Applying Caching to Two-level Adaptive Branch Prediction

Colin Egan, Gordon B. Steven
University of Hertfordshire
Hatfield, Hertfordshire, U.K.
AL10 9AB
email: C.Egan@herts.ac.uk

Won Shim
Seoul National Univ. of Technology
Seoul, Korea
139-743
wonshim@duck.snut.ac.kr

Lucian Vintan
University of Sibiu
Sibiu-2400, Romania
vintan@cs.sibiu.ro

Abstract

During the 1990s Two-level Adaptive Branch Predictors were developed to meet the requirement for accurate branch prediction in high-performance superscalar processors. However, while two-level adaptive predictors achieve very high prediction rates, they tend to be very costly. In particular, the size of the second level Pattern History Table (PHT) increases exponentially as a function of history register length. Furthermore, many of the prediction counters in a PHT are never used; predictions are frequently generated from non-initialised counters and several branches may update the same counter, resulting in interference between branch predictions. In this paper, we propose a Cached Correlated Two-Level Branch Predictor in which the PHT is replaced by a Prediction Cache. Unlike a PHT, the Prediction Cache saves only relevant branch prediction information. Furthermore, predictions are never based on uninitialised entries and interference between branches is eliminated. We simulate three versions of our Cached Correlated Branch Predictors. The first predictor is based on global branch history information while the second is based on local branch history information. The third predictor exploits the ability of cached predictors to combine both global and local history information in a single predictor. We demonstrate that our predictors deliver higher prediction accuracy than conventional predictors at a significantly lower cost.

Key Words

Two-level Adaptive Branch Predictors, Cached Correlated Branch Predictors, Prediction Cache.

1. Introduction

High-performance processors typically use dynamic branch prediction to avoid pipeline stalls whenever a branch is taken. A traditional Branch Target Cache (BTC), based on the previous history of each branch, gives a prediction accuracy of between 80 to 95% [1].

More recently, 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 [2] and by Pan, So and Rahmeh [3]. Although researchers report very high success rates with two-level adaptive predictors, this success is only achieved by providing very large arrays of prediction counters or PHTs (Pattern History Tables). Patt [4] argues that it will be practical to implement these large predictors in the early 21st century and suggests that between 256K bytes and 1024K bytes of the silicon budget should be devoted to branch prediction. We argue that such profligate use of silicon area is unlikely to be cost effective.

Two-level Adaptive Branch Predictors have two other disadvantages. Firstly, in most practical implementations each prediction counter may be shared between several branches. There is therefore interference 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. As a result, training requirements limit the amount of branch history that can be successfully exploited.

We have developed a Two-level Branch Predictor that addresses the three problems of conventional two-level predictors: cost, interference and initial training. We have called this novel predictor the Cached Correlated Branch Predictor. Through a disciplined use of silicon area, we dramatically reduce the cost of a Two-level Adaptive Branch Predictor. 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.

2. Two-level Adaptive Branch Prediction

Two-level branch predictors are usually classified using a system proposed by Yeh and Patt [2]. 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" in the middle 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 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, it should be emphasised that 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 history register bits and the prediction accuracy stops improving.

Several researchers have attempted to reduce interference in the PHT. The Gshare Predictor [5, 6], 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 [7] uses twin PHT arrays to decrease destructive interference between branch predictions and to maximise positive interference.

GAp was first proposed by Pan et al [3] 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, 2^{30+k} 2-bit 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, limited branch interference is reintroduced. As in the case of GAg, a separate BTC is required 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 [8] 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 in the PAg configuration 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.

Both PAg and PAs predictors require two sequential table accesses, one to the BTC to obtain the appropriate local history register and a second to the PHT, to obtain the prediction. However, to achieve high performance the prediction must be made in one clock cycle from the time the branch address is known. Fortunately, the next prediction for each branch can be determined as soon as the current instance of the branch is resolved. The next prediction can therefore be obtained as part of the predictor updating process and cached in the BTC [9].

