CACHED TWO-LEVEL ADAPTIVE BRANCH PREDICTORS WITH MULTIPLE STAGES

EGAN, C., STEVEN, G. B., VINTAN, L.

University of Hertfordshire, *University "Lucian Braga" of Sibiu*

Hatfield, Hertfordshire, U.K. AL10 9AB *Sibiu-2400, Romania*

email: G.B.Steven@herts.ac.uk vintan@cs.sibiu.ro

Abstract

During the last decade, the accuracy of branch predictors was significantly improved by the development of Two-Level Adaptive Branch Predictors. However, although these predictors deliver very high prediction rates, they have several disadvantages. Firstly, the size of the second-level Pattern History Table (PHT) increases exponentially as a function of history register length and therefore becomes very costly if a large amount of branch history is exploited. Secondly, many of the prediction counters in the PHT are never used. Thirdly, predictions are frequently generated from non-initialised counters. Finally, several branches may update the same counter, resulting in interference between branch predictions. In this paper, we quantify the performance of a novel family of multi-stage Two-Level Adaptive Predictors. In each two-level predictor, the PHT is replaced by a Prediction Cache. Unlike a PHT, a Prediction Cache saves only relevant branch prediction information. Furthermore, predictions are never based on uninitialised entries and interference between branches is eliminated. In the case of a Prediction Cache miss in the first stage, our two-stage predictors uses a default two-bit prediction counter stored in a second stage. We demonstrate that a two-stage Cached Predictor is more accurate than a conventional two-level predictor and quantify the crucial contribution made by the second prediction stage in achieving this high accuracy. We then extend our Cached Predictor by adding a third stage and demonstrate that a Three-Stage Cached Predictor further improves the accuracy of cached predictors.

Keywords

Two-Level Adaptive Branch Predictors, Cached Correlated Branch Predictors, Prediction Cache, Multi-Stage Branch Predictors.

INTRODUCTION

The advent of superscalar processors has given renewed impetus to branch prediction research. On a scalar processor, an incorrect branch prediction costs only a small number of processor cycles and only one or two instructions are lost.

In contrast, in a superscalar processor many cycles may elapse before a mispredicted branch instruction is finally resolved. Furthermore, each cycle lost now represents multiple lost instructions. As a result branch mispredictions are far more costly on a superscalar processor.

This renewed interest in branch prediction led to a dramatic breakthrough in the 1990s with the development of Two-Level Adaptive Branch Predictors by Yale Patt's group [Yeh91] and by Pan, So and Rahmeh [Pan92]. More recently two-level branch predictors have been implemented in several commercial microprocessors [Int00, Kes99]. However, although high prediction rates are achieved with two-level adaptive predictors, this success is obtained by providing very large arrays of prediction counters or PHTs (Pattern History Tables). Since the size of the PHT increases exponentially as a function of history register length, the cost of the PHT can become excessive, and it is difficult to exploit a large amount of branch history effectively. Two-level Adaptive Branch Predictors have two other disadvantages. Firstly, in most practical implementations each prediction counter is shared between several branches. There is therefore interference or aliasing between branch predictions. Secondly, large arrays of prediction counters require extensive initial training. Furthermore, the amount of training required increases as additional branch history is exploited, further limiting the amount of branch history that can be exploited.

We have developed [Ega00] a family of Two-Level Branch Predictors that addresses the three problems of conventional two-level predictors: cost, interference and initial training. We have called these novel predictors Cached Correlated Branch Predictors. By replacing the second level PHT with a cache, we significantly reduce the cost. At the same time, our predictor outperforms the traditional implementations. For equal cost models, this performance advantage is particularly significant. These advantages are achieved for three reasons. Firstly, our cached predictor only holds those prediction counters that are actually used. Secondly, interference between branches is eliminated; each branch prediction is determined solely by historical information related to the branch being predicted. Thirdly, a simple default prediction mechanism is included that is initialised after a single occurrence of each branch. This avoids the high number of initial mispredictions sustained during the warm-up phase of conventional two-level predictors and minimises the impact of misses in the Prediction Cache.

