Branch prediction using both global and local branch history information

M.-C. Chang and Y.-W. Chou

Abstract: As the pipeline depth and issue rate of high-performance superscalar processors increase, the importance of an excellent branch predictor becomes more crucial to delivering the potential performance of a wide-issue, deep pipelined processor. Conventional two-level branch predictors predict the outcome of a branch either based on the local branch history information, comprising the previous outcomes of a single branch, or based on the global branch history information, comprising the previous outcomes of all branches. The authors propose a new branch prediction scheme, called LGshare, which employs both the global and local branch history information simultaneously to improve the branch prediction accuracy for superscalar processors. It is shown that LGshare can achieve higher branch prediction accuracy than conventional two-level predictors when the size of the pattern history table is fixed.

1 Introduction

Today's superscalar processors achieve high performance by executing multiple independent instructions in parallel. One of the most important impediments to the performance of wide-issue superscalar processors is the presence of conditional branches. Branches hinder the fetching and execution of the instructions following the branches, and thus cause the functional units to be underutilised. Speculative execution can alleviate the effects of branches by predicting the outcomes of branches and executing instructions speculatively before their control dependence branches are resolved. However, if a branch is mispredicted, all speculative work beyond the branch must be discarded. Thus, accurate branch prediction is essential for achieving high performance in wide-issue superscalar processors.

Branch prediction schemes can be classified as static or dynamic. Static branch prediction schemes predict branches statically by examination of the program behaviour and by the use of profile information gathered from earlier runs of the programs [1–4]. Dynamic branch prediction schemes use branch run-time execution history to make predictions [5–16], and have been shown to achieve higher prediction accuracy than static branch prediction schemes.

Smith [1] proposed a dynamic branch prediction scheme that uses a table of 2-bit saturating up-down counters to keep trace of the direction a branch is likely to take. Each branch is mapped via its address to a counter in the table. A branch is predicted as taken if the most significant bit of the corresponding counter is 1; otherwise, it is predicted as untaken. When the branch outcome is resolved, the associated counter is updated. If the branch outcome is taken, the associated saturating counter is incremented by 1; otherwise, the associated saturating counter is decremented by 1.

Yeh and Patt [5] proposed the two-level adaptive branch predictors, which have achieved substantially higher accuracy than previous schemes, and many later researches are based on their work. Two-level branch predictors make prediction either based on the global branch history or based on the local branch history. The PAg branch predictor makes predictions based on the per-address branch history (local branch history); the GAg branch predictor makes predictions based on the global branch history [5]. McFarling [6] proposed the gshare branch predictor, a variation of the GAg predictor. The gshare predictor makes predictions based on the global branch history, and it is considered as a better predictor than GAg.

However, some branches in a program cannot be correctly predicted either based on the global branch history merely or based on the local branch history merely. In this paper, we propose a new branch predictor, LGshare, which combines the global branch history and the local branch history to make predictions. We show that the LGshare can perform branch prediction better than conventional two-level branch predictors.

2 Related works

Various branch prediction schemes have been studied to improve prediction accuracy. In this Section we introduce two related branch prediction schemes, the PAg branch predictor and the gshare branch predictor.

Yeh and Patt [5] proposed the two-level branch predictor, which uses two levels of history to make branch predictions. As shown in Fig. 1, the first-level history, stored in the branch history register (BHR), records the outcomes of the most recently executed branches, and the second-level history, stored in the pattern history table (PHT), keeps track of the most likely direction of a branch when a

IEE Proceedings online no. 20020273

DOI: 10.1049/ip-cdt:20020273

Paper first received 26th June and in revised form 6th December 2001

The authors are with the Department of Electrical Engineering, Chang-Gung University, 259 Wen-Hwa 1st Road, Kwei-Shan, Tao-Yuan 333, Taiwan

