A COST-EFFECTIVE TWO-LEVEL ADAPTIVE BRANCH PREDICTOR

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

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

Vintan, L. *University "Lucian Braga" 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 access the same counter, resulting in branch interference or aliasing. In this paper, we propose a Cached Correlated Branch Predictor in which the PHT is replaced by a Prediction Cache. Unlike the PHT, the Prediction Cache saves only relevant branch prediction information. Furthermore, predictions are never based on uninitialised entries, and branch interference is eliminated. We simulate two versions of our cached correlated branch predictors, the first uses global branch history information and the second uses local branch history information. We demonstrate that our predictors deliver higher prediction accuracy than conventional predictors at a significantly lower cost.

Keywords: Branch Prediction, 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% (Hennessey and Patterson, 1996).

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 discarded. 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 branch prediction techniques in the 1990s with the development of

Two-Level Adaptive Branch Predictors by Yale Patt's group (Yeh and Patt, 1992) and by Pan, So and Rahmeh (Pan, *et* al., 1992). 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 (Patt, *et al*., 1987) 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 initial training required increases as additional branch history is exploited. As a result, counter initialisation limits 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 initialisation. We have called this novel predictor the Cached Correlated Branch Predictor. Through a disciplined use of silicon area, we dramatically reduce the cost of two-level adaptive branch prediction. 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 default prediction mechanism is included that is initialised after a single occurrence of a 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 any failures in the caching mechanism.

2. TWO-LEVEL BANCH PREDICTION

Two-level branch predictors are usually classified using a system proposed by Yeh and Patt (Yeh and Patt, 1992). 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 prediction counters. To generate a prediction, the *k* bit pattern in the first-level global history register is used to index the array of two bit saturating 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 global history register and the prediction counter in the PHT 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 in the PHT, the outcome of one branch may interfere with the prediction of all other branches. Although this branch interference limits the performance of global predictors, 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.

GAp was first proposed by Pan et al (Pan, *et al*., 1992) 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, the global PHT is replicated to provide a separate per-address PHT for each branch. Conceptually, the PC and 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, branch interference is now only partially eliminated. 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 (Yeh and Patt, 1995) was later classified as PAg. PAg conceptually uses a different local branch history register for each branch or a Per-Address Branch History Table and a single shared global PHT. Each branch prediction is therefore based entirely on the history of the branch being predicted. The Branch History Table can be combined with the BTC by adding a history register field to each entry in a traditional BTC. Since all branches share a single PHT, PAg is also characterised by interference between different branches.

The interference in the PAg configuration can be removed 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. Instead, 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 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 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 (Yeh and Patt, 1993).

3. CACHED CORRELATED 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, the second-level table is therefore replaced with a Prediction Cache, while the first level is unchanged. Unlike conventional two-level predictors, the number of entries in the Prediction Cache is not a 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 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 Predictors are presented in this paper. The first predictor employs a global history register, while the second employs multiple local or per branch history registers. In an earlier feasibility study (Steven, *et al*., 2000) 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, 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. Each entry in the Prediction Cache consists of a PC tag field, a history register tag field, 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 prediction counter and therefore consists of a branch target address, a branch address tag, a two-bit saturating counter, a valid bit and a LRU field.

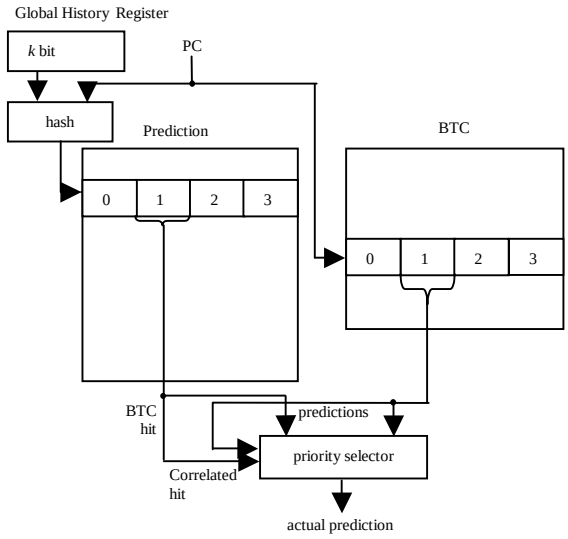

Figure 1: A four-way set-associative Global Cached Correlated Branch Predictor.

