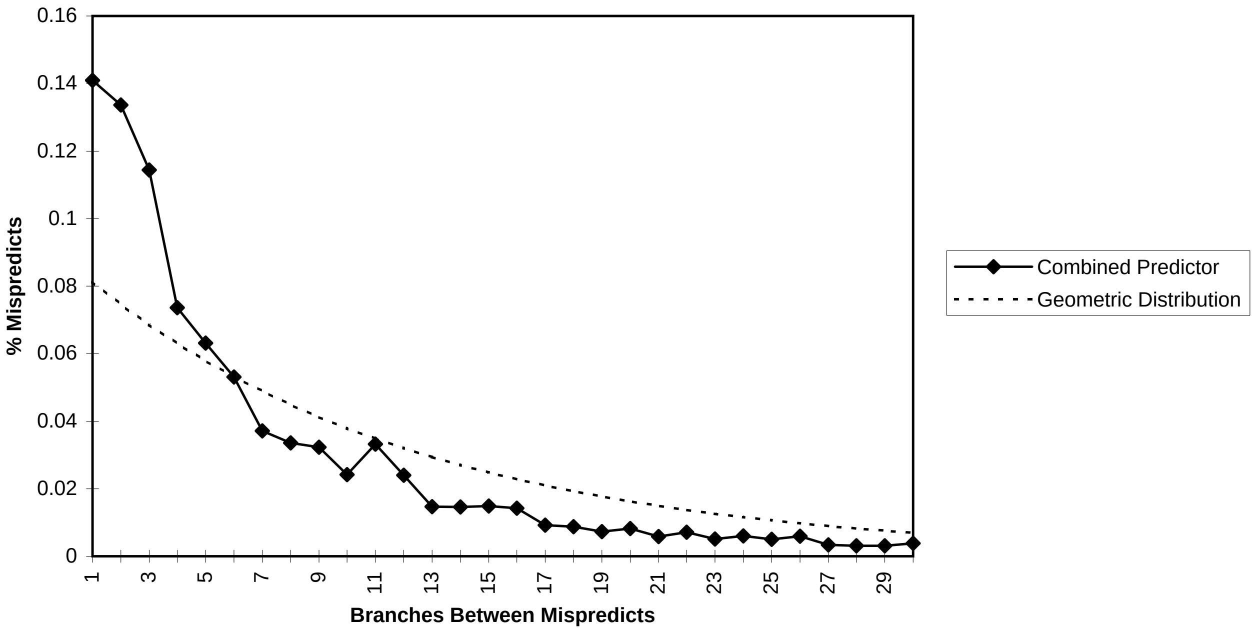

Figure 5: Branch Mispredict Distribution for combined predictor

In each figure, the distance (in branch instructions) between mispredicted branches is plotted, as well as the geometric distribution of mispredictions one would expect if there was no clumping at all.[2] The figures indeed show that mispredicted branches tend to come in clusters. For each of the predictors, 39% of mispredicted branches are within 3 branches of the previous mispredicted branch. However, the figures also show that as the quality of the branch predictor increases, the distribution comes closer and closer to the geometric. The difference between predictors is slight, but it is noticeable and may indicate that as branch predictors continue to increase in accuracy, this problem may become less and less onerous.

Clustering increases the likelihood of the following two scenarios, both of which diminish the gains from RDPE. First, a mispredicted branch can be covered by a preceding mispredicted branch. If one mispredicted branch closely follows another, then at least one of the two branches will not benefit from forking. Forking both branches would result in more then two threads of execution. Second, a mispredicted branch can be covered by a preceding branch in the low confidence set. Since the low confidence branch is forked, any closely following mispredicted branch cannot be forked (at least, not immediately).

[2]*The distance between mispredicted branches is the number of branches that separate them. If a mispredicted branch is immediately followed by a second, the distance is one.

5.2. Varying Branch Misprediction Penalty

There is a trend toward deeper processor pipelining; for example, the latest generation DEC processor, the 21264 [Kell96], and the latest Intel processor, the Pentium Pro [Gwen95], are both more deeply pipelined than their predecessors. As levels of processor pipelining increases, it is likely that the cycles lost for each mispredicted branch will also increase. Wider instruction fetching, register renaming and larger register files will also contribute to higher branch misprediction penalties when the instruction fetch pipeline must be refilled.

Intuitively, as branch misprediction penalties become worse, RDPE should become more effective. On the other hand, as the pipeline depth is increased, more branches overlap in execution which increases the effect of clustering (counteracting the increase in RDPE effectiveness.) This is illustrated in Fig. 6 where the performance of RDPE on a range of branch misprediction penalties is presented. This figure shows that RDPE performs quite well on compress and go, the two benchmarks with the worst branch misprediction rates. On some of the other benchmarks, the improvement is much more moderate. However, in all cases performance improvement is indicated.