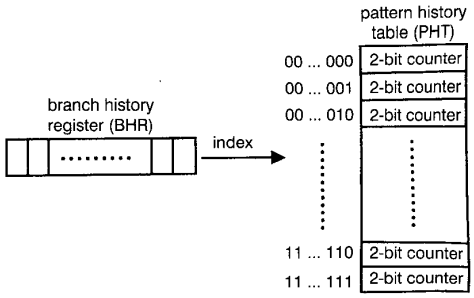

Fig. 1 *Structure of GAg branch predictor proposed by Yeh and Patt*

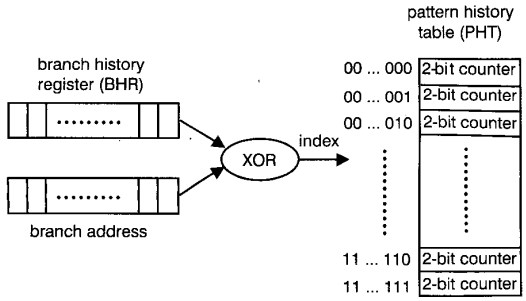

Fig. 2 *Structure of gshare branch predictor proposed by McFarling*

particular pattern is encountered in the first-level history. In a two-level branch predictor, each PHT entry is a 2-bit saturating counter that is used to predict whether a branch using that entry is taken or not taken. The counter's state is updated when the outcome of the branch is resolved. If the branch was taken, the corresponding counter is incremented. If the branch was not taken, the corresponding counter is decremented. If there is one BHR per branch, the predictor is called a per-address (PAg) predictor. If there is only one BHR to record the outcomes of all branches, it is called a global (GAg) predictor [5, 7].

McFarling [6] proposed the gshare branch predictor (Fig. 2), a variation of the global-history two-level branch predictor, which combines a global branch history with the branch address to select a 2-bit counter from the PHT. As shown in Fig. 2, the structure of the gshare predictor is very similar to that of the GAg predictor except that in gshare the global branch history is XORed with the branch address to form the index to PHT. Gshare can achieve higher prediction accuracy than GAg because the addressing scheme of gshare can reduce the possibility of branch interferences in the pattern history table [6].

Vintan and Egan [14] proposed a branch predictor, called MPAg, which makes use of improved path information (i.e. the successive branch addresses) to uniquely identify each program path. The MPAg predictor can overcome the problem that the limited information in the traditional global branch history register is insufficient to allow one path to a branch to be distinguished from other potential paths to the same branch.

Our LGshare predictor makes predictions based on both local and global branch history. A similar idea can also be found in the combined cached correlated predictor [16]. In the combined cached correlated predictor, the second level table, PHT, is replaced with a prediction cache, and the PC, local history and global history are hashed together 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 history and global history all match. In this way, interference between branches can be eliminated. However, to implement caching, a tag field must be added to each entry. A cached correlated branch predictor will be cost-effective if the cost of redundant counters removed from the PHT exceeds the cost of the added tags.

3 LGshare branch predictor

Conventional branch predictors make predictions either based on the global branch history information or based on the local branch history information. Global two-level branch predictors, such as GAg and gshare, make predictions based on the global branch history, and the reason they can work well is that many branches in the programs are correlated. That is to say, we can predict the outcome of a branch based on the outcomes of other branches. This kind of correlation between branches is called inter-branch correlation. Consider the example in Fig. 3. There are four if-statements in the code fragment, and they will be translated to conditional branch instructions after compilation. In this example, branches C and D are correlated with branches A and B. When the results of branches A and B are determined, the results of branches C and D are also determined. A global two-level branch predictor is able to record the various outcomes of branches C and D corresponding to different outcome combinations of branches A and B. When a specific outcome combination of branches A and B happens again, the branch predictor can predict the results of branches C and D by looking up the pattern history table. In real programs there exists a large amount of inter-branch correlation, which is the reason why global two-level branch predictors can work.