In an earlier feasibility study [Ste00] we presented a Cached Correlated Branch Predictor that used a fully associative Prediction Cache. Although the concept of a cached PHT was successfully demonstrated, a fully associative cache would be too costly in practice. In contrast, all the Cached Correlated Branch Predictors, presented in this paper, use a set-associative Prediction Cache that is indexed by hashing the PC with the history register. We also quantify the crucial role played by a second prediction stage in our cached predictor. We then extend cached prediction techniques to three-stages for the first time.

TWO-LEVEL BANCH PREDICTION

Recent research on branch prediction has focused almost exclusively on Two-Level Adaptive Branch Predictors, which are usually classified using a system proposed by Yeh and Patt [Yeh92, Yeh93]. The six most common configurations are GAg, GAp, GAs, PAg, PAp and PAs. The first letter specifies the first-level mechanism and the last letter the second level, while the "A" emphasises the adaptive or dynamic nature of the predictor. GAg, GAp and GAs rely on global branch history while PAg, PAp and PAs rely on local branch history.

GAg uses a single global history register, that records the outcome of the last *k* branches encountered, and a single global PHT containing an array of two-bit prediction counters. To generate a prediction, the *k* bit pattern in the first-level global history register is used to index the array of prediction counters in the second level PHT. Each branch prediction therefore seeks to exploit correlation between the next branch outcome and the outcome of the *k* most recently executed branches. The prediction counter in the PHT and the global history register are updated as soon as the branch is resolved. Finally, a separate BTC is still required to provide branch target addresses.

Unfortunately, since all the branches in a GAg predictor share a common set of prediction counters, the outcome of one branch can affect the prediction of all other branches. Although this branch interference limits the performance, the prediction accuracy improves as the history register length is increased. At the same time, the number of counters in the PHT also increases, which in turn increases both the number of initial mispredictions and the cost of the PHT.

Eventually, the increased number of initial mispredictions negates the benefit of additional branch history and the prediction accuracy stops improving.

Several researchers have attempted to reduce interference in the PHT. The Gshare Predictor [McF93, Cha94], for example, hashes the PC and history register bits before accessing the PHT, in an attempt to spread accesses more evenly throughout the PHT. Alternatively, the Bimodal Predictor [Lee97] uses twin PHT arrays to decrease destructive interference between branch predictions and to maximise positive interference. Finally, the Agree Predictor [Spr97] also attempts to maximise positive interference.

GAp was first proposed by Pan et al [Pan92] and called Correlated Branch Prediction. Like GAg, GAp uses a single history register to record the outcome of the last *k* branches executed. However, to reduce the interference between different branches, a separate per-address PHT is provided for each branch. Conceptually in GAp, the PC and the history register are used to index into an array of PHTs. Although this ideal model eliminates interference between branches, it leads to an exceptionally large PHT array. For example, with a 30-bit PC and 12-bit history register, 2^{42} counters are required. In practice, to limit the size of the predictor, only a limited number of PHT arrays is provided; each PHT is therefore shared by a group of PCs with the same least significant address bits. Since a separate set of PHT counters is provided for each set of branch addresses, this configuration is classified as GAs. However, while the size of the PHT array is significantly reduced, branch interference is now reintroduced. As in the case of GAg, a separate BTC is provided to furnish branch target addresses in both the GAp and GAs configurations.

The Two-Level Adaptive Branch Prediction mechanism originally proposed by Yeh and Patt in 1991 [Yeh91] was later classified as PAg. PAg uses a separate local history register for each branch, or a Per-address history register, and a single shared global PHT. Each branch prediction is therefore based entirely on the history of the branch being predicted. The local history registers can be integrated into the BTC by adding a history register field to each entry. Since all branches share a single PHT, PAg is also characterised by interference between different branches. Interference can be reduced by providing multiple PHTs. If we retain the Per-Address Branch History Table and provide a separate PHT for each address or a Per-Address PHT, we have the PAp configuration. As in the

case of GAp, the size of the PHT array is excessive, and the initial training problem is exacerbated. A separate PHT is therefore usually provided for sets of branches, giving rise to the PAs configuration.

