The Effects on Branch Prediction when Utilizing Control Independence

Chris J. Michael, David M. Koppelman
Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, Louisiana, USA
cmichael@cct.lsu.edu, koppel@ece.lsu.edu

***Abstract*—Though current general-purpose processors have several small CPU cores as opposed to a single more complex core, many algorithms and applications are inherently sequential and so hard to explicitly parallelize. Cores designed to handle these problems may exhibit deeper pipelines and wider fetch widths to exploit instruction-level parallelism via out-of-order execution. As these parameters increase, so does the amount of instructions fetched along an incorrect path when a branch is mispredicted. Some instructions are fetched regardless of the direction of a branch. In current conventional CPUs, these instructions are always squashed upon branch misprediction and are fetched again shortly thereafter. Recent research efforts explore lessening the effect of branch mispredictions by retaining these instructions when squashing or fetching them in advance when encountering a branch that is difficult to predict. Though these control independent processors are meant to lessen the damage of misprediction, an inherent side-effect of fetching out of order, branch weakening, reduces realized speedup and is in part responsible for lowering potential speedup. This study formally defines and works towards identifying the causes of branch weakening. The overall goal of the research is to determine how much weakening is avoidable and develop techniques to help reduce weakening in control independent processors.**

***Keywords*-computer architecture; pipeline processing; microprocessors**

I. Introduction

Though current general-purpose processors have several small CPU cores as opposed to a single more complex core, often all but one of these cores sits idle because many algorithms and applications are inherently sequential and thus hard to explicitly parallelize. A common research approach involves designing CPUs heterogeneously with many different types of cores, some of which may handle these sequential codes by aggressively exploiting instruction-level parallelism (ILP). These cores will most likely have longer pipelines and wider fetch widths to exploit ILP via out-of-order execution. As these parameters increase, so does the number of instructions fetched along an incorrect path when a branch is mispredicted. Some instructions are fetched regardless of the direction of the branch. These are referred to as *control independent* (CI) instructions. In current conventional CPUs, these instructions are always squashed upon a branch misprediction and are fetched again shortly thereafter. Recent research efforts explore lessening the effect of branch mispredictions by retaining CI instructions when squashing [1], [2] or fetching these instructions in advance when encountering a branch that is difficult to predict. Though these control independence processors (CIPs) are meant to lessen the damage of misprediction, an inherent side-effect of fetching out of order, *branch weakening*, reduces realized speedup and is in part responsible for lowering the potential of speedup. Branch weakening is a property of many if not all proposed CIPs. The goal of the research is to formally define and identify the causes of branch weakening, determine how much weakening is avoidable, and offer techniques to help reduce weakening and its impact.

II. Prior Work

In the Transparent Control Independence study by Al-Zawawi et al. [1], a complex perceptron predictor [3] is used to obtain worthwhile results in branch prediction accuracy. Even with this predictor, 10 of 14 benchmarks suffered weakening. The worst case caused the number of mispredicts to rise by about 20%. It is mentioned that the decrease in prediction accuracy is due to gaps in the global history from the CD region. It is also mentioned that the reason the perceptron predictor is chosen is because it is less affected by these gaps when compared to other common predictors.

Perhaps the most attention to weakening in recently proposed mechanisms is given in the Ginger implementation study by Hilton and Roth [2]. In fact, it is mentioned explicitly that control independence interferes with conventional branch predictors due to weakening: “It would be counter-productive if Ginger induced more mispredictions than it tolerated.”[2] The implementation attempts to resolve weakening using two separate predictor states. The first of these excludes any CD history outcomes from its global history. In doing so, prediction accuracy is preserved for branches not correlated to CD outcomes. To accommodate for cases where branches may be correlated to these outcomes, a second predictor state is added that uses contiguous global history. A chooser is used to extract the best prediction of the two states. It is said in the study that the table used in the latter predictor can be small because the number of branches that correlate to CD data is small. Overall results are given to support this claim but there is no deeper look

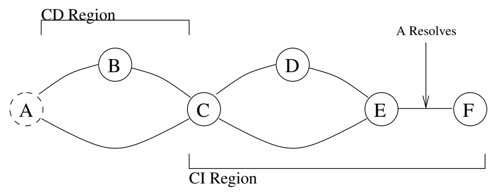

Figure 1. Example Control Flow of Execution

into weakening. It is shown that this scheme only improves performance slightly as opposed to using a predictor which does not contain CD information in its global history register (GHR). In some cases, performance may lower when using the dual scheme. One interesting result shows that using a scheme that excludes CD data from history can actually perform better than the perfect in-order case since there are more useful outcomes that would otherwise not appear in the global history.

