A Comprehensive Analysis of Indirect Branch Prediction

Oliverio J. Santana[1], Ayose Falcón[1], Enrique Fernández[2], Pedro Medina[2], Alex Ramírez[1], and Mateo Valero[1]

[1] Dpto. de Arquitectura de Computadores, Universidad Politécnica de Cataluña
{osantana,afalcon,aramirez,mateo}@ac.upc.es
[2] Dpto. de Informática y Sistemas, Universidad de Las Palmas de Gran Canaria
{efernandez,pmedina}@dis.ulpgc.es

Abstract. Indirect branch prediction is a performance limiting factor for current computer systems, preventing superscalar processors from exploiting the available ILP. Indirect branches are responsible for 55.7% of mispredictions in our benchmark set, although they only stand for 15.5% of dynamic branches. Moreover, a 10.8% average IPC speedup is achievable by perfectly predicting all indirect branches.
The Multi-Stage Cascaded Predictor (MSCP) is a mechanism proposed for improving indirect branch prediction. In this paper, we show that a MSCP can replace a BTB and accurately predict the target address of both indirect and non-indirect branches. We do a detailed analysis of MSCP behavior and evaluate it in a realistic setup, showing that a 5.7% average IPC speedup is achievable.

Keywords: microarchitecture, branch prediction, Branch Target Buffer, indirect branch, Multi-Stage Cascaded Predictor

1 Introduction

Current computer systems rely on instruction level parallelism (ILP) to achieve high performance. Superscalar processors are designed to exploit ILP, but control hazards prevent them from taking advantage of all the available ILP. One of the most used mechanisms to overcome control hazards is branch prediction. A decoupled branch prediction architecture [2] uses a direction predictor to predict if conditional branches will be taken or not taken and a target address predictor to predict where a taken branch will go. There is a wide amount of research about improving the accuracy of branch direction predictors [15,9,10], but the use of a branch target buffer (BTB) is commonly accepted as a good way of predicting target addresses [8,16,12]. However, some authors [7,1,3,4] claim that the target addresses of indirect branches are not accurately predicted by a simple BTB.

In this paper, we show that indirect branch prediction is a hard problem for computer systems and that there is room for improvement. In our benchmark set, branch instructions are correctly predicted 91.2% of the time on average.

If we do not take into account return instructions, indirect branches are only correctly predicted 45.3% of the time. Overall, indirect branches are responsible for 55.7% of all branch mispredictions, although they only stand for 15.5% of dynamic branch instructions. A 10.8% average IPC speedup can be obtained by perfectly predicting all indirect branches.

The Multi-Stage Cascaded Predictor (MSCP) is a mechanism described by Driesen et al. [6] for improving the prediction of indirect branches. In their previous study, they evaluate MSCP behavior using only indirect branch traces. We contribute to the study of this technique by showing that MSCP can work well with complete programs, replacing the BTB. Using a MSCP to predict the target address of branch instructions provides an increase in target address prediction accuracy, as well as an improvement in processor performance. This improvement indicates that MSCP can predict accurately target addresses of both indirect and non-indirect branches.

We do a detailed analysis of MSCP behavior in order to understand how it works. MSCP improves processor performance by providing an important reduction in mispredictions due to indirect branches. We show that, in a three level configuration, the intermediate level is poorly used. Most branches, whose target addresses are easy to predict, use the first level, while difficult to predict branches are better predicted using the last level.

This deep analysis of MSCP allows us to find better configurations and to optimize its performance in a realistic setup. Using a *gskewed* branch direction predictor [10] and replacing the BTB with a MSCP of an equivalent hardware budget we achieve 5.7% average IPC speedup. We show that this speedup cannot be achieved by a larger BTB. Therefore, problems in target address prediction in studied benchmarks are not due to conflict misses, but to the unpredictability of indirect branches using a simple BTB.