In contrast to global branch predictors, per-address two-level predictors such as PAg predict the outcomes of a branch based on the local branch history, i.e. the previous outcomes of the branch itself. This kind of correlation between the results of a single branch is called intra-branch correlation. Intra-branch correlation often arises from loop branches with a regular number of iterations and branches with periodic outcome patterns. Consider the example in Fig. 4: the while-loops in the code fragment will be translated to conditional branches, denoted as branches E and F, after compilation. Branches E and F will be taken nine times and then untaken once repeatedly. Per-address two-level predictors keep a dedicated BHR for each branch to record its previous outcomes. If a branch has periodic outcomes and the length of the BHR is long enough to capture the whole periodic pattern, per-address two-level predictors are able to predict the result of the branch perfectly. In real programs many branches exhibit intra-branch correlation, which is why per-address two-level predictors work well.

```
flag1 = 0;  flag2 = 0;
if (x < y)   /* Branch A */
  flag1 = 1;
if (x < z)   /* Branch B */
    flag2 = 1;
if (x < y || x < z)   /* Branch C */
    cout << "x is not largest";
if (flag1 && flag2)   /* Branch D */
    cout << "x is smallest";
```

Fig. 3 *Example of inter-branch correlation*

```
i = 1;
while (i < 10)  {/* Branch E */
    j = 1;
    while (j < 10)  {/* Branch F */
      cout << i * j << endl;
      j = j + 1;
      }
    i = i + 1;
    }
```

Fig. 4 *Example of intra-branch correlation*

We have shown examples where branches are predictable based on the global branch history or based on the local branch history. However, there exist branches that are not predictable based on merely the global branch history or merely the local branch history. Consider the example in Fig. 5. This example contains three if-statements, each of which will be translated to a conditional branch after compilation. We shall use the notation C1 to denote the condition $\langle i\%35 == 0\rangle$, and the notation C2 to denote the condition $\langle i\%3! = 0\rangle$. In this example, branch Z has partial correlation with branches X and Y. That is, if the results of branches X and Y are both known, the condition C1 can be determined. However, since the outcome of branch Z also depends on condition C2, it is not sufficient to derive the outcome of branch Z from the outcomes of branches X and Y. In this example, global two-level branch predictors cannot predict the outcome of branch Z exactly due to lacking the information of C2. Similarly, per-address two-level predictors cannot predict the outcome of branch Z exactly because the BHR fails to capture the whole periodic outcome pattern of branch Z unless the BHR has a nonrealistic length of more than 105 bits.

To improve the branch prediction accuracy, we propose a new branch predictor, called LGshare, which exploits both of the global branch history and the local branch history simultaneously to enhance branch prediction accuracy. Fig. 6 shows the structure of LGshare. The LGshare predictor has an n-bit global BHR to record the recent n outcomes of all branches and an m-bit local BHR (i.e. per-address BHR) for each branch to record the m recent outcomes of the branch. In real implementation, the local BHRs are attached with the entries of BTB (branch target buffer). Every time when the PHT is to be accessed, the

```
i = 1;
while (i < 1000) {
  ...
  if (i%5 == 0)   /* Branch X */
    cout << "5 divides" << i << endl;
  if (i%7 == 0)   /* Branch Y */
    cout << "7 divides" << i << endl;
  if ((i%35 == 0) && (i%3! = 0))   {/* Branch Z */
    cout << "35 divides" << i << endl;
    cout << "3 does not divide" << i << endl;
    }
  ...
  i++;
  }
```

Fig. 5 *Example of hybrid correlation*

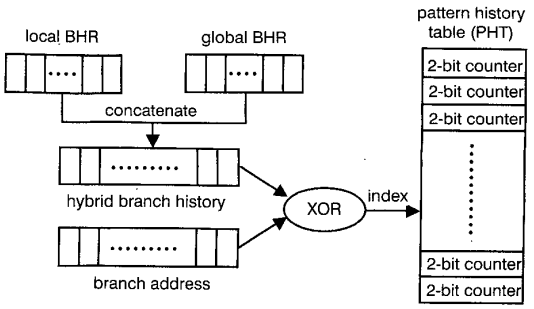

Fig. 6 *Structure of LGshare predictor*

m-bit history in local BHR and the n-bit history in global BHR are concatenated to form the hybrid branch history, and then the hybrid branch history is XORed with the branch address to form the index to PHT.