We have emphasised that most branch prediction research is based on Two-Level Adaptive Branch Predictors. Yet, branch prediction is a specific example of a general Time Series Prediction problem that occurs in many diverse fields of science. It is therefore surprising that there has not been more cross-fertilisation of ideas between different application areas. A notable exception is a paper by Mudge‘s group [Che96] that demonstrates that all Two-Level Adaptive Predictors implement special cases of the Prediction by Partial Matching [PPM] algorithm that is widely used in data compression. Mudge uses the PPM algorithm to compute a theoretical upper bound on the accuracy of branch prediction. Another exception is a recent attempt to use Neural Networks for dynamic branch prediction [Ste01].

TWO STAGE CACHED CORRELATED PREDICTION

The high cost of Two-level Adaptive Branch Predictors is a direct result of the size of the second level PHTs which increase exponentially in size as a function of History Register length. In a Cached Correlated Predictor [Ega00, Ste00], the second-level table is therefore replaced with a Prediction Cache, while the first level is unchanged. Unlike PHTs in conventional two-level predictors, the number of entries in a Prediction Cache is not a direct function of the History Register length. Instead, the size of the cache is determined by the number of prediction counters that are actually used. This number increases only slowly as a function of History Register length. Since the Prediction Cache only needs to store active prediction counters, most of the entries in a traditional PHT can be discarded. However, to implement caching, a tag field must be added to each entry and the size of the tag field increases linearly as a function of History Register length. A Cached Correlated Branch Predictor will therefore only be cost effective as long as the cost of the redundant counters removed from the PHT exceeds the cost of the added tags. Two Cached Correlated Branch Predictors are presented in this section. The first predictor employs a global history register, while the second employs multiple local or per-address history registers.

Global Cached Correlated Predictor

Figure 1 shows a four-way set-associative Global Cached Correlated Branch Predictor. Each entry in the Prediction Cache consists of a PC tag, a history register tag, a two-bit prediction counter, a valid bit and a LRU (Least Recently Used) field. A four-way set-associative BTC is also provided to furnish the branch target address. Each BTC entry is augmented with a two-bit default prediction counter and consists of a branch target address, a branch address tag, the two-bit prediction counter, a valid bit and a LRU field. The BTC is accessed using the least significant bits of the PC, while the Prediction Cache index is obtained by hashing the PC with the global history register bits. As long as there is a miss in the BTC, the predictor has no previous record of the branch and defaults to predict not taken. Whenever there is a BTC hit a prediction is attempted. If there is also a hit in the Prediction Cache, the corresponding two-bit counter from the Prediction Cache entry is used to generate the prediction. In this case the prediction is based on the past behaviour of the branch with the current history register pattern. If, however, there is a miss in the Prediction Cache, the prediction is based on the default prediction counter held in the BTC and is therefore based on the overall past behaviour of the branch. Once the branch outcome is known, the relevant saturating counters are updated in both the Prediction Cache and the BTC. In the case of misses in either cache, new entries are added using an LRU replacement algorithm. Finally, the global history register is updated.

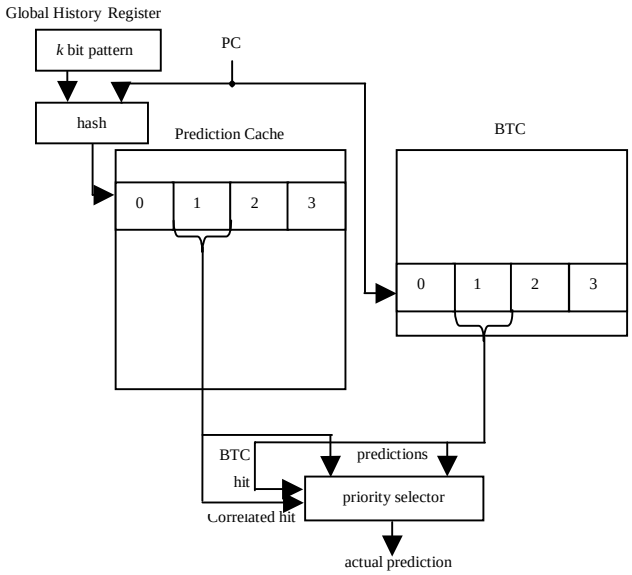

Figure 1: A Global Cached Correlated Branch Predictor.