The Prediction Cache index is obtained by hashing the PC with the global history register bits while the BTC is accessed using the least significant bits of the PC. 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 and initialised. Finally, the global history register is updated.

Providing a default predictor has several advantages. Firstly, the default counters are initialised after only a single encounter with a branch. In contrast, with a *k* bit history register, up to 2^k counters must be initialised for each branch before the two-level predictor is fully functional. 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 (McFarling, 1993) also use two or more distinct predictors to generate each prediction. A hybrid predictor, however, uses a selection mechanism to choose dynamically between two or more predictions, 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 prediction 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 (Figure 2). However, since a history register is now required for every branch, a local history register field is now added to each BTC entry. As with the Global Cached Correlated Predictor, a default prediction counter is also added to 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 counter. 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 default counter in the BTC is used.

One problem with the local predictor as described is that two sequential 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. This prediction is then stored in the BTC entry for the branch to allow the next prediction to be made in one cycle.

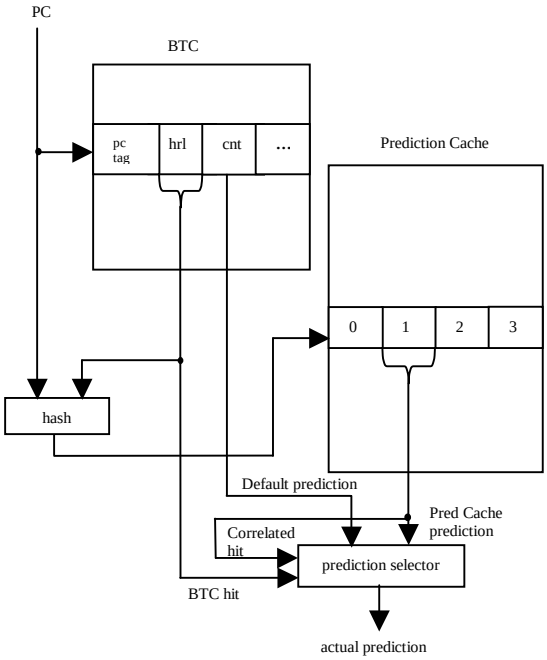

Figure 2: A four-way set-associative Local Cached Correlated predictor.

3.3. The Prediction Cache Organisation

We originally used a fully associative Prediction Cache to test our Cached Correlated Branch Predictor (Steven, *et al*., 2000). 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 PCs, each PC is still 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 PC. Although

many of these 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, collisions are frequent enough to prevent the predictor from reaching its full potential.

We found that the most accurate predictions are obtained when the history register bits are hashed with the PC bits to form a cache index. In this paper, the PC is XORed with the history register bits to form the cache index. The tag field in the Prediction Cache therefore consists of the most significant bits of the PC and all of the history register bits.

4. SIMULATION RESULTS

In this section we quantify the performance of set-associative Cached Correlated Predictors and compare their performance and cost with conventional two-level predictors. Both global and local predictors are evaluated.

Our simulations use 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. As a result, the branches are more difficult to predict and predictor initialisation problems are more acute. A classic BTC therefore achieves an average misprediction rate of only 11.86% with the Stanford benchmarks.

The benchmarks were compiled for the Hatfield Superscalar Architecture (HSA) (Steven, 1997), 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 therefore always available to minimise BTC misses.

4.1. Global Predictors

For comparative purposes, we first simulated a GAg predictor, a GAs predictor with 16 sets and a GAp predictor (Figure 3). 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 branch correlations or any benefit is negated by the additional initialisations required in the PHTs.

