A Comparative Study of Branch Predictors

Aparna Kotha
Graduate Student
University of Maryland,College Park
www.ece.umd.edu/∼akotha

10 Dec 2007

1 Introduction

A branch predictor is the part of the processor that determines whether a conditional branch in the instruction flow of a program is likely to be taken or not. Almost all pipelined processors have branch predictors, because they must guess the address of the next instruction to fetch before the current conditional instruction has been executed [1]. Hence, attempts have been made to design accurate branch predictors. This project attempts at modeling different branch predictors and comparing their performances for a set of benchmarks.

2 Branch Predictors

The intuition behind designing a branch predictor is that the behavior of typical branches is far from random. Most branches are either usually taken or usually not taken [4]. This intuition led to the design of many branch predictors of which four have been modeled and studied as a part of this project; they are

Bimodel: The lower order bits of the branch address are used to index the *branch history table* of 2-bit values. An entry of 0 or 1 in the branch history table predicts a "not taken" for the branch and an entry of 2 or 3 predicts a "taken".The 2-bit values are updated as 2-bit counters with 0 indicating "strongly not taken" and 3 indicating "strongly taken". Every time a branch is taken the corresponding 2-bit counter is incremented saturating at 3 and when it is not-taken the counter is decremented saturating at 0.

If two branches have the same lower order bits for their branch addresses, then the two branches will have the same entry in the bimodel predictor. This is called aliasing and it reduces the prediction accuracy. To increase the prediction accuracy, global schemes have been proposed.

Gshare: The intuition for this branch predictor is that the prediction of this branch depends on the most recently predicted branches.i.e. the prediction for a branch is dependent on the global history of branch predictions. In this predictor the lower order of the branch address xored with the global history is used to index the *branch history table* of 2-bit values. An entry of 0 or 1 in the branch history table predicts a "not taken" for the branch and an entry of 2 or 3 predicts a "taken".These 2-bit values are updated and used similar to the bimodel predictor. If a branch is "taken" then the counter is incremented saturating at 3 and if it is "not taken" then the counter is decremented saturating at 0.

YAGS: YAGS [3] is the acronym for "Yet Another Global Scheme". The motivation behind YAGS is the observation that for each branch we need to store its bias and the instances when it does not agree with its bias. It is implemented as a *choice table* and two *direction caches*, one each for "taken" and "not-taken". When a branch occurs in the instruction flow, the choice table is accessed. If the prediction indicates "taken", the not-taken direction cache is accessed to check if it is a special case where the prediction does not agree with the bias. If there is a miss in the "not-taken" cache, the choice table is used for prediction. If there is a hit in the "not-taken" cache it supplies the prediction. A similar set of actions is taken if the choice table indicates "not taken" but this time the check is done in the "taken" cache.

Both the choice tables and the direction caches have 2-bit entries which are updated as 2-bit counters saturating at 0 and 3. The choice table is updated whenever it is used to predict. It is also updated when the direction cache is used, but its prediction is wrong. The direction caches are updated when predictions

from them are used. They are also updated when the choice table is used to predict, but its prediction is wrong. i.e. for this particular branch the outcome is not the same as the bias stored in the choice table and hence we need to add this branch to the direction caches.
Meta-Predictor: The idea of the Meta-predictor is that different predictors predict better for certain branches. Hence, a scheme that combines predictors intelligently choosing between predictors will have a higher prediction accuracy. Meta-predictors proposed combine two predictors and choose between them. The *Meta-predictor table* is indexed with the lower order bits of the address and has 2-bit for each of them. An entry of 0 or 1 in the table indicates that the first predictor should be used to predict this branch and an entry of 2 or 3 indicates that the second predictor should be used to predict this branch.

This table is updated as follows. When both the predictors predict the branch with the same outcome no change is made to the entry in the table. If the first predictor mis-predicts and the second predictor predicts correctly the 2-bit counter is incremented saturating at 3. The 2-bit counter is decremented saturating at 0 if the first predictor predicts the branch correctly and the second predictor mis-predicts it.For this project a Meta-predictor has been designed by combining Gshare and YAGS.

3 Performance Study

3.1 Methodology

The four benchmarks that were used to test the performance of the **Branch Predictors** are:

- JPEG Encoder
- G721 Speech Encoder
- Mpeg Decoder
- Mpeg Encoder

Simplescalar [2] tool set was used to generate the branch trace files for each of these benchmarks. The sim-safe.c file was modified to generate the trace files. For each conditional branch there are two entries in the trace file. The first entry is the Program counter(PC) at which this branch instruction is present and the second entry is the outcome of this branch instruction. If the second entry is "1" this branch was "taken" and if it is "0" this branch was "not taken".

The four Branch Predictors described in Section 2 were modeled using C. The inputs required by the model are

- **Branch predictor:** This input indicates the branch predictor to be used from the four predictors that have been modeled. **"bi"**, **"gs"**, **"yg"**, **"mp"** indicate bimodel, gshare, yags and meta-predictor in that order.
- **Predictor Table size:** This input indicates the table size in KB to be used for this predictor. The valid inputs for table size are "1", "2", "4", "8", "16". The size of the table is the number of bits(in case of 1kb, it is 10 bits) used to index the branch history table (in bimodel and gshare) , the choice table (in YAGS) and meta-predictor table(for Meta-predictor).
- **Input trace file:** This is the trace file that is the input to the predictor. The entries in the trace file are as described above in this section.
- **Output file:** This is the output file into which the prediction results are written.