Adding a default prediction counter to each BTC entry has several advantages. Firstly, the default predictor is initialised after only one execution of the branch. In contrast, with a *k* bit history register, up to 2^k Prediction Cache entries must be initialised for each branch before the two-level predictor is fully trained. Adding a default predictor should therefore significantly reduce the number of initial mispredictions. Secondly, the default predictor minimises the impact of misses in the Prediction Cache.

The hashing function to access the Prediction Cache requires careful consideration. Both a BTC and an instruction cache are usually indexed by the least significant bits of the PC. However, this solution is completely unsatisfactory for a Prediction Cache. Consider, for example, a four-way set-associative cache. In the absence of collisions with other branches, each branch is restricted to only four entries. However, if *k* history register bits are used by the predictor, as many as 2^k cache entries may theoretically be required for each branch. Although most history register patterns will never occur, a PC indexed cache will clearly suffer from excessive collisions, even with modest history register lengths.

A second alternative is to use the history register to index the Prediction Cache. This solution also has disadvantages. Firstly, if only a small number of history register bits is used, only part of the Prediction Cache will be used. Secondly, when the number of history register bits exceeds the number of bits in the cache index, sufficient collisions occur to prevent the predictor from reaching its full potential.

We found that the most accurate predictions were obtained when the history register bits were XORed with the PC bits to form the Prediction Cache index. A single XOR followed by truncation was found to be non optimum. Instead, the following hashing algorithm was adopted. First, the PC was concatenated with the history register. Second, the resulting bit pattern was divided into groups that contained the same number of bits as the required index. Finally, all the groups were XORed to generate the Prediction Cache index.

Local Cached Correlated Predictor

The Local Cached Correlated Predictor (Figure 2) also replaces the PHT with a Prediction Cache. Since a history register is now required for every

branch, a local history register field is added to each BTC entry. As with the Global Cached Correlated Predictor, a prediction counter is included in each BTC entry. The BTC is accessed using the least significant bits of the PC. On a BTC hit, the history register associated with the PC is obtained along with a default prediction. The history register is then hashed with the PC and the resulting bit pattern is used to access the Prediction Cache. Whenever possible a prediction counter stored in the Prediction Cache is used to make a prediction. However, in the case of a Prediction Cache miss and a hit in the BTC, the prediction from the BTC is used.

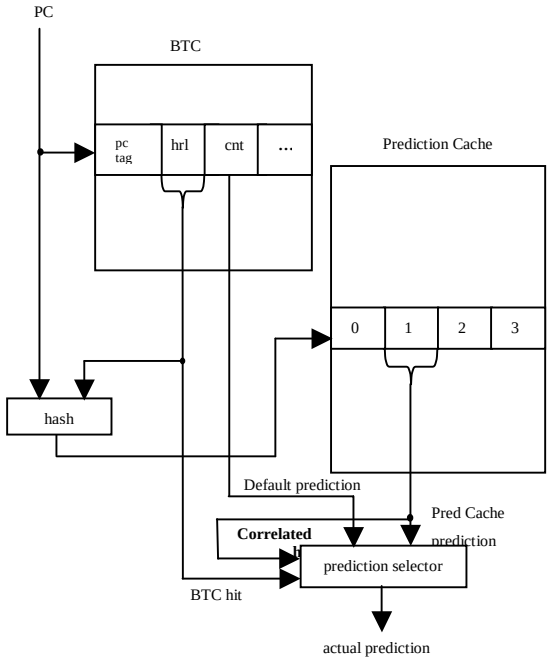

Figure 2: A Local Cached Correlated Branch Predictor.

Hybrid predictors [McF93, Cha95] also use two or more predictors to generate each prediction. A hybrid predictor chooses dynamically between two or more distinct predictors on the basis of each predictor's past success. In contrast, our priority prediction mechanism uses the Prediction Cache whenever possible, and only uses the prediction counter in the BTC when no other prediction is available.