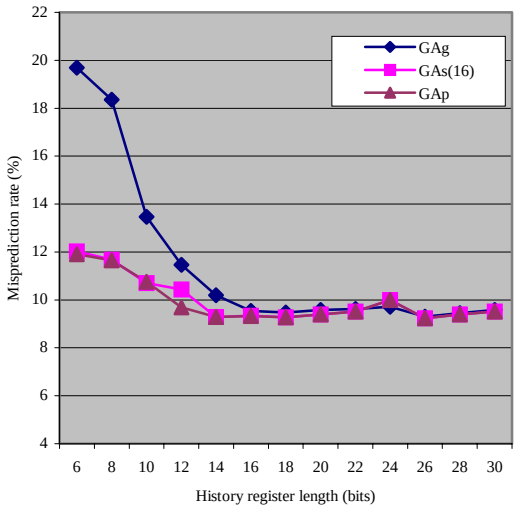

Figure 3: Conventional global two-level misprediction rates.

The average misprediction rates achieved with a four-way set associative version of our Global Cached Correlated predictors are shown in Figure 4. 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 versions. 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 size of the Prediction Cache the better the misprediction rates. The best misprediction rate of 5.99%, achieved with a 32K entry Prediction Cache and a 20-bit history register, represents a 54% reduction over the best misprediction rate achieved by a conventional global two-level predictor.

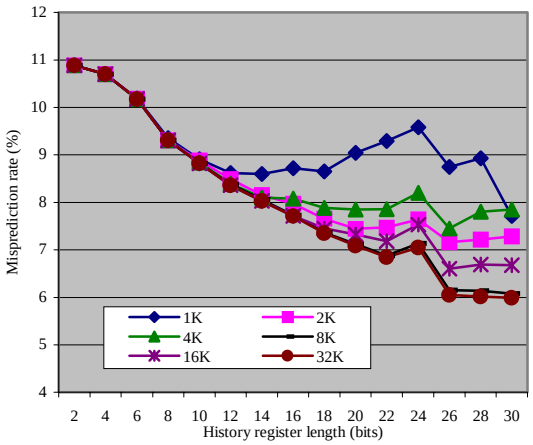

Figure 4: Global Cached Correlated misprediction rates.

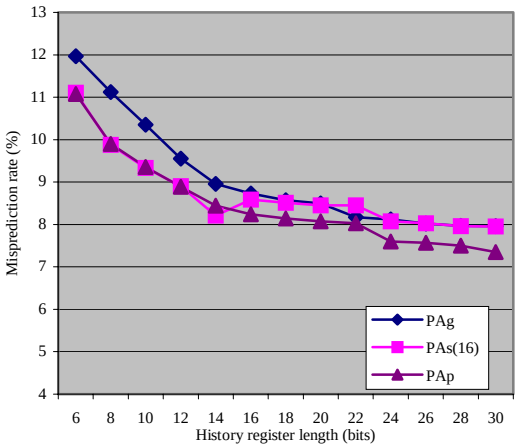

Figure 5: Conventional local two-level misprediction rates.

4.2. Local Predictors

Again for comparative purposes, we first simulated conventional PAg, PAs and PAp predictors (Figure 5). Conventional local predictors achieve average misprediction rates of around 7.5%, better than GAg/GAs, but still significantly worse than the best Global Cached Correlated Predictor. 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.

The misprediction rates achieved by our Local Cached Correlated Predictor are recorded in Figure 6. The number of entries in the 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.

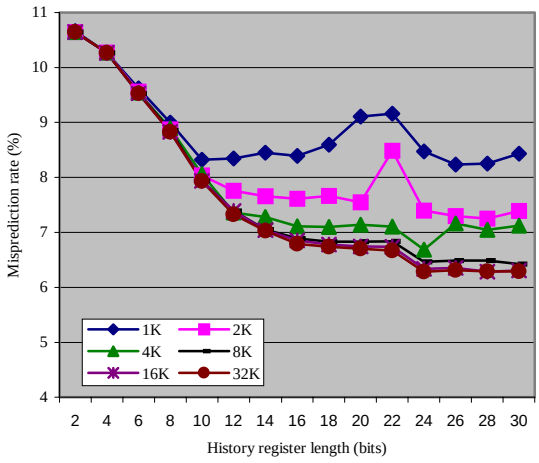

Figure 6: Local Cached Correlated Predictor misprediction rates.