3. Cached Correlated Branch Prediction

The high cost of Two-level Adaptive Branch Predictors is a direct result of the excessive size of the second level PHTs. In a Cached Correlated Predictor [10], 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. 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. A Cached Correlated Branch Predictor will therefore only be cost effective if 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-branch history registers. In an earlier feasibility study [11], 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 Prediction Cache would be too costly to implement 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.

3.1. Global Cached Correlated Predictor

Figure 1 shows a four-way set-associative Global Cached Correlated Branch Predictor.

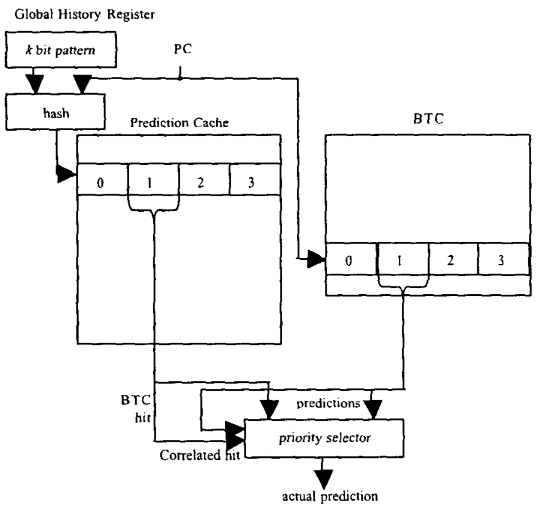

Figure 1: A Global Cached Correlated Branch Predictor.

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, a 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.

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 reduce the number of initial mispredictions. Secondly, the default predictor minimises the impact of misses in the Prediction Cache.

Hybrid predictors [5] also use two or more predictors to generate each prediction. A hybrid predictor, however, chooses dynamically between two or more 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 BTC when no other prediction is available.

3.2. Local Cached Correlated Predictor

The Local Cached Correlated Predictor also replaces the PHT with a Prediction Cache. However since a history register is now required for every branch, a local history register field is added to each BTC entry (Figure 2). As with the Global Cached Correlated Predictor, a prediction counter is also 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.

One problem with the local predictor as described is that two sequential table accesses are required to make a prediction, one to access the BTC and a second to access the Prediction Cache. This problem can be overcome by caching local predictions in the BTC so that the prediction is available after only one table access. As soon as the outcome of a branch is known the local history register and the Prediction Cache are updated. At this point, the prediction for the next encounter of the branch is fully determined. The prediction is then saved in the BTC entry for the branch to allow the next prediction to be made in one cycle.

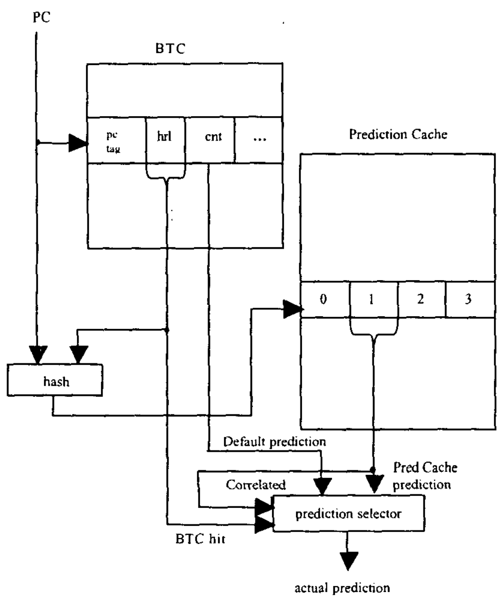

Figure 2: A Local Cached Correlated predictor.

4. A Combined Global and Local Predictor

Global predictors perform better with some branches, while local predictors perform better with others. It is therefore highly desirable to combine both forms of prediction within a single predictor. Unfortunately, combining both global and local history registers within a conventional two-level predictor is very costly since both the size and cost of the PHTs increase exponentially as a function of history register length. For example, consider combining a 16-bit global history register with a 16-bit local register; the PHT would require 2^{32} entries, a prohibitive size.