Since the LGshare branch predictor makes predictions based on both the global and local history, it is very possible for LGshare to achieve higher accuracy than conventional two-level predictors. Consider the example in Fig. 5 again. As stated previously, conventional two-level predictors cannot predict the outcome of branch Z well. However, LGshare is able to predict the result of branch Z perfectly. In LGshare, as the index to PHT is based on both the global branch history and the local branch history, for each different combination of global branch history (say, the outcomes of branches X and Y) and local branch history (say, the outcomes of branch Z in previous iterations) there is a dedicated counter in the PHT for tracing the behaviour of branch Z. Hence, when a specific combination of the global and local branch history happens again, the LGshare predictor can correctly predict the outcome of branch Z by looking up the corresponding counter in PHT. If the global BHR has a length of more than 2 bits and the local BHR has a length of more than 3 bits, the LGshare predictor can predict the outcome of branch Z exactly. The performance of LGshare will be compared with conventional two-level branch predictors in the next section.

4 Simulation results

In this Section, we compare the performance of three branch predictors, LGshare, gshare and modified PAg. The structures of LGshare and gshare have been shown in Figs. 6 and 2, respectively. The structure of modified PAg is very similar to that of PAg except that in modified PAg the per-address branch history is XORed with the branch address to form the index to PHT. Modified PAg and gshare can be considered as special cases of LGshare. As shown in Fig. 6, when the length of hybrid branch history is fixed, LGshare will be degenerated to gshare if the length of local branch history is 0. Similarly, LGshare will be degenerated to modified PAg if the length of global branch history is 0. Modified PAg makes predictions based on local branch history only; gshare makes predictions based on global branch history only. LGshare makes predictions by employing both of global and local branch history simultaneously.

Six benchmarks from SPECint95 benchmark suit are used in the experiment. The six benchmarks include gcc, li, perl, m88ksim, vortex, compress95. All benchmarks were