4.3. *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 only 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 cost also rises exponentially as a function of history register. For this reason, cached predictors are cheaper when more history register bits are used and are therefore better placed to exploit additional branch history information.

In Figure 7, we compare the performance of global predictors with a maximum storage requirement of 250 Kbytes. As can be seen, the 4K Cached Correlated Predictor (CCP) is more cost effective than any low-cost conventional global predictor. Similarly, the 16K Cached Correlated Branch Predictor outperforms conventional predictors with comparable cost. Similar cost comparisons are shown in Figure 8 for local predictors.

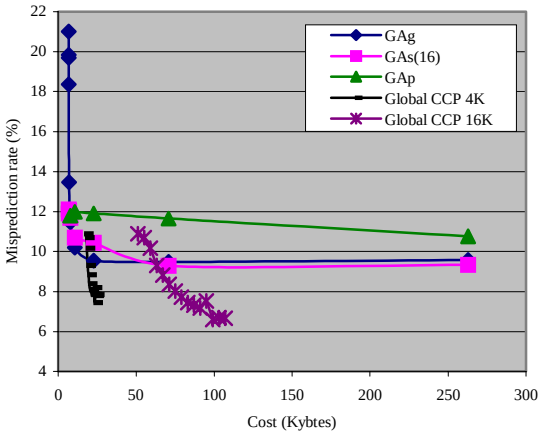

Figure 7: A comparison of global predictor performance as a function of cost.

The most important difference illustrated by Figure 7 and Figure 8 is the sharply contrasting impact on costs of increasing history register length. Cached Correlated Predictors can safely seek to exploit additional branch correlation by increasing the history register length to as much as 30 bits. In contrast, with conventional two-level 0predictors, storage cost becomes a major concern when the length of the history register reaches about half this size.

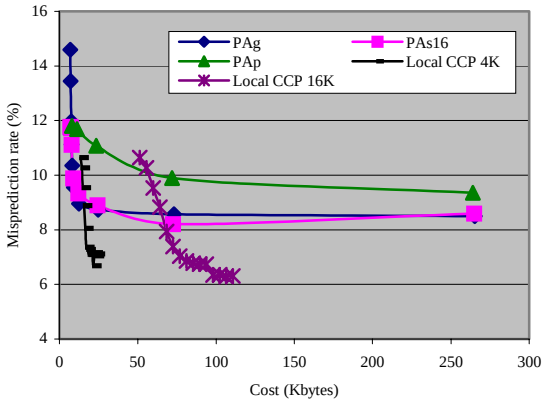

Figure 8: A comparison of local predictor performance as a function of cost.

5. CONCLUSIONS

Our simulations suggest that Cached Correlated Predictors are both significantly more accurate and require less silicon area than conventional Two-level Adaptive Predictors. We ascribe this higher accuracy to our more disciplined approach. Our predictions are always based on counters that have been trained as a result of 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 and is therefore able to provide predictions while further Prediction Cache entries are being initialised. Furthermore, the default counter effectively reduces the impact of misses in the Prediction Cache.

REFERENCES

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

McFarling, S. (June 1993). Combining Branch Predictors, *Western Research Laboratories Technical Report TN-36*.

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

Patt, Y. N., S. J. Patel, M. Evers, D. H. Friendly and J. Stark. (1987). One Billion Transistors, One

Uniprocessor, One Chip, *Computer*, pp. 51 – 57.

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

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

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

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

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

Table 1: Cached Correlated Predictor Costs in Kbytes.

	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