We also analyze the behavior of some static indirect branches in our integer benchmark suite. There is a small amount of static indirect branches that are responsible for a great number of mispredictions in some benchmarks. These branches are indirect jumps in high level switch structures and function calls using pointers. Therefore, not only object oriented programs will find beneficial an improvement in indirect branch prediction. In addition, we show that a difficult to predict indirect branch not necessarily should have a lot of different dynamic target addresses. The predictability of an indirect branch in MSCP depends on whether or not there is correlation of its target address with the target address of previously executed branches.

The remainder of this paper is organized as follows. Section 2 exposes previous related work. In section 3 we describe our simulation environment and the selected benchmark set. In section 4 we show the relative importance of indirect branches with regards to the rest of branch instructions. In section 5 we describe the MSCP and analyze its behavior, proposing a realistic configuration. In section 6 we study the behavior of some static indirect branches. Finally, section 7 exposes our concluding remarks.

2 Related Work

One of the best known approaches to indirect branch prediction is the return address stack (RAS) proposed by Kaeli et al. [7]. In [1] Calder et al. show that indirect branches will be a limiting factor to performance with the increase of popularity of object oriented programming. They analyze some static and dynamic techniques for improving the prediction of indirect function calls. We show in this paper that better predicting indirect branches is important not only for object oriented programs.

In [11] Nair proposes to record the path of target addresses of recent branches in the history register instead of a pattern of bits meaning taken or not taken branches. This technique is used for indexing the prediction tables of MSCP. In [3] Chang et al. propose a target cache to predict the target address of indirect branches. This mechanism has a prediction table indexed by a value obtained hashing the branch instruction address with the contents of a history register. Stark et al. present in [14] a mechanism which index its prediction table using different history lengths for each branch, according to previously collected profile information.

The main source of inspiration of this paper is the previous work of K. Driesen and U. Hölzle in indirect branch prediction. In [4] they explore a variety of two-level predictor configurations dedicated to predict the target address of indirect branches. In [5] Driesen et al. propose the cascaded predictor as a new way of combining two prediction tables. This predictor keeps easy to predict branches in a first level table and allows difficult to predict branches, i.e. indirect branches, to use a second level table, which takes advantage of correlation for keeping more information. The Multi-Stage Cascaded Predictor is a generalization of this technique proposed in [6]. We contribute to the study of this mechanism by showing that it is able to replace a BTB and accurately predict the target address of both indirect and non-indirect branches. We do a detailed analysis of its internal behavior and evaluate its impact on processor performance presenting IPC measures.

3 Experimental Methodology

Our data has been obtained using the *sim-outorder* simulator from the *SimpleScalar 3.0 tool set*. This simulator, configured to use the Alpha instruction set architecture, models a six stage pipeline with a register update unit (RUU). We have collected detailed information about static branch instructions from *sim-outorder*. The baseline configuration is a 4-way issue processor described in table 1. This configuration uses a 2048 entry 4-way associative BTB for target address prediction and a 64 entry RAS for predicting return instructions. A *gskewed* predictor [10] is used to predict the direction of conditional branches. It is a 12 KB predictor with three 16K entry tables and a history length of 14 bits.

We have selected the nine programs from the SPECint 95 and 2000 benchmarks where indirect branch prediction is a harder problem. These benchmarks

Table 1. Configuration of the baseline processor

fetch width	4
issue width	4
RUU entries	64
load/store queue entries	32
integer ALUs	4
floating point ALUs	4
integer multiplier/dividers	1
floating point multiplier/dividers	1
first level instruction cache	32 KB, 2-way associative, block size: 32 B
first level data cache	32 KB, 2-way associative, block size: 32 B
second level unified i+d cache	1 MB, 4-way associative, block size: 64 B

allow us to measure MSCP performance with complete programs instead of using only indirect branch traces. They also allow us to show that not only object oriented programs would find MSCP beneficial. The selected benchmarks were compiled with the Compaq C V5.8-015 compiler on Compaq UNIX V4.0 with options *-O2*, *-g3* and *-non_shared*. SPECint 95 benchmarks were executed to completion using a modified version of the *test* or *train* input set. SPECint 2000 benchmarks were executed to completion or until 500 million instructions were executed. SPECint 2000 used the *test* input set, excluding *253.perlbmk*, which used the *train* input set.

4 The Importance of Indirect Branches

In this section we show the importance of indirect branches with regards to the rest of branch instructions. By perfectly predicting all indirect branches, we get a 10.8% average IPC speedup over our baseline configuration, getting 20% in *253.perlbmk* or even 31% in *134.perl*. This data shows the potential improvement a mechanism for better predicting indirect branches can achieve and that such a mechanism will be worthwhile.

Figure 1 shows the prediction accuracy of branch instructions. The first bar represents prediction accuracy for indirect branches excluding return instructions, which are already correctly predicted by a RAS [7]. The second bar represents prediction accuracy for all branch instructions. While the average branch prediction accuracy is 91.2%, indirect branches are correctly predicted only 45.3% of the time. This low prediction accuracy is caused by the BTB, since indirect branches are always taken in the Alpha instruction set architecture. Therefore, a BTB cannot predict accurately the target address of indirect branches.

In figure 2 we show the percentage of indirect branches over the total number of dynamic branch instructions and the percentage of all branch mispredictions due to indirect branches. The first bar indicates that indirect branches are a small percentage of branch instructions in our benchmark set. However, the second bar

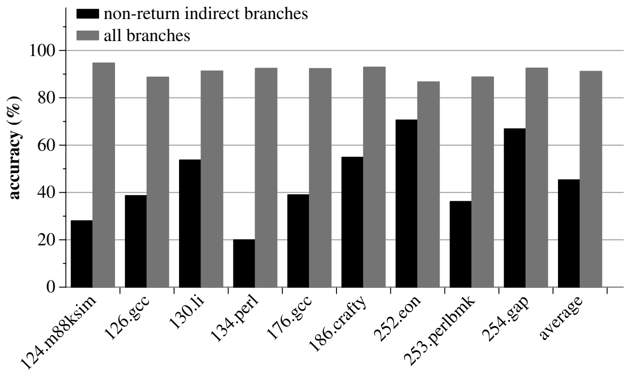

Fig. 1. Comparison of prediction accuracy for indirect branches, excluding returns, and all branches

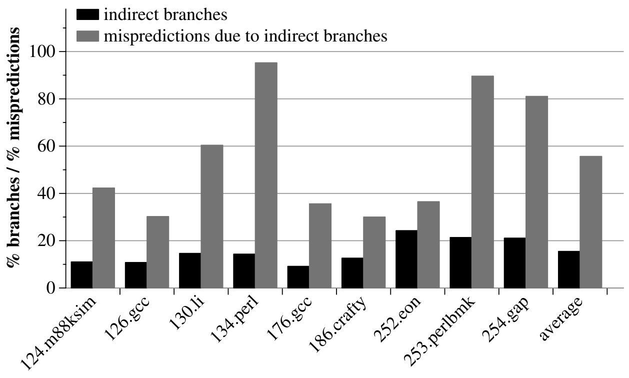

Fig. 2. Comparison of the percentage of indirect branches over the total number of dynamic branch instructions and the percentage of all branch mispredictions due to indirect branches

points out that indirect branches are responsible for most branch mispredictions. On average, indirect branches represent only 15.5% of dynamic branch instructions, although they are responsible for 55.7% of all mispredictions. This makes clear that, although research has been focused on branch direction predictors, better ways of predicting indirect branches should be developed.

5 Analysis of Multi-stage Cascaded Predictor

In this section we analyze and evaluate the Multi-Stage Cascaded Predictor. A more detailed study can be found in [13]. First of all, we do a description of MSCP in section 5.1. In section 5.2 we present an analysis of MSCP behavior and performance measures using IPC. We show the influence of MSCP in indirect branch prediction and how different table levels are used. Finally, we evaluate

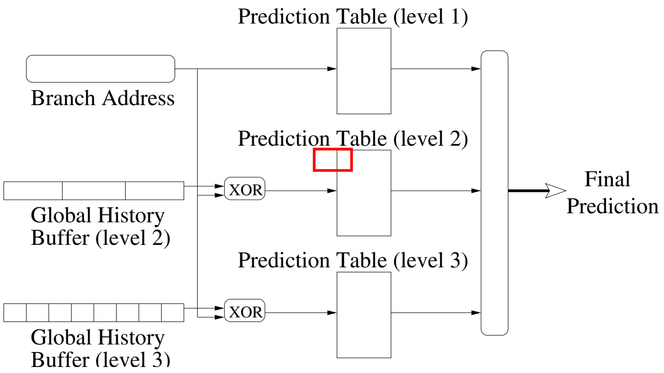
Prediction Table (level 1)
Branch Address
Prediction Table (level 2)
XOR
Global History Buffer (level 2)
Prediction Table (level 3)
XOR
Global History Buffer (level 3)
Final Prediction

Fig. 3. Multi-Stage Cascaded Predictor

MSCP in section 5.3, showing that this mechanism can outperform a BTB with the same hardware budget in a realistic setup.

5.1 Description

The Multi-Stage Cascaded Predictor, shown in figure 3, is designed to predict the target address of branch instructions. Therefore, this predictor replaces the BTB in a decoupled branch prediction architecture [2]. MSCP consists on three prediction tables that make their prediction in parallel. Each prediction table is indexed by *xoring* the branch instruction PC with the contents of a history register. This history is built using the target address of recently executed branches.

The number of branches used to build the history is called path length. The higher the table level is, the larger its path length should be. First level table uses a zero path length, so it is indexed only with the branch instruction PC and behaves like a BTB. MSCP chooses the prediction from the higher level which has information about the branch being predicted. Higher levels are supposed to be more accurate because they keep more information about each branch by using correlation with a larger path length.

To work well, MSCP should keep easy to predict branches in the first level table while only difficult ones should be allowed to use higher level tables. A new branch, that is, a branch which cannot be predicted because MSCP has no information about it, is always introduced in the first level table. Only if the prediction for later instances of that branch fails repeatedly it will promote to higher level tables.

5.2 Behavior Analysis

In order to study the behavior of MSCP we evaluate a configuration with three prediction tables. Each one is a 2048 entry 4-way associative table. Therefore, this configuration has 6K entries, three times more than the baseline BTB. The path length is three branches for the second level and eight branches for the third level, as in [6].

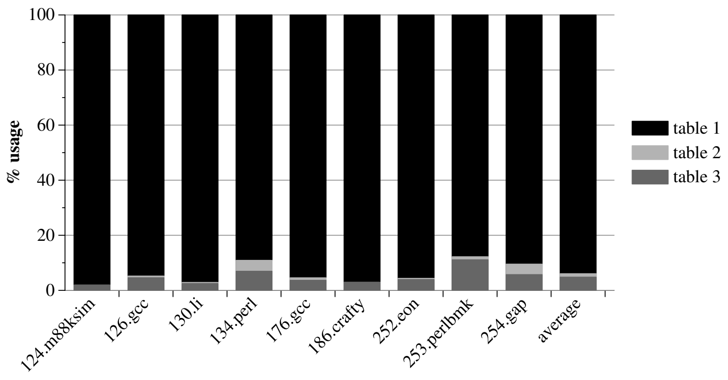

Fig. 4. Percentage of usage of each table prediction in MSCP over the whole number of predictions done

The size of this MSCP configuration is large enough to analyze its behavior without resource limitations. In this section we use a perfect branch direction predictor to ensure that all branch mispredictions are actually target address mispredictions. With this approach, all non-indirect branches are easily predictable, so we can focus on the predictability problems of indirect branches.

We obtain a 5.7% average IPC speedup along our benchmark set by replacing the baseline BTB with the analyzed setup of MSCP. The benchmark *134.perl* even achieves a 21.5% IPC speedup. MSCP gets an average reduction of 55% of this kind of mispredictions, achieving even 87% in *134.perl* or 93% in *254.gap*. This data shows that MSCP improves processor performance by better predicting indirect branches.

Behavior of Each MSCP Prediction Table. To better understand how MSCP works, we analyze the use of each table. In figure 4 we show the percentage of usage of each table prediction over the total number of predictions done. Most predictions have been done by the first level table. This means that a majority of branches can be easily predicted by a simple BTB without correlation and do not need to be upgraded toward higher levels. The third level table is the one that predicts difficult branches, while the second is rarely used.

Almost all target mispredictions are caused by predictions done by the third level table. This happens because predictable branches remain in the first level while only difficult branches are promoted to the third level table. Therefore, the first and the second level tables will only predict branches that they are able to predict, while the third level table will predict not only the branches it is able to predict, but also those branches which cannot be predicted by any table.

Figure 5 shows the percentage of indirect branch predictions done by each table over the total number of indirect branch predictions. Most indirect branch predictions are done by the third level table because indirect branches are supposed to be too difficult for the first level table to predict them. However, there is a small amount of indirect branches in some benchmarks that can be correctly

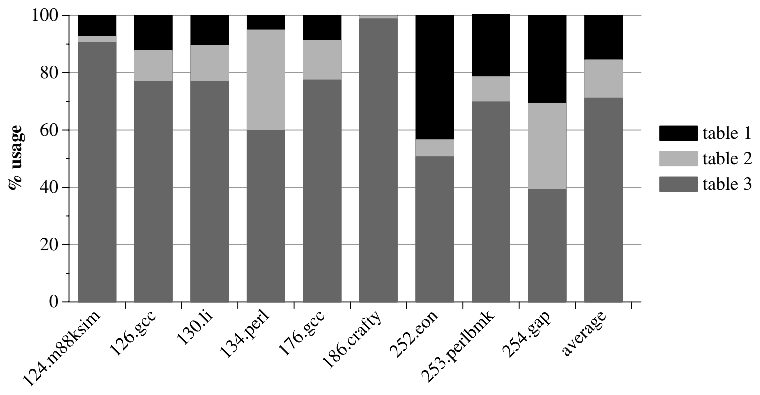

Fig. 5. Percentage of indirect branch predictions done by each table

predicted by the first level table. This happens to indirect branches having only a single target address or a target address that repeats in consecutive instances of the branch. In addition, we can observe that the second level table is poorly used in comparison with the third level one in the majority of benchmarks. This data points out that MSCP would work well using only two levels of prediction tables.

5.3 Evaluation

In the previous section we analyzed a MSCP three times larger than the baseline BTB. We explore now some possible configurations of MSCP using approximately the same hardware budget than the baseline BTB, that is, 2048 entries. We are not taking into account the little cost of global history registers and associated logic.

Figure 6 shows IPC speedup of some MSCP configurations over the baseline using a perfect branch direction predictor. We have simulated a three level MSCP, labeled as *3t* in figure 6. This configuration has a 1024 entry 2-way associative table in the first level and a 512 entry direct mapped table in both the second and the third level. Path length three is used in the second level and path length eight is used in the third level. We have also simulated three two level MSCP configurations, labeled as *2t3*, *2t8* and *2t9*. These configurations have a 1536 entry 3-way associative table in the first level and a 512 entry direct mapped table in the second level. The second level is indexed using path length three (*2t3*), eight (*2t8*) and nine (*2t9*) branches.

The best MSCP configuration shown in figure 6 is the one with two levels using a path length of nine branches to index second level prediction table. This configuration achieves a 6.4% average IPC speedup using 2048 entries while a 6K entry one achieves only a 5.7% average IPC speedup, as we said in section 5.2. This happens because 2048 entries is a size big enough and a larger path length is more beneficial than bigger prediction tables. We can also see that the three

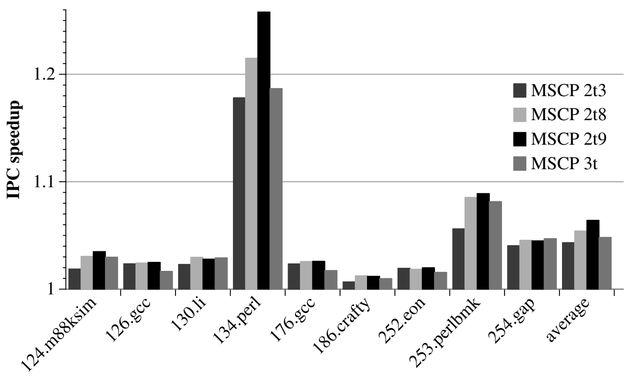

Fig. 6. Some 2048 entry MSCP configurations IPC speedup over baseline configuration

level MSCP performance is worse than the two level configurations with path length eight and nine. The first level table is smaller in the three level MSCP than in the two level one. This fact causes more conflict misses in the three level MSCP.

We have also evaluated a 4096 entry 4-way associative BTB, twice the size of the baseline BTB. This larger BTB achieves very little speedup over the baseline, since a 2048 entry BTB is big enough to avoid the majority of conflict misses. Therefore, using MSCP is a better option than increasing the size of the BTB.

Realistic Branch Direction Predictor. All previous data in this section has been obtained using a perfect branch direction predictor. Next, we show that MSCP outperforms the 2048 entry 4-way associative baseline BTB using a realistic branch direction predictor. Figure 7 shows IPC speedup for the best 2048 entry MSCP configuration described above using a 12 KB *gskewed* branch direction predictor. This speedup is measured over the baseline BTB using the same *gskewed* predictor. This data is compared with speedup obtained perfectly predicting all indirect branches. As we can see, this MSCP configuration achieves 5.7% average IPC speedup with regards to 10.8% average potential IPC speedup. We should highlight that *134.perl* benchmark is achieving almost all its potential speedup, but there is still room for improvement in the remaining benchmarks.

Besides, we have simulated a path correlated target address predictor, that is, a 2048 entry 4-way associative table managed in the same way as the second level of MSCP and using a path length of nine branches. This mechanism is similar to the target cache proposed by Chang et al. [3] but applied to all branches instead of only indirect branches. Figure 7 also shows that the path correlated BTB only achieves a 4.3% average IPC speedup and even loses performance with regards to the baseline configuration in three benchmarks. This effect is clear in *126.gcc* and *176.gcc* due to conflict misses caused by their high number of static branches. The first level table of MSCP prevents easy to predict branches from using path correlation, avoiding aliasing; this not happens in the path correlated BTB.

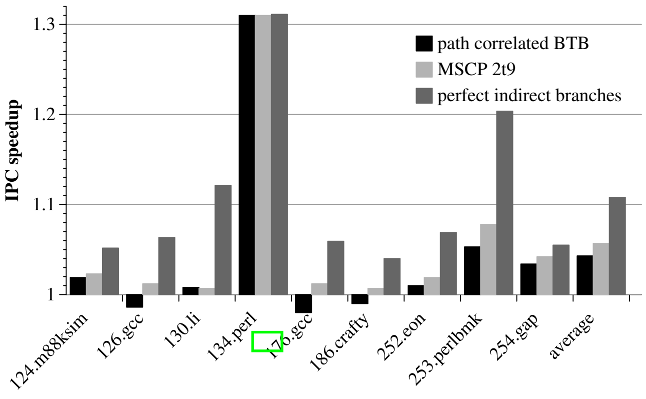

Fig. 7. Comparison of the performance of the best MSCP 2048 entry configuration, using a *gskewed* branch direction predictor, with a perfect indirect branch predictor and a path correlated BTB

6 Analysis of Static Branches

In this section we analyze the behavior of some static indirect branches. We have found [13] that a little number of static branches are responsible for almost all indirect jump and indirect function call mispredictions in the majority of benchmarks. For example, only three branches are responsible for 99% of mispredictions due to indirect jumps in *124.m88ksim*. Only one branch is responsible for 99% of mispredictions due to indirect function calls in *254.gap*.

We have selected 10 static indirect branches for a more detailed analysis. Table 2 shows information collected about these branches; each of the 10 entries in the table corresponds to one of them. The first column identifies the branch and the second one shows if it is an indirect jump or a function call. All studied indirect jumps correspond to high level switch structures and all studied indirect function calls correspond to calls using a pointer to the called function [13]. The next four columns have the percentage of all mispredictions caused by the branch, the number of different dynamic targets found for the branch, the percentage of time the branch went to a different target address than in its latest execution and the reduction in mispredictions achieved using MSCP.

Data of the branch selected from the benchmark *252.eon*, which is written in C++, points out that we can expect a good performance of MSCP with object oriented programs. The number of mispredictions for this branch is reduced in a 62.5% by using MSCP. Besides, a reduction of almost all mispredictions in the three indirect branches studied in *134.perl* is achieved, so MSCP can also be beneficial for not object oriented programs.

The number of different targets where an indirect branch can go after execution is not necessarily related with its predictability or unpredictability. Although some difficult to predict branches have a lot of different dynamic targets, as the branch selected from *253.perlbmk* which has 43 targets, there are other difficult branches with only three or four different targets. However, these branches have

Table 2. Data of the 10 selected static indirect branches

	type	**misp**	**tgt**	**chn**	**red**
124.m88ksim, file dpath.c, line 444	jump	8.8%	28	81.2%	24.7%
124.m88ksim, file stats.c, line 83	jump	8.5%	9	78.2%	75.6%
130.li, file xldmem.c, line 446	jump	4.0%	3	36.5%	40.3%
130.li, file xleval.c, line 106	call	3.3%	22	83.5%	21.5%
134.perl, file eval.c, line 137	jump	36.1%	7	80%	99.9%
134.perl, file eval.c, line 450	jump	27.1%	12	99.9%	99.9%
134.perl, file cmd.c, line 224	jump	9.0%	5	66.6%	99.9%
186.crafty, file swap.c, line 165	jump	4.0%	8	92.9%	48.3%
252.eon, file mrMaterial.cc, line 50	call	5.49%	4	79.0%	62.5%
253.perlbmk, file run.c, line 30	call	41.2%	43	94.2%	55.4%

a high percentage of executions in which their target address is different than the one observed in their latest execution. Therefore, the predictability of an indirect branch depends not on the number of different targets it has but on the predictability of the sequence of changes in its target address. More research should be done in order to analyze the path of different targets that an indirect branch follows and how we can better predict it.

7 Conclusions

In this paper we have shown that indirect branches are an important limiting factor for computer systems. Indirect branches are responsible for 55.7% of all mispredictions on average along our benchmark set, although they represent only 15.5% of dynamic branch instructions. Besides, indirect branches, excluding return instructions, have a prediction accuracy of 45.3% while the average prediction accuracy for all branch instructions is 91.2%. Taking into account this data, it is not surprising that a 10.8% IPC speedup is achievable by perfectly predicting all indirect branches.

We contribute to the study of Multi-Stage Cascaded Predictor [6] by showing that it can replace a BTB and accurately predict the target address of all branch instructions, improving processor performance. This improvement is due to an important reduction in indirect branch mispredictions achieved by MSCP. We have done a detailed analysis of MSCP behavior. We show that the majority of MSCP predictions are done by first level table, since most branches are easy to predict by a simple BTB. Difficult to predict branches are frequently upgraded up to the third level table, so second level one is poorly used.

We have found that the best MSCP configuration is one with only two prediction tables. This configuration has a 1536 entry 3-way associative first level table and a 512 entry direct mapped second level table. Second level is indexed using correlation with the target address of the latest nine executed branches. In a realistic setup, using a *gskewed* conditional branch direction predictor, a 5.7% IPC speedup is obtained over a BTB with the same hardware budget.

We have also shown that integer benchmarks can take advantage of better predicting indirect branches. These benchmarks do not have as much indirect branches as object oriented programs. However, only a little number of static indirect branches can harm the processor performance in such a way that techniques to improve their prediction will be worthwhile. Improving the prediction of indirect branches will be beneficial for programs that call functions using pointers, like *253.perlbmk*, or which have some high level switch structures depending on input data, like *gcc* compiler. Nevertheless, indirect branch prediction would be even a more important topic in computer architecture since object oriented programs, like *252.eon*, are becoming more popular.

Acknowledgements

This research has been supported by CICYT grant TIC-2001-0995-C02-01, CEPBA and an Intel scholarship grant. O. J. Santana is also supported by Generalitat de Catalunya grant 2001FI-00724-APTIND. A. Falcón is also supported by Ministry of Education of Spain grant AP2000-3923. E. Fernández is supported by Gobierno de Canarias. The authors would like to thank Jesús Corbal and Fernando Latorre for their valuable comments, as well as Carlos Navarro for his previous work in the simulation tool.

References

1. B. Calder and D. Grunwald. Reducing indirect function call overhead in C++ programs. *21st Symp. on Principles of Programming Languages*, 1994.
2. B. Calder and D. Grunwald. Fast & accurate instruction fetch and branch prediction. *21st Intl. Symp. on Computer Architecture*, 1994.
3. P. Y. Chang, E. Hao and Y. Patt. Target prediction for indirect jumps. *24th Intl. Symp. on Computer Architecture*, 1997.
4. K. Driesen and U. Hölzle. Accurate indirect branch prediction. *25th Intl. Symp. on Computer Architecture*, 1998.
5. K. Driesen and U. Hölzle. The cascaded predictor: economical and adaptive branch target prediction. *31st Intl. Symp. on Microarchitecture*, 1998.
6. K. Driesen and U. Hölzle. Multi-Stage Cascaded Prediction. *5th Intl. Euro-Par Conf.*, 1999
7. D. Kaeli and P. Emma. Branch history table prediction of moving target branches due to subroutine returns. *18th Intl. Symp. on Computer Architecture*, 1991.
8. J. Lee and A. Smith. Branch prediction strategies and branch target buffer design. *IEEE Computer Magazine, 17(1)*, 1984.
9. S. McFarling Combining branch predictors. *Digital Equipment Corporation, WRL Technical Note TN-36*, 1993.
10. P. Michaud, A. Seznec and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. *24th Intl. Symp. on Computer Architecture*, 1997.
11. R. Nair. Dynamic path-based branch correlation. *28th Intl. Symp. on Microarchitecture*, 1995.
12. C. Perleberg and A. Smith. Branch target buffer design and optimization. *IEEE Transactions on Computers, 42(4)*, 1993.

13. O. J. Santana, A. Falcón, E. Fernández, P. Medina, A. Ramírez and M. Valero. Analysis and evaluation of the Multi-Stage Cascaded Predictor. *Departamento de Arquitectura de Computadores, UPC, Technical Report DAC-UPC-2001-24*, 2001.
14. J. Stark, M. Evers and Y. Patt. Variable length path branch prediction. *8th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 1998.
15. T. Y. Yeh and Y. Patt. Two level adaptive training branch prediction. *24th Intl. Symp. on Microarchitecture*, 1991.
16. T. Y. Yeh and Y. Patt. A comprehensive instruction fetch mechanism for a processor supporting speculative execution. *25th Intl. Symp. on Microarchitecture*, 1995