III. Control Independence and Branch Weakening

Figure 1 shows a sample of execution control flow with each node representing a branch. Assume the system predictor correlates on recent branch outcomes. Branch `A` is difficult to predict so a CIP may *protect* it, exploiting control independence so that the branch's misprediction has less impact on performance. All other branches shown in the figure are predicted with nearly 100% accuracy in the conventional system. The CI and CD regions are marked in the figure. Branch `C` is the first branch independent of `A` and so is called the *reconvergence point* of `A`. Also marked on the graph is where the system is fetching when `A` resolves. Assume `C` is very highly correlated to `A`. The two common program paths in this area of execution are `ABCDEF` and `ACEF`. A CIP may fetch similarly to a conventional system until `A` resolves – at that point only the CD region will be squashed and re-fetched as opposed to a conventional system where all subsequent instructions are squashed. Fetching then continues where it left off immediately before `A` resolved.

Weakening due to the increase in warm-up time when exploiting CI, or *insulated weakening*, may occur when the system predictor uses global history data to make predictions. The branch `E` (which only sees two paths in the conventional system at predict time) sees an additional two paths in the CIP, `ABCE` and `ACDE`. This is because `E` is fetched before `A` is resolved mispredicted but is not flushed. These two additional paths require predictor warm-up and the additional warm-up mispredicts are considered insulated weakening. Insulated weakening can be lessened by using special history update techniques that prevent the speculated CIP paths from being observed in the system state.

Absentee weakening is caused by correlation data not being available at a post-reconvergent branch's predict time.

Table I
Selected Base Snipper Configuration Parameters

Core	8-way superscalar 512-entry ROB 20 cycle decode 8 integer, 4 floating pointALUs
Memory	64kiB 8-way 32B line 10 cycle lat L1 ICache 64kiB 4-way 64B line 1 cycle lat L1 DCache 8MiB 8-way 64B line 19 cycle lat L2 DCache 150 cycle minimum memory access latency
Predictor	16-bit GHR 64kiB entry 2 bits per entry BHT/PHT

Similarly to insulated weakening, absentee weakening may occur when a system predictor uses history data to make predictions. Unlike insulated weakening, however, data crucial to make a correct prediction is missing. In the figure, say branch `D` is highly correlated to `B`. In a conventional system, the outcome of `B` is always available at `D`'s predict time. Now consider a CIP where `D` is seen on the new speculated path `ACDE`. The history data for `B` is not observable in this path and therefore `D` has nothing to correlate with, causing absentee weakening. It is difficult to eliminate absentee weakening when exploiting control independence, so any CI mechanism may be turned off where it occurs to minimize the damage it causes.

When employing any CI technique, there may be longer predictor update times due to instructions' speculative execution when waiting for a covered branch to resolve. This causes predictor tables to update later than they normally would on a conventional system. The increased *update lag* causes the third type of weakening, *delayed-update weakening*. This type of weakening may be avoided by updating predictor tables earlier than commit time; for example, the first time a branch resolves. However, this does not always alleviate the damage. Dynamic branch instances in CIPs may re-execute and re-resolve, changing their branch direction multiple times before committing. This phenomenon, called *vacillation*, may cause predictors to update incorrectly.

IV. Methodology

Snipper is a CIP that retains CI instructions when squashing covered branches [4]. It uses a hardware method of detecting and analyzing CD regions and only protects a branch when it detects potential for improving performance. Once fetched, instructions remain in the scheduler until they commit and are re-executed when necessary. Registers may need to be re-mapped or preserved in Snipper since instructions may be fetched out of order. This is done using special injected instructions that correct register mappings as needed. Branches may be protected and unprotected dynamically in execution. Snipper has been configured as shown in Table I. By default, CD global history is used to predict CI branches, whether or not the data is correct. This method was chosen because it outperformed the method