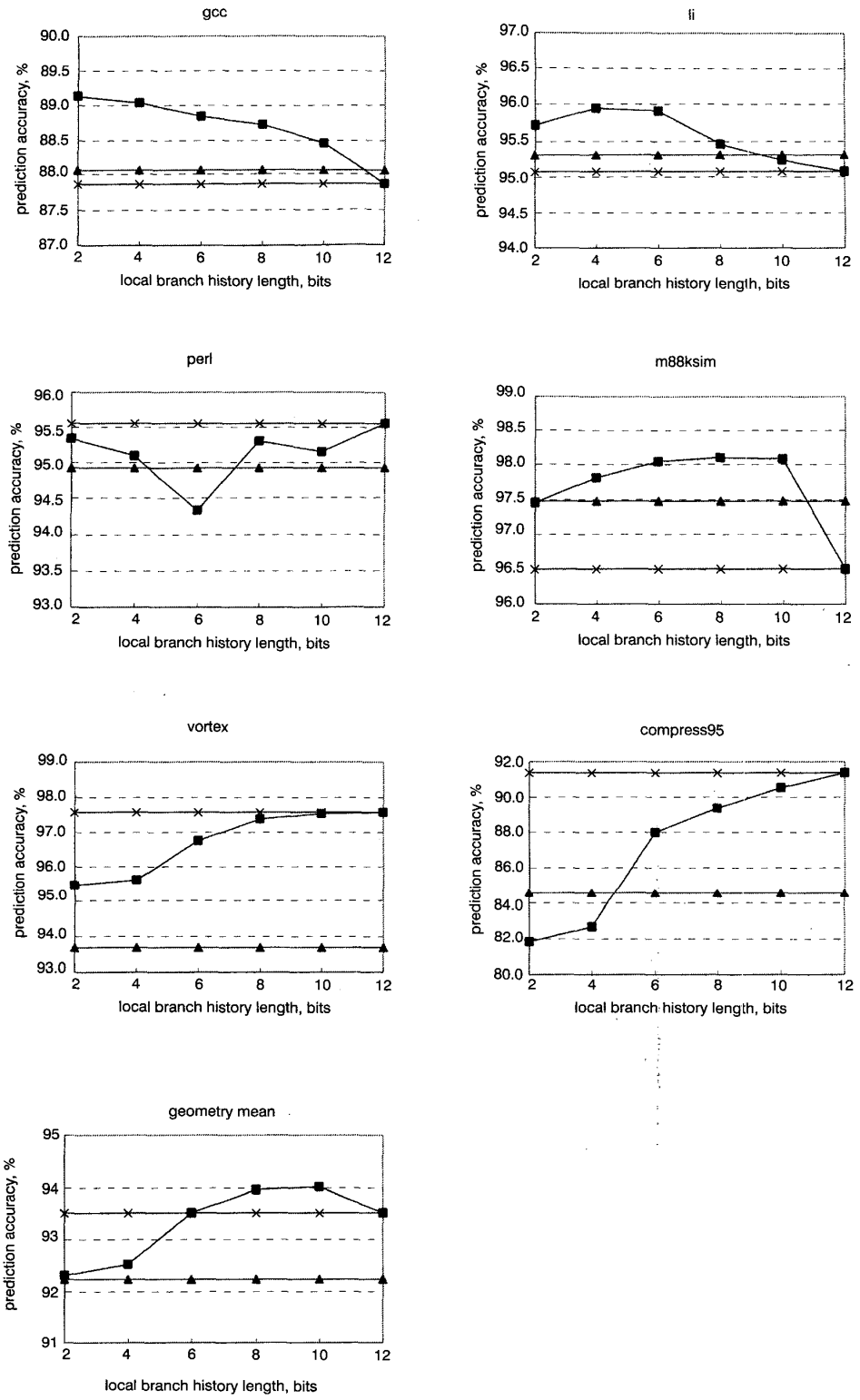

Fig. 7 *Performance comparisons of three branch predictors with 4k-entry PHT*

—■— LGshare
—×— modified PAg
—▲— gshare

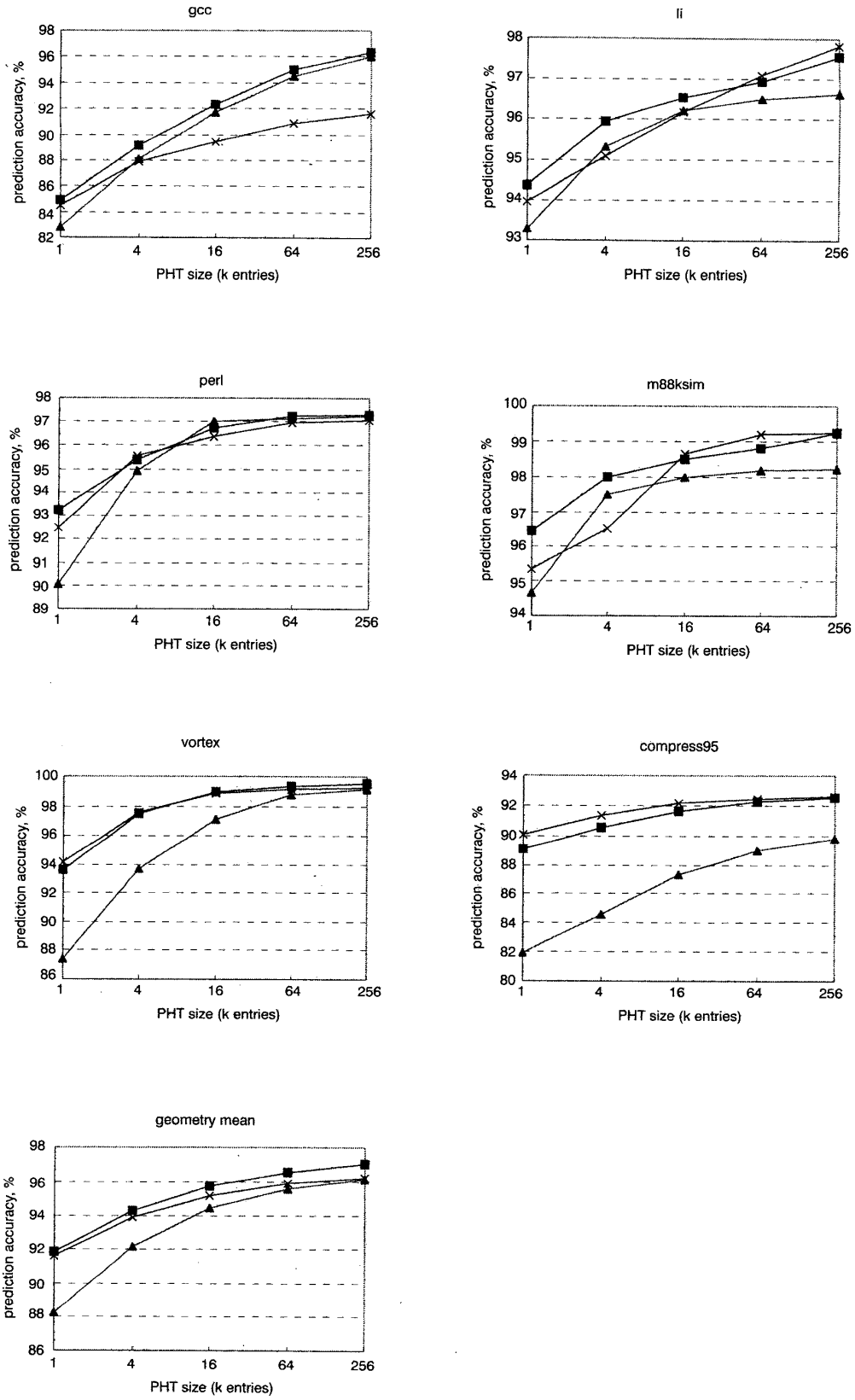

Fig. 8 *Performance comparisons of three branch predictors when PHT size is varied*

—■— LGshare
—×— modified PAg
—▲— gshare

simulated until finish or for over 10 000 000 conditional branches.

Fig. 7 shows the performance comparisons of the three branch predictors when the size of PHT is fixed at 4096-entry. In this example, modified PAg makes predictions based on 12-bit local branch history; gshare makes predictions based on 12-bit global branch history. LGshare makes predictions based on 12-bit hybrid branch history, consisting of m-bit local branch history and $(12 - m)$-bit global branch history, and its performance varies with m, the local branch history length. The x-axis in Fig. 7 represents the local branch history length m used in the LGshare scheme, and it has no relation with either gshare or modified PAg. From the simulation results in Fig. 7, we see that no one predictor can beat the other two predictors absolutely for all six benchmarks. However, on the average, the LGshare predictor can outperform gshare and modified PAg when the length of local branch history is selected to be around 8-bit. The average prediction accuracy of LGshare ($m = 10$), modified PAg and gshare are 94.0%, 93.5% and 92.2%, respectively. That is, the misprediction rates of LGshare ($m = 10$), modified PAg and gshare are 6.0%, 6.5% and 7.8%, respectively. As the branch penalty is proportional to branch misprediction rate, about 7.7% of branch penalty can be deleted when LGshare replaces the modified PAg branch prediction scheme, and 23.1% of the branch penalty can be deleted when LGshare replaces the gshare branch prediction scheme.

Fig. 8 shows the performance comparisons of the three branch predictors when the size of PHT is varied from 1k to 256k-entry. On the average, LGshare can outperform gshare and modified PAg. When the size of PHT is 256k-entry, the average prediction accuracy of LGshare, modified PAg and gshare are 97.05%, 96.23% and 96.15%, respectively. That is, the misprediction rates of LGshare, modified PAg and gshare are 2.95%, 3.77% and 3.85%. Hence, when the PHT size is fixed at 256k-entry, 21.75% of branch penalty can be deleted when LGshare replaces the modified PAg branch prediction scheme, and 23.38% of the branch penalty can be deleted when LGshare replaces the gshare branch prediction scheme.