One possibility, used in the Alloyed Predictor [12], is to reduce the PHT size by hashing the history register bits before accessing the PHT. Alloyed Predictors improve the accuracy of conventional two-level predictors by over 20% [12]. However, the initialisation problem is not addressed and the hashing exacerbates the branch interference that is already an undesirable feature of conventional two-level predictors.

Alternatively, separate global and local predictors can be combined in a hybrid predictor [5]. Here two distinct predictors are provided and a further table of counters is used to dynamically select the most effective predictor for each branch at run time.

In contrast, it is very easy to combine global and local history information in a Cached Predictor since there is no explosive cost increase corresponding to the exponential growth of the PHT size. Instead, the total cost of the predictor increases only slowly as a function of history register length. Firstly, the size of the history register field in each cache entry must be increased. Secondly, the total size of the cache must be slowly increased to accommodate additional entries.

In this paper, we present a combined version of our Cached Correlated Branch Predictor for the first time (Figure 3). As in the local predictor, the PC is used to access the BTC which contains a local history register and a default prediction counter for each branch. The local history register is then hashed with the PC and the global register to obtain the index for the Prediction Cache. Tag fields in the Prediction Cache ensure that a hit is only recorded if the PC, local and global history register all match. Whenever possible the next prediction is taken from the Prediction Cache. However, as in our other cached predictors, a default prediction from the BTC is used whenever there is a miss in the Prediction Cache and a hit in the BTC.

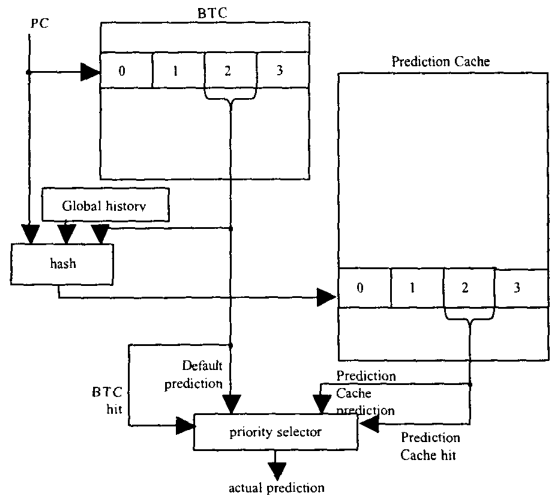

Figure 3: A Combined Cached Correlated Predictor.

5. Prediction Cache Access

We originally used a fully associative Prediction Cache to test our Cached Correlated Branch Predictor [11]. Clearly, a practical branch predictor must use either a direct mapped cache or a set associative organisation. However, the detailed organisation of this 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, an 8-way set associative cache. In the absence of collisions with other branches, each branch is restricted to only eight 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.

In general, 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.

As an example, consider a 16-bit history register (HR), a 30-bit PC and a Prediction Cache with 4K entries. A cache index of 12 bits is required. The following 12-bit groups are therefore XORed to generate the Prediction cache index:

HR: bits 11-0
PC: bits 9-2; HR: bits 15-12
PC: bits 21-10
PC: bits 31-22

In practice, the most significant bits of the PC change so infrequently that the final group can be discarded.

6. Simulation Results

In this section we quantify the performance of our three Cached Correlated Predictors and compare their performance and cost with conventional two-level predictors. 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 initial training problems are more acute. A classic BTC therefore achieves an average prediction accuracy of only 88.14% with the Stanford benchmarks.

The benchmarks were compiled for the Hatfield Superscalar Architecture (HSA) [13], 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 our branch prediction simulations. All the predictors simulated in this paper use a four-way set-associative BTC with 1K entries; sufficient entries are always available to minimise BTC misses.

6.1. Conventional Two-level Predictors

For comparative purposes, we first simulated a GAg predictor, a GAs predictor with 16 sets (GAs(16)) and a GAp predictor (Figure 4). 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 average misprediction rate of 9.23% is achieved with the GAs(16) configuration and 26 history register bits. In general, however, there is little benefit from increasing the history register length beyond 16-bits for GAg and 14-bits for GAs/GAp. Beyond this point, there is either no benefit from new correlations or any benefit is negated by the additional training required in the PHTs.