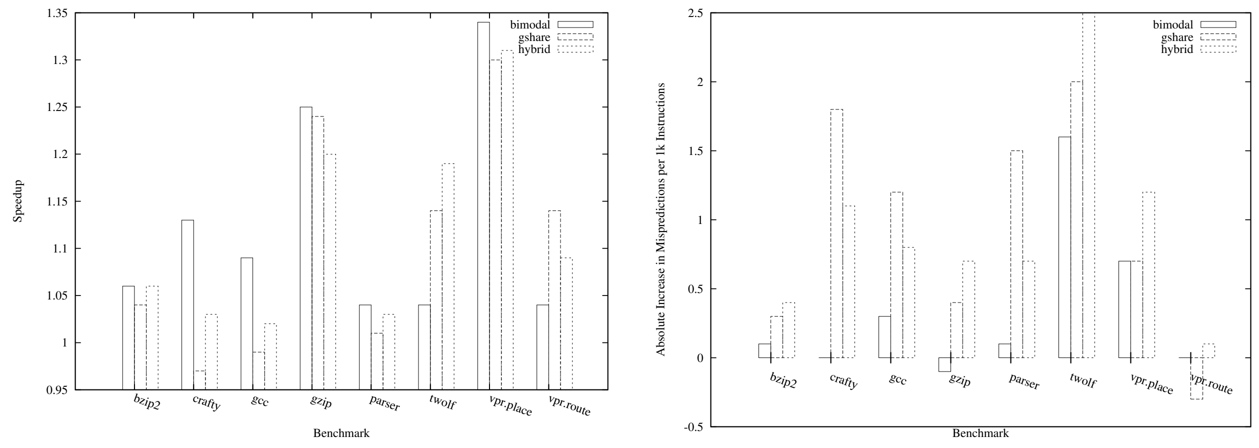

Figure 2. Snipper speedup and weakening for several benchmarks

of leaving the CD outcomes out of the global history. The benchmarks studied are those very active in exploiting CI using Snipper of the SPEC CPU 2000 set [5]. Figure 2 shows results for speedup and weakening in a Snipper system using bimodal, GShare [6], and bimodal/GShare hybrid predictors. As can be seen, weakening is certainly not negligible for most of the benchmarks.

V. Approach

If weakened branches can be classified by their cause, then the amount of reparable weakening may be measured. The first plot of Figure 2 shows that substantial speedup is possible despite weakening, with some benchmarks enjoying 20% or more speedup. But the second plot of the figure shows a substantial amount of weakening, increasing the mispredict rate by 1 misprediction per 1k instructions on average across the set. Since the studied system may fetch 160-512 instructions down a wrong path of a mispredicted branch, this represents over a 10% potential loss of useful fetch. Methods to measure the different causes of weakening on a range of predictors will soon be developed, but for now it is possible to measure delayed-update weakening since it is the only weakening that CIPs using a bimodal predictor are vulnerable to. This is because the bimodal predictor does not use any global history information. As shown in Figure 2, several benchmarks are significantly affected by this type of weakening.

Insulated weakening can be measured by collecting data regarding *correlated paths* of branches. A path of a branch is the state of the global history register at a dynamic instance of the branch. A correlated path is a path that is highly predictable. Each correlated path takes some time to warm-up. For example, a path with biased behavior may mispredict twice to set the predictor's saturating counter towards its bias. By measuring the increase in correlated paths from a conventional system to a CIP, one can measure insulated weakening. This type of weakening may be reduced by not adding any CD data to the global history state. There are a couple of reasons why this will not eliminate insulated weakening completely. First, some CIPs, including Snipper, dynamically protect and un-protect branches and it can not be known before hand where the CD region is. Second, when excluding CD data, the global history may contain outcomes farther away from the branch being predicted – these outcomes will not be observed in a conventional system and may hinder prediction accuracy. Aside from these problems, removing CD data from the GHR may needlessly induce more weakening. For instance, Snipper performs better when CD data is inserted in to the GHR and predictor tables are updated regardless of any protected branch being unresolved. The reasons for this behavior are currently being researched.

Absentee weakening is perhaps the most difficult to measure. To be weakened in this manner, a branch must be correlated to data CD of some protected branch. Though measurement may be difficult, the solution is simple. If exploiting CI will induce absentee weakening, don't do it. If it turns out that the majority of weakened branches are absentee, then control independence is unreasonable since relieving one misprediction is causing another.

In targeting delayed-update weakening, branches must be updated earlier than commit time. This is not a trivial problem due to branch vacillation. Through comparison of several techniques that allow predictors to update before commit time, a *change* method is found to perform best. When using this technique, a dynamic branch instance updates its predictor entry on its first resolve and additionally any other resolve which differs from its last. This helps ease the predictor of incorrect updates due to vacillation while allowing predictor tables to be updated earlier than commit time.