5 Conclusions

Conditional branches significantly reduce the amount of instruction level parallelism that can be exploited by today's wide-issue superscalar processors. To alleviate the negative impact of conditional branches, a branch predictor with very high accuracy is essential to a superscalar processor. Conventional two-level branch predictors make predictions either based on global branch history only or based on local branch history only. In this paper, we have proposed a new branch predictor, LGshare, which makes predictions based on both inter-branch correlation and inter-branch correlation simultaneously. We have shown that LGshare can achieve higher prediction accuracy than gshare and modified PAg. When the PHT size is fixed at 4k-entry, 7.7% of branch penalty can be removed if LGshare replaces the modified PAg branch prediction scheme, and 23.1% of the branch penalty can be removed if LGshare replaces the gshare branch prediction scheme. When the PHT size is fixed at 256k-entry, 21.75% of branch penalty can be removed when LGshare replaces the modified PAg branch prediction scheme, and 23.38% of the branch penalty can be removed when LGshare replaces the gshare branch prediction scheme.

6 References

1 SMITH, J.E.: 'A study of branch prediction strategies'. 8th annual international symposium on *Computer architecture*, 1981, pp. 135–148
2 McFARLING, S., and HENNESSY, J.: 'Reducing the cost of branches'. 13th annual international symposium on *Computer architecture*, 1986, pp. 396–403
3 FISHER, J.A., and FREUDENBERFER, S.M.: 'Predicting conditional branch directions from previous runs of a program'. 5th international conference on *Architectural support for programming languages and operating systems*, October 1992, Boston, pp. 85–95
4 DEITRICH, B.L., CHEN, B.C., and HWU, W.W.: 'Improving static branch predictor in a compiler'. 1998 international conference on *Parallel architectures and compilation techniques*, 1998, pp. 214–221
5 YEH, T.-Y., and PATT, Y.N.: 'Alternative implementations of two-level adaptive branch prediction'. 19th annual international symposium on *Computer architecture*, 1992, pp. 124–134
6 McFARLING, S.: 'Combining branch predictors'. Technical Report TN-36, Digital Western Research Laboratory, June 1993
7 CHANG, P.-Y., HAO, E., YEH, T.-Y., and PATT, Y.N.: 'Branch classification: a new mechanism for improving branch predictor performance'. 27th annual ACM/IEEE international symposium on *Microarchitecture*, 1994, pp. 22–31
8 CHANG, P.-Y., HAO, E., and PATT, Y.N.: 'Alternative implementations of hybrid branch predictors'. 28th ACM/IEEE annual international symposium on *Microarchitecture*, 1995, pp. 252–257
9 CHANG, P.-Y., EVERS, M., and PATT, Y.N.: 'Improving branch prediction accuracy by reducing pattern history table interference'. 1996 international conference on *Parallel architecture and compilation techniques*, 1996, pp. 48–57
10 LEE, C.-C., CHEN, I.K., and MUDGE, T.N.: 'The bi-mode branch predictor'. 30th annual IEEE/ACM international symposium on *Microarchitecture*, 1997, pp. 4–13
11 SPRANGLE, E., CHAPPELL, R.S., ALSUP, M., and PATT, Y.N.: 'The agree predictor: a mechanism for reducing negative branch history interference'. 24th annual international symposium on *Computer architecture*, 1997, pp. 284–291
12 EVERS, M., PATEL, S.J., CHAPPELL, R.S., and PATT, Y.N.: 'An analysis of correlation and predictability: what makes two-level branch predictors work'. 25th annual international symposium on *Computer architecture*, 1998, pp. 52–61
13 EDEN, A.N., and MUDGE, T.: 'The YAGS branch prediction scheme'. 31st annual IEEE/ACM international symposium on *Microarchitecture*, 1998, pp. 69–77
14 VINTAN, L.N., and EGAN, C.: 'Extending correlation in branch prediction schemes'. 25th EUROMICRO conference, 1999, pp. 441–448
15 MANNE, S., KLAUSER, A., and GRUNWALD, D.: 'Branch prediction using selective branch inversion'. 1999 international conference on *Parallel architectures and compilation techniques*, 1999, pp. 48–56
16 EGAN, C.: 'Dynamic branch prediction in high performance superscalar processors'. PhD dissertation, Department of Computer Science, Faculty of Engineering and Information Sciences, University of Hertfordshire, UK, August 2000