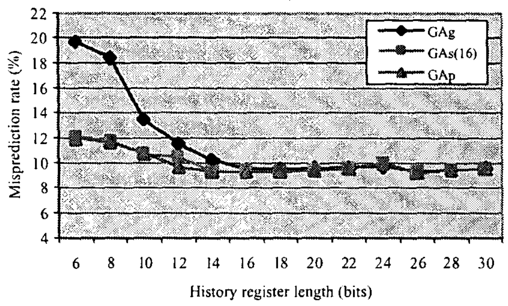

Figure 4: Conventional Global misprediction rates.

We also simulated conventional PAg, PAs and PAp predictors (Figure 5). 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. Local predictors are therefore able to benefit from longer history registers than their global counterparts.

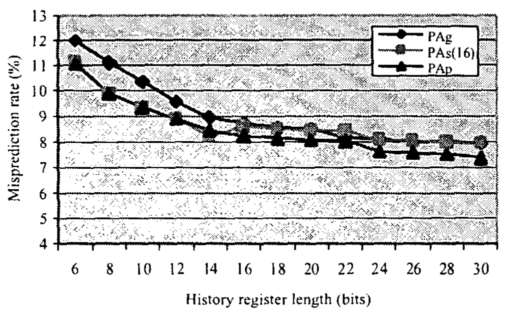

Figure 5: Conventional Local misprediction rates.

6.2. Global Cached Predictors

The average misprediction rates achieved with a four-way set-associative version of our Global Cached Correlated predictors are shown in Figure 6. The number of entries in the Prediction Cache is varied from 1K to 32K. 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 best misprediction rate of 5.99% is achieved with a 32K entry Prediction Cache and a 20-bit history register. This represents a 54% reduction over the best misprediction rate achieved by a conventional global two-level predictor.

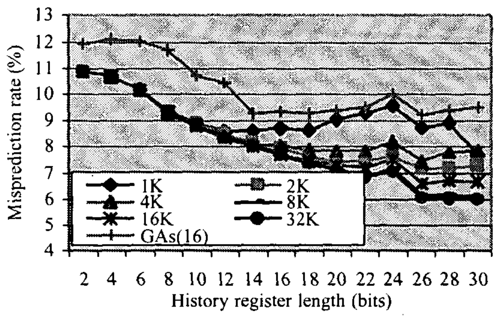

Figure 6: Global Cached Correlated Predictor misprediction rates.

6.3. Local Cached Predictors

The misprediction rates achieved by our Local Cached Correlated Predictor are recorded in Figure 7. The number of entries in the Prediction Cache is varied between 1K and 32K. Initially the misprediction rate falls steadily as a function of history register length. Then as more and more predictions need to be cached, the larger caches deliver superior prediction rates. However, no further benefit is derived from increasing the cache size beyond 16K. The best misprediction rate of 6.28% is achieved with a 16K cache and a 32-bit history register. This figure is marginally worse than the best global predictor, but represents a 15% improvement over the best PAg/PAp configuration.

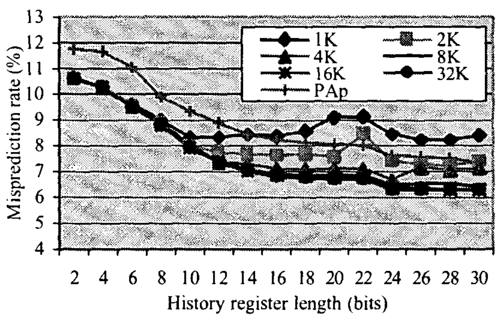

Figure 7: Local Cached Correlated Predictor misprediction rates.