TWO STAGE PREDICTOR PERFORMANCE

In this section, we quantify the performance of two-stage Cached Correlated Predictors. First we compare their performance with conventional two-level predictors. We then quantify the crucial contribution of the second stage. Our

simulations used a set of eight integer programs known collectively as the Stanford benchmarks. Since the programs are shorter than the SPEC benchmarks, each branch is executed fewer times. The branches are therefore more difficult to predict and the initial training problems are more acute. As a result, a classic BTC only achieves an average misprediction rate of 11.86% with the Stanford benchmarks. The benchmarks were compiled for the Hatfield Superscalar Architecture [Ste97], a high-performance multiple-instruction-issue architecture developed to exploit instruction-level parallelism through static instruction scheduling. The HSA instruction-level simulator was then used to generate instruction traces for the branch prediction simulations. All the predictors simulated use a four-way set-associative BTC with 1K entries; sufficient entries are always available to minimise BTC misses. A four-way set-associative organisation is also always used for the Prediction Cache.

Global Cached Predictors

For comparative purposes, we first simulated a GAg predictor, a GAs predictor with 16 sets and a GAp predictor. The best misprediction rates were achieved by the GAp predictor (Figure 2). The average misprediction rate initially falls steadily as a function of the history register length before flattening out at a misprediction rate of around 9.5%. The best misprediction rate of 9.23% is achieved with the 26 history register bits. In general, however, there is little benefit from increasing the history register length beyond 16 bits.

The average misprediction rates achieved with a Global Cached Correlated Predictor are also shown in Figure 2. The number of entries in the Prediction Cache is varied from 1K to 64K. Initially, the misprediction rate steadily improves as a function of history register length for all cache sizes. However, after history register lengths of 12 bits, the limited capacity of the 1K Prediction Cache prevents further improvement. In contrast, with larger Prediction Cache sizes, the prediction rate continues to improve until a history register length of 26 bits is reached. Not surprisingly, the larger the Prediction Cache the better the misprediction rates. The lowest misprediction rate of 5.99% is achieved with a 32K entry Prediction Cache and a 30-bit history register. This represents a 54% reduction over the best misprediction rate achieved by a conventional global Two-Level Adaptive Predictor.

The high performance of the Cached Predictor depends crucially on the provision of the two-stage mechanism. Without the default predictors in the BTC, Prediction Cache misses result in an excessive number of mispredictions. To quantify the contributions of the default prediction counters, we repeated our simulations with the BTC counters removed (Figure 3). The best misprediction rate achieved rose to 9.12%. Removing the second stage therefore degrades the prediction accuracy by 52%. As a result, the Prediction Cache performance is now only marginally better than a conventional Two-Level Adaptive Predictor and only 12 bits of history register information can be exploited. Even worse, as the history register length is increased beyond twelve bits, the prediction accuracy is degraded catastrophically.

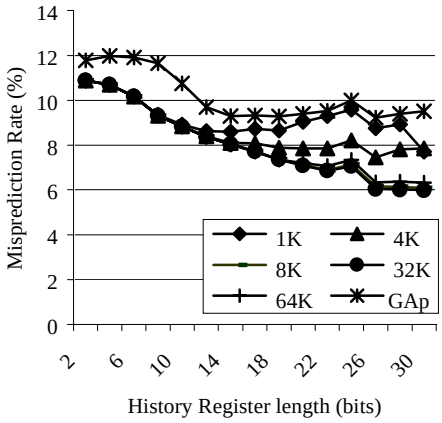

Figure. 2: Global Cached Correlated misprediction rates.

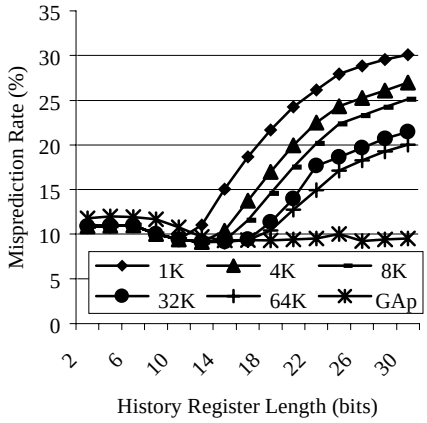

Figure. 3: Global Cached Correlated misprediction rates (no default predictor).