The executable developed for this project is called "branch_predictor.exe". An example for using this executable will be to run the following at the command prompt

#./branch_predictor.exe bi 1 trace_test.txt out.txt

This simulates a **bimodal** predictor of **1KB** size using **trace_test.txt** as the input trace file and writes the output into a file called **out.txt**

3.2 Results

Simulations were performed for the four modeled branch predictors using the four benchmarks. The results of the simulations are shown in the Figure.

Parts (a) to (d) show the performance of the different predictors for the four benchmarks with the increase in predictor size. Part (e) shows the performance of different predictors for the four benchmarks when the prediction table size is 8Kb. Part (f) shows the performance of different benchmarks with the increase of tag size in YAGS. Observations from the data are

- The prediction accuracy increases with the increase in predictor size and saturates.
- The global predictors such Gshare, YAGS and Meta-predictor perform better than the local bimodel predictor.
- The Meta-predictor predicts better than Bimodel, Gshare and YAGS.
- The prediction accuracy is dependent on the application. For e.g. The average prediction accuracy for the Mpeg decoder using any of the branch predictors is higher than the prediction accuracy for the other benchmarks.
- With the increase in tag-size for YAGS, the prediction accuracy increases and saturates with tag-size.

4 YAGS Neo Branch Predictor

An intuition that global predictors do better that local predictors, led me to design the YAGS Neo predictor using the YAGS predictor. Instead of using only the lower order bits of the branch address to index the choice table, I propose to use the lower order bits of the address xored with the global history to index the choice table. Incorporating this change into the YAGS code the following were simulated.

- The YAGS predictor as described in [3].
- The YAGS Neo predictor as described above.
- A Meta-predictor using YAGS and Gshare
- A Meta-predictor using YAGS Neo and Gshare
- A Meta-predictor using YAGS and YAGS Neo

The percentage of branches predicted correctly for each is shown in Figure 1.

4.1 Observations

From the data in Table 1 the following observations were made

- The YAGS Neo predictor gives a higher performance than YAGS for Jpeg and Speech Encoder.
- For Jpeg the Meta-predictor with YAGS and Gshare performs best.
- For Speech Encoder the Meta-predictor with YAGS and YAGS Neo performs best.
- For Mpeg Encoder the Meta-predictor with YAGS and Gshare performs best.
- For Mpeg Decoder the Meta-predictor with YAGS and YAGS Neo performs best.

Benchmark	Table Size	YG	YG Neo	YG-GS	YG Neo-GS	YG-YG Neo
Jpeg	1	89.938	90.138	90.452	90.383	90.227
	2	90.052	90.288	90.598	90.546	90.351
	4	90.208	90.425	90.744	90.685	90.488
	8	90.254	90.513	90.825	90.787	90.547
	16	90.312	90.589	90.907	90.868	90.613
Speech Encoder	1	91.347	91.706	91.439	91.720	91.813
	2	91.652	91.948	91.870	91.978	92.068
	4	91.982	92.024	92.077	92.060	92.087
	8	92.003	92.065	92.092	92.083	92.107
	16	92.086	92.119	92.160	92.137	92.150
Mpeg Encoder	1	79.763	79.713	81.417	80.077	81.165
	2	80.539	80.348	81.891	80.693	81.650
	4	80.955	80.773	82.184	81.102	81.955
	8	81.253	81.088	82.387	81.428	82.173
	16	81.478	81.319	82.539	81.655	82.312
Mpeg Decoder	1	95.805	95.7	95.645	95.543	95.825
	2	95.89	95.843	95.738	95.686	95.909
	4	95.918	95.839	95.759	95.717	95.937
	8	95.929	95.851	95.776	95.743	95.948
	16	95.939	95.949	95.848	95.817	95.988

Figure 1: Simulation Results

4.2 Summary

The results from simulations for the YAGS Neo branch predictor show that there is an improved performance for some applications. Even 0.1% increase in performance could be valuable for Bracnh Predictors as the typical number of branches in an application is of the order of millions. For e.g. The total number of branches in the *Mpeg Decoder* benchmark used for this project was 19675743. So, a 0.1% increase would result in ∼20,000 more branches being predicted correctly.

References

[1] http://en.wikipedia.org/wiki/Branch˙prediction.

[2] Doug Burger, Todd M. Austin, and Steve Bennett. Evaluating future microprocessors: The simplescalar tool set. Technical Report CS-TR-1996-1308, 1996. citeseer.ist.psu.edu/burger96evaluating.html.

[3] A. N. Eden and T. Mudge. The yags branch prediction scheme. In *MICRO 31: Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, pages 69–77, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[4] Scott McFarling. Combining branch predictors. Technical Report TN-36, June 1993. citeseer.ist.psu.edu/mcfarling93combining.html.