6.4. Combined Cached Predictors

The misprediction rates achieved by our combined global and local cached predictors are given in Figure 8. In all cases, an equal number of local and global history register bits is provided. As before the size of cache is varied between 1K and 32K. This predictor is the only one that achieves a misprediction rate significantly below 6%. The lowest misprediction rate of 5.68% is obtained with a cache size of 32K, 24 global history register bits and 24 local history bits. This represents a 5.46% improvement over the best Global Cached Predictor and a 10.56% improvement over the best Local Cached Predictor. These results demonstrate that a Cached Correlated Branch Predictor can successfully exploit both global and local branch correlation within a single predictor.

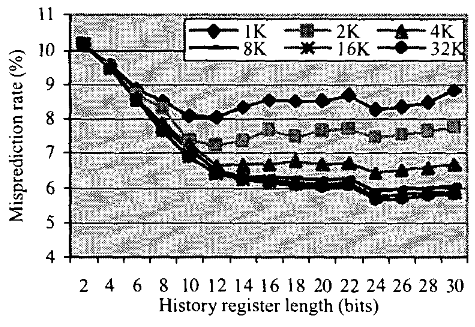

Figure 8: Combined Cached Correlated Predictor misprediction rates.

6.5. Cost Comparisons

Misprediction rates are only one metric; cost is also important. For example, the best Global Cached Correlated predictor requires 87 Kbytes of storage, and the best Local Cached Correlated Predictor requires 93.75 Kbytes of storage. However, these figures are completely dwarfed by the staggering 268 gigabytes of storage required by the best PAp predictor. Table 1 summarises the storage requirements of the Cached Correlated Predictors simulated in this paper. As can be seen, the cost of our cached predictors increases linearly as a function of history register length. In contrast, in traditional two-level predictors, the size of the PHT increases exponentially as a function of the history register length and as a result the total cost also rises exponentially as a function of history register length. For this reason, cached predictors are cheaper for larger history register sizes, and are therefore better placed to exploit additional branch correlation information.

In Figure 9, we compare the performance of global predictors with a maximum storage requirement of 250 Kbytes. As can be seen, the 1K Cached Correlated Predictor is more cost effective than any low-cost conventional global predictor. Similarly, the 16K Cached Correlated Branch Predictor outperforms conventional predictors with comparable cost. Local Cached Correlated Predictors deliver very similar cost advantages.

Finally, in Figure 10 we compare the costs of Combined Cached Predictors with conventional two-level predictors. Again, the Cached Predictors are more cost effective. However, in order to achieve misprediction rates under 6%, the total storage cost of the predictor must be increased to around 150K bytes.

The most important difference illustrated by Figure 9 and Figure 10 is the sharply contrasting impact on costs of increasing history register length. Cached Correlated Predictors can reasonably seek to exploit additional branch correlation by increasing the history register length to as much as 30 bits. In contrast, with conventional two-level predictors storage cost becomes a major concern with history register lengths in the 12 to 16 bit range.

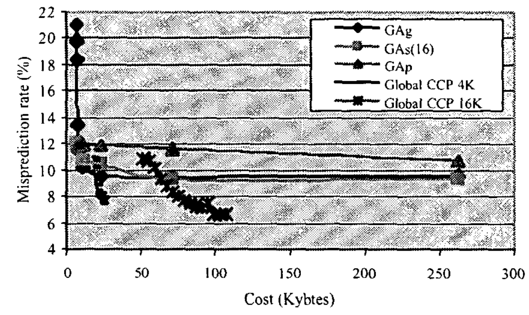

Figure 9: A comparison of Global predictor performance as a function of cost.

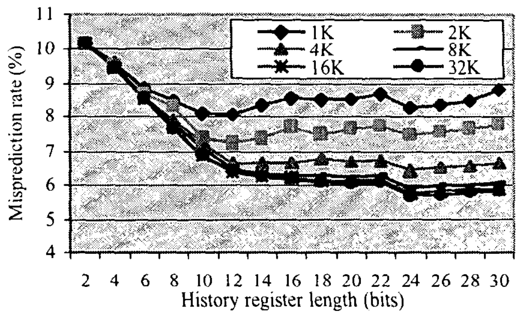

Figure 10: A comparison of Combined predictor performance as a function of cost.