Local Cached Predictors

Again for comparative purposes, we first simulated conventional PAg, PAs and PAp predictors. Conventional local predictors achieve average misprediction rates of around 7.5%, significantly better than GAg/GAs predictors. The best conventional local performance of 7.35% is achieved with a PAp predictor and a 30-bit history register length (Figure 4). Local predictors are therefore able to benefit from longer history registers than their global counterparts.

The misprediction rates achieved by a Local Cached Correlated Predictor are also recorded in Figure 4. The number of entries in the Prediction Cache is varied between 1K and 64K. Initially the misprediction rate falls steadily as a function of history register length for all cache sizes. Then as more and more

predictions need to be cached, the larger caches deliver superior prediction rates. However, no significant benefit is derived from increasing the cache size beyond 8K. The best misprediction rate of 6.19% is achieved with a 64K cache and a 28-bit history register. This figure is slightly worse than the best global predictor, but represents a 19% improvement over the best PAg/PAp configuration.

In Figure 5, we record the impact of removing the default prediction stage from our Global Cached Correlated Predictors. Again, the impact is severe. The best misprediction rate rises to 8.21%, an increase of 33%. This figure is achieved with 12 history register bits and a 32K Prediction Cache. Overall, the performance is now worse than a conventional Two-Level Adaptive Predictor. We conclude that Local Cached Predictors are ineffective without a default prediction mechanism and are unable to exploit more than around 12 bits of branch history information.

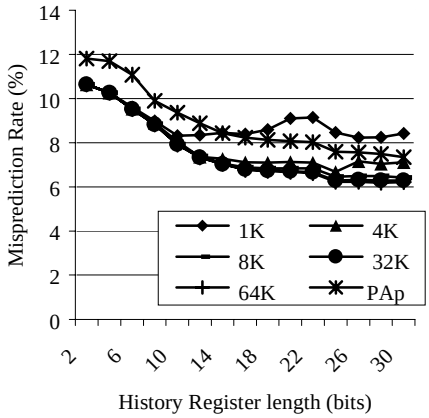

Figure. 4: Local Cached Correlated misprediction rates.

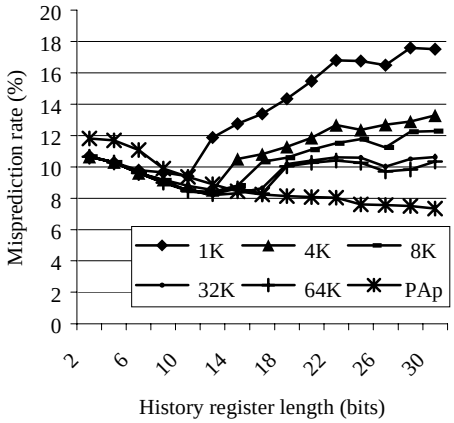

Figure. 5: Local Cached Correlated misprediction rates (no default predictor).

THREE-STAGE PREDICTOR

The simulation results in the previous section suggest that a Cached Predictor can deliver a higher prediction accuracy than a conventional Two-Level Adaptive Predictor. However, this superiority is crucially dependent on the provision of default prediction counters in the BTC. Default prediction counters improve performance for two reasons. Firstly, each counter is initialised after only a single execution of a branch. In contrast, a branch may have to be executed many times before a useful entry is made in the Prediction Cache. Furthermore, several entries must be initialised for each branch. Secondly, the Prediction Cache is of finite size and is therefore unable to retain all the relevant branch

prediction information. In the absence of a default predictor, a high proportion of Prediction Cache misses will generate mispredictions.

Furthermore, the best misprediction rates were achieved with long history registers. For example, the best Global Cached Predictor achieved a misprediction rate of 5.99% with a 30-bit History register, while the best Local Cached Predictor achieved 6.19% with 28 bits. This is remarkable, since Prediction Caches using 30-bit history registers require considerable initiation. We therefore believed that there was scope for introducing a third prediction level of intermediate complexity. This third prediction stage would use fewer history register bits than the main Prediction Cache, but, unlike the BTC, would not throw away all the history register information.