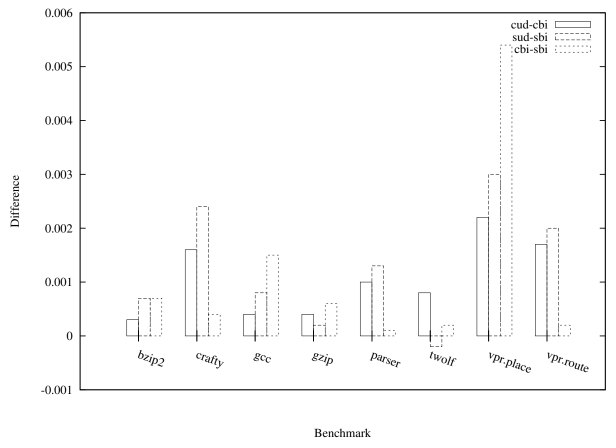

Figure 3. Differences in branch prediction ratio for several configurations

VI. Preliminary Results

In order to address delayed-update weakening, several special-purpose predictors have been developed which attempt to choose the best time to perform an update. Branches are either updated earlier than commit time to reduce delayed-update weakening or updated at commit time to prevent incorrect update from vacillation. These predictors, called *flexible update predictors*, can be used with any branch predictor.

Of all flexible update predictors developed, the best results are obtained by using the *update chooser*. This predictor dynamically chooses what prediction time is best. There are two core predictor states, identical in every way except that one always updates at change and the other always updates at commit. The core predictor is set to be either bimodal, GShare, or bimodal/GShare hybrid. A 2-bit chooser table similar to the hybrid chooser described in [6] is used to reference the most accurate predictor entry among the two states. The predictor will update entries at commit by default and the choice table entry is updated at branch commit time. This predictor is most direct of all that have been researched but is also most demanding since it requires an additional table look-up for the chooser and is two times the size of the regular predictor.

Call a conventional system using a bimodal predictor that updates at commit `cbi` and the same system using a bimodal update chooser predictor `cud`. Call a Snipper-enabled system using a bimodal predictor that updates at commit `sbi` and one using a bimodal update chooser `sud`. The systems are all configured as shown in Table I. Figure 3 shows differences in branch prediction accuracy for relevant configuration pairs. The first result, `cud-cbi`, shows the improvement of branch prediction ratio when using an update chooser on a conventional system while the second, `sud-sbi`, shows improvement for Snipper. The final result, `cbi-sbi`, shows the differences in branch prediction ratio due to weakening using a bimodal predictor.

The first result in the figure shows that even `cbi` benefits from selectively updating earlier than commit time. The benchmarks benefiting most have three significant common behaviors. First, they usually exhibit a large amount of biased or *bistable* branches. A bistable branch is one which exhibits long runs of the same outcome. The second common behavior is that branches generally have a long predict-to-commit time. The last is that dynamic instances of a branch will often overlap in execution, exhibiting a "tight-loop" behavior. In studying branch behaviors it has been found that it is these types of branches that benefit from updating earlier than commit time.

The second result shows how Snipper benefits from the update chooser. Note that in 6 of the 8 shown cases the benefit is significantly larger than that of the conventional system. This is expected since activating Snipper increases branch's predict-to-commit time by about 15 cycles on average from a conventional system. When compared with the last result, the alleviation of delayed-update weakening can be estimated. For 4 of the benchmarks, the improvement of using the update chooser with Snipper surpasses or equalizes the branch prediction ratio to that of a conventional system. In the cases of crafty, parser, and vpr.route it may be appropriate to assume that most if not all of the weakening has been overcome since the benefit of the Snipper system far surpasses the sum of the conventional benefit and the weakening. The benchmarks where benefit does not come near the weakening have been examined. The main cause of this irreparable weakening is either high vacillation rates or erratic branch behaviors.

References

[1] A. S. Al-Zawawi, V. K. Reddy, E. Rotenberg, and H. Akkary, "Transparent control independence (tci)," in *ISCA*, 2007, pp. 448–459.

[2] A. D. Hilton and A. Roth, "Ginger: control independence using tag rewriting," in *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2007, pp. 436–447.

[3] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2001, p. 197.

[4] "Rsiml," http://www.ece.lsu.edu/koppel/work/proc.html.

[5] J. L. Henning, "Spec cpu2000: Measuring cpu performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28–35, 2000.

[6] S. McFarling, "Combining branch predictors," *WRL TN-36*, 1993.