7. Conclusions

Our simulations demonstrate that our Cached Correlated Branch Predictors are significantly more accurate and require less silicon area than conventional Two-level Adaptive Predictors. Our best global predictor is 54% better than the best GAs predictor and our best local predictor is 15% 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 is also due to the addition of default predictors in the BTC. As history register lengths increase, 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. Furthermore, the default counter effectively reduces the impact of misses in the Prediction Cache.

A major advantage of Cached Correlated Branch Predictors is their ability to exploit correlations from a large number of history bits. In our 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%, the lowest figure achieved by any predictor simulated in this paper and 29.4% better than the best conventional two-level predictor.

Finally, throughout this paper, we have been concerned with development of more cost-effective individual predictors. Nonetheless, all the predictors presented can be used as components in a hybrid predictor.

References

[1] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*, (Morgan Kaufmann, 1996).

[2] T. Yeh. and Y. N. Patt. *Alternative Implementations of Two-Level Adaptive Branch Prediction*, ISCA-19, Gold Coast, Australia, pp. 124 – 134, 1992.

[3] S. Pan, K. So and J. T. Rahmeh. *Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation*, ASPLOS-V, Boston, pp. 76 - 84, 1992.

[4] Y. N. Patt, S. J. Patel, M. Evers, D. H. Friendly and J. Stark. *One Billion Transistors, One Uniprocessor*, One Chip, Computer, pp. 51 - 57, 1987.

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

[6] P. Chang, E. Hao, T. Yeh and Y. N. Patt. *Branch Classification: A New Mechanism for Improving Branch Predictor Performance*, Micro-27, San Jose, California, pp. 22-31, November 1994.

[7] C. C. Lee, I. C. K. Chen and T. N. Mudge. *The Bi-Mode Branch Predictor*, Micro-30, Research Triangle Park, North Carolina, pp. 4-13, December 1997.

[8] T. Yeh and Y. N. Patt. *Two-Level Adaptive Training Branch Prediction*, Micro-24, Albuquerque, New Mexico, pp. 51 - 61, November 1991.

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

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

[11] G. B. Steven, C. Egan, P. Quick and L. Vintan. *A Cost Effective Cached Correlated Two-level Adaptive Branch Predictor*, 18th IASTED International Conference on Applied Informatics (AI 2000), Innsbruck, February 2000.

[12] K. Skadron, M. Martonosi and D. W. Clark. *A Taxonomy of Branch Mispredictions, and Alloyed Prediction as a Robust Solution to Wrong-History Mispredictions*, PACT-2000, Philadelphia, October 200.

[13] G. B. Steven, D. B. Christianson, R. Collins, R. D. Potter and F. L. Steven. *A Superscalar Architecture to Exploit Instruction Level Parallelism*, Microprocessors and Microsystems, 20(7), pp. 391 – 400, 1997.

	HR=2	HR=4	HR=6	HR=8	HR=10	HR=12	HR=14	HR=16	HR=18	HR=20	HR=22	HR=24	HR=26	HR=28	HR=30
Global 1K entries	10.25	10.50	10.75	11.00	11.25	11.50	11.75	12.00	12.25	12.50	12.75	13.00	13.25	13.50	13.75
Global 16K entries	51.00	55.00	59.00	63.00	67.00	71.00	75.00	79.00	83.00	87.00	91.00	95.00	99.00	103.0	107.0
Local 1K entries	10.50	11.00	11.50	12.00	12.50	13.00	13.50	14.00	14.50	15.00	15.50	16.00	16.50	17.00	17.50
Local 16K entries	51.25	55.50	59.75	64.00	68.25	72.50	76.75	81.00	85.25	89.50	93.75	98.00	102.3	106.5	110.8
Combined 1K entries	10.75	11.50	12.25	13.00	13.75	14.50	15.25	16.00	16.75	17.50	18.25	19.00	19.75	20.50	21.25
Combined 16K entries	55.25	63.50	71.75	80.00	88.25	96.50	104.75	113.00	121.25	129.50	137.75	146.00	154.25	162.50	170.75

Table 1: Cached Correlated Predictor Costs in Kbytes.