Independent hashing as confidence mechanism for value predictors in microprocessors

Veerle Desmet, Bart Goeman, and Koen De Bosschere

Department of Electronics and Information Systems, Ghent University
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
E-mail:{vdesmet,bgoeman,kdb}@elis.rug.ac.be

Abstract. Value prediction is used for overcoming the performance barrier of instruction-level parallelism imposed by data dependencies. Correct predictions allow dependent instructions to be executed earlier. On the other hand mispredictions affect the performance due to a penalty for undoing the speculation meanwhile consuming processor resources that can be used better by non-speculative instructions. A confidence mechanism performs speculation control by limiting the predictions to those that are likely to be correct.
When designing a value predictor, hashing functions are useful for compactly representing prediction information but suffer from collisions or *hash-aliasing*. This hash-aliasing turns out to account for many mispredictions. Our new confidence mechanism has its origin in detecting these aliasing cases through a second, independent, hashing function. Several mispredictions can