These considerations led to the development of a Three-Stage Cached Predictor with the following stages: a Primary Prediction Cache with *k* history register bits, a Secondary Prediction Cache with *k/2* history register bits and a default BTC predictor. Our expectation was that the new Secondary Prediction Cache, with only half the number of history register bits, would be initialised more rapidly than the Primary Prediction Cache. It would therefore be able to generate more accurate predictions than the BTC when there were misses in the Primary Prediction Cache.

A Three-Stage Predictor can be viewed as a practical implementation of Prediction by Partial Matching [Che96]. Predictions are generated as follows. If there is a miss in the BTC, the predictor has no knowledge of the branch and defaults to predict-not-taken. However, whenever there is a BTC hit, a prediction is attempted on a strict priority basis. Whenever possible, the Primary Prediction Cache is used, then the Secondary Prediction Cache, and finally the BTC.

Three Stage Predictor Performance

We repeated our simulations using both Global and Local versions of our Three-Stage Cached Predictors. As before, the size of the Primary Prediction Cache was varied between 1K and 64K. The Secondary Prediction Cache was always half the size of the Primary Cache and used exactly half the number of history register bits. The results for the Global Three-Stage Predictors are summarised in Figure 6. As expected, the three-stage predictor consistently outperforms the simpler global two-stage predictor, particularly when a large

number of history register bits is involved. The best misprediction rate of 5.57% is achieved with a 32K Primary Prediction Cache and a 30-bit history register. This represents a 7.5% improvement over the best Two-Stage Global Predictor.

The results for the Local Three-Stage Cached Predictor are summarised in Figure 7. Again, the three-stage predictor consistently outperforms its two-stage counterparts. The best misprediction rate of 6.00% is achieved with a 64K Primary Prediction Cache and a 28-bit history register, an improvement over the best Local Two-Level Predictor of 3.2%.

Three-Stage Predictors therefore consistently recorded a small but significant improvement over their two-stage counterparts. Furthermore, this improvement was not necessarily achieved by increasing cost. For example, a Global Three-Stage Predictor with an 8K primary cache and a 4K secondary cache outperforms a Global Two-Stage Predictor with a 32K Prediction Cache.

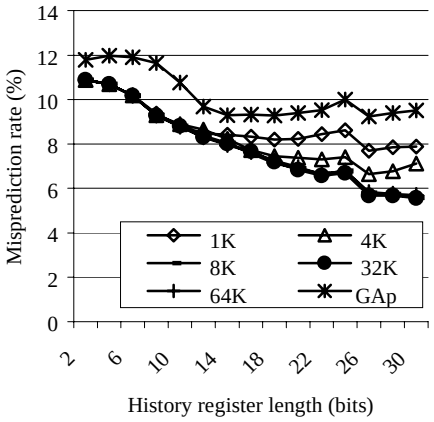

Figure. 6: Global Three Stage misprediction rates.

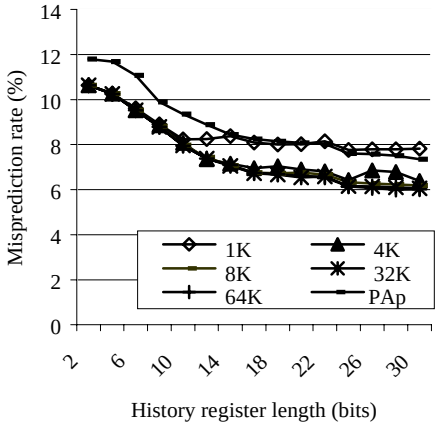

Figure. 7: Local Three Stage misprediction rates.

CONCLUSIONS

Our simulations show that a Cached Correlated Branch Predictor is significantly more accurate than a conventional Two-level Adaptive Predictor. In earlier work, we also demonstrated that cached predictors are more cost-effective than conventional predictors [Ega00, Ste00]. Our best global predictor is 54% better than the best GAs predictor and our best local predictor is 19% better than the best PAg/PAp predictor. We ascribe this higher accuracy to our more disciplined approach. Our predictions are always based on counters that have been trained using at least one previous encounter with the branch being

predicted. Furthermore, there is never any interference between branch predictions.

The higher accuracy depends crucially on the addition of default predictors in the BTC. Removing the default prediction counters degrades the performance of the best global predictors by 52% and the best local predictor by 33%. As history register lengths increase, two-level predictors require an increasing number of counter initialisations and therefore suffer an increasing numbers of initial mispredictions. In contrast, the default counter is initialised after only one execution of a branch, significantly reducing the number of initial mispredictions.