6. Further Research

The greatest hardware limitation in RDPE is the need to fetch down two paths of execution simultaneously. However, the increased instruction fetch bandwidth of trace-cache based

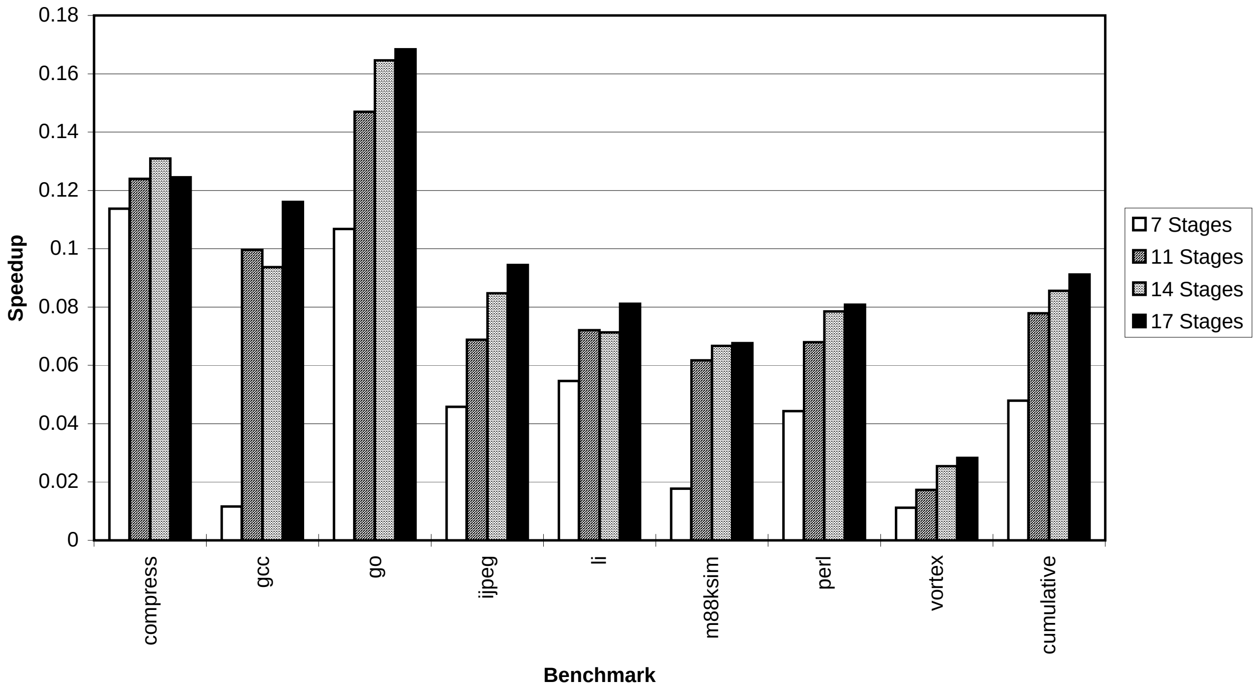

Figure 6: Percentage Improvement in Performance for Varying Pipeline Lengths

processors may allow RDPE to improve performance without having to fetch instructions from both paths every cycle. It may also be possible to stuff pieces of the alternate path onto the unused end of the trace line. These types of questions merit further investigation.

Branch mispredict clustering has been shown to be an impediment to achieving higher performance from RDPE. Modifying the behavior of the processor when two paths are being executed and another low-confidence branch prediction is encountered can reduce the negative effects of clustering. As described earlier, the forking policy determines which of the possible alternate paths the processor follows. If a processor is only executing a single path, and a low-confidence prediction is made, the choice is obvious; the branch is forked and the other path after the branch is followed. However, if two paths are already being followed, then a choice must be made. The new branch can be ignored; the alternate path from the previous branch can be truncated, allowing a fork on the new path; or the new fork can be delayed until the previous fork completes. These new options help with clustering, because it allows more branches to be forked, even though their life-times overlap. The whole question of clustering and how best can it be dealt with needs further study as well.

7. Conclusions

Restricted Dual Path Execution provides a way of further reducing branch misprediction penalties by accepting that there will be mispredictions, and then dealing with the situation by executing instructions on both branch paths when there is a relatively high likelihood that the prediction will be wrong.

This study indicates that RDPE can potentially reduce branch misprediction penalties and provide an increase in overall performance of approximately 7%. An improvement of this size is certainly worthwhile, given the relatively minor impact this approach should have on the size and complexity of the processor. It is also important to keep in mind that the branch predictors used in current systems are not a small portion of the chip area. While DPE requires extra resource to reduce the number of missed branches, it should be compared to the resource requirements of larger history tables required to achieve the same benefit (when possible).

In order to see how well this approach will apply to future designs, a set of experiments were run varying the number of pipeline stages. This was done because the increased latency of resolving low confidence branches will reduce the effectiveness of the DPE scheme by restricting the application of DPE - however, as the number of pipeline stages increases, the penalty paid for a misprediction climbs as well. We have shown that as issue widths widen and pipeline lengths increase, RDPE does in fact provide an increase in performance which ranges from quite small (< 2%) to substantial (> 15%).

A possible problem that we have identified is that improvements are limited somewhat by the statistical properties of conditional branches. Specifically, clustering of mispredicted branches and low confidence branches reduce the fraction of branch mispredictions that RDPE can capture. A partial solution may be to consider additional forked paths beyond two. In addition, it appears that better branch predictors are slightly better at reducing the clumping - if predictors continue to improve in accuracy as they have in the recent past, this problem may disappear.

We have also observed that an implementation of RDPE must have careful attention paid to supporting sufficient instruction fetch bandwidth. One straightforward approach is to duplicate the instruction fetch unit, although other approaches could be used. In any implementation, register renaming naturally extends to support multiple instruction paths. Some minor complications occur when squashing instructions after a forked branch has been resolved, however.

8. References

[BuAu97] D. Burger and T. M. Austin, ‘‘The SimpleScalar Tool Set, Version 2.0’’, Computer Sciences Department Technical Report #1342, University of Wisconsin, Madison, Madison, WI (June 1997).

[CoFP79] W. D. Connors, J. Florkowski and S. K. Patton, ‘‘The IBM 3033: An Inside Look’’, *Datamation*(May 1979), pp. 198-218.

[Gwen94] L. Gwennap, ‘‘MIPS R10000 Uses Decoupled Architecture’’, Microprocessor Report (October 1994).

[Gwen95] L. Gwennap, ‘‘Intel’s P6 Uses Decoupled Superscalar Design’’, Microprocessor Report (February 1995).

[JaRS96] E. Jacobsen, E. Rotenberg and J. E. Smith, ‘‘Assigning Confidence to Conditional Branch Predictions’’, *Proceedings of the 29th Annual International Symposium on Microarchitecture*, Paris, France (December 2-4, 1996), pp. 142-151.

[Kell96] J. Keller, ‘‘The 21264: A Superscalar Alpha Processor with Out-of-Order Execution’’, Presented at Microprocessor Forum (October 1996).

[LiTy96] K. Lick and G. Tyson, ‘‘Hybrid Branch Prediction Using Limited Dual Path Execution’’, Department of Computer Science Technical Report UCR-CS-96-7, University of California, Riverside, Riverside, Ca (July 1996).

[McFa93] S. McFarling, ‘‘Combining Branch Predictors’’, *Digital Western Research Laboratory Technical Note TN-36*(June 1993).

[RoBS96] E. Rotenberg, S. Bennett and J. E. Smith, ‘‘Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching’’, *Proceedings of the 29th Annual International Symposium on Microarchitecture*, Paris, France (December 2-4, 1996), pp. 24-34.

[Tyso94] G. Tyson, ‘‘The Effects of Predicated Execution on Branch Prediction’’, *Proceedings of the 27th Annual International Symposium on Microarchitecture*, San Jose, Ca. (November 30 - December 2, 1994), pp. 196-206.

[UhS95] A. K. Uht and V. Sindagi, ‘‘Disjoint Eager Execution: An Optimal Form of Speculative Execution’’, *Proceedings of the 28th Annual International Symposium on Microarchitecture*, Ann Arbor, Michigan (November 29 - December 1, 1995), pp. 313-325.

[Wall93] D. Wall, ‘‘Limits of Instruction-Level Parallelism’’, *Digital Western Research Laboratory Research Report*, no. 93/6 (November 1993).

[WaUh90] S. S. H. Wang and A. K. Uht, ‘‘Ideograph/Ideogram: A Framework/Architecture for Eager Execution’’, *Proceedings of the 23rd Annual Symposium and Workshop on Microprogramming and Microarchitectures*, Orlando, Florida (November 27-29, 1990), pp. 125-134.

[WeSm94] S. Weiss and J. Smith, *POWER and PowerPC*, Morgan Kaufmann Publishers, Inc., (1994).