Even higher prediction accuracy was achieved with our Three-Stage Cached Predictors, which can be viewed as a practical implementation of Prediction by Partial Matching. The best three-stage predictor delivered a misprediction rate of 5.43%, a 35% improvement over the best conventional Two-Level Adaptive Predictor, and a 4.6% improvement over the best two-stage cached predictor.

A major advantage of Cached Correlated Branch Predictors is their ability to exploit correlations from a large number of history bits. In our two-stage Combined Cached Predictor, this advantage is exploited to combine local and global history information in a single predictor. This combined predictor delivered a misprediction rate of 5.68%, 29.4% better than the best conventional two-level predictor. Finally, a three-stage combined predictor delivered a misprediction rate of 5.42%, the lowest misprediction rate reported in this paper.

REFERENCES

[Cha94] CHANG, P.; HAO, E.; YEH. T. and PATT, Y. N. *Branch Classification: A New Mechanism for Improving Branch Predictor Performance*, Micro-27, San Jose, California, pp. 22-31, November 1994.

[Cha95] CHANG, L; HAO, E. and PATT, Y. N. *Alternative Implementations of Hybrid Branch Predictors*, Micro-29, Ann Arbor, Michigan, pp. 252-257, November 1995.

[Che96] CHEN, I. K.; COFFEY, J. T. and MUDGE, T. N. *Analysis of Branch Prediction via Data Compression*, ASPLOS VII, pp. 128-137, October 1996.

[Ega00] EGAN, C. *Dynamic Branch Prediction in High Performance Superscalar Processors*, PhD thesis, University of Hertfordshire, August 2000.

[Int00] *IA-64 Application Developer's Guide*, Intel, 2000.

[Kes99] KESSLER, R. E. *The Alpha 21264 Microprocessor*, IEEE Micro, pp. 24-36, March 1999.

[Lee97] LEE, C. C.; CHEN, I. K. and MUDGE, T. N. *The Bi-Mode Branch Predictor*, Micro-30, Research Triangle Park, North Carolina, pp. 4-13, December 1997.

[McF93] McFARLING, S. *Combining Branch Predictors*, Western Research Laboratories Technical Report TN-36, June 1993.

[Pan92] PAN, S.; SO, K. and RAHMEH, J. T. *Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation*, ASPLOS-V, Boston, pp. 76 - 84, 1992.

[Spr97] SPRANGLE, E., CHAPPELL, R. S., ALSUP, M. and PATT, Y. N. The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference, ISCA '24, Denver, Colorado, pp. 284-291, June 1997.

[Ste97] STEVEN, G. B.; CHRISTIANSON, D. B.; COLLINS, R.; POTTER, R. D. and STEVEN, F. L. A Superscalar Architecture to Exploit Instruction Level Parallelism, Microprocessors and Microsystems, 20 (7), pp. 391 – 400, 1997.

[Ste00] STEVEN, G. B.; EGAN, C.; QUICK P. and VINTAN, L. A Cost Effective Cached Correlated Two-level Adaptive Branch Predictor, 18th IASTED International Conference on Applied Informatics (AI 2000), Innsbruck, February 2000.

[Ste01] STEVEN, G.; ANGUERA, R.; EGAN, C.; STEVEN, F. and VINTAN, L. Dynamic Branch Prediction Using Neural Networks, DSD 2001, September 2001, Warsaw, pp178-185.

[Yeh91] YEH, T. and PATT, Y. N. Two-Level Adaptive Training Branch Prediction, Micro-24, Albuquerque, New Mexico, pp. 51 - 61, November 1991.

[Yeh92] YEH, T. and PATT, P. Alternative Implementations of Two-Level Adaptive Branch Prediction, ISCA-19, Gold Coast, Australia, pp. 124 – 134, 1992.

[Yeh93] YEH, T. and PATT, Y. N. A Comparison of Dynamic Branch Predictors that use Two Levels of Branch *History*, ISCA - 20, pp. 257 - 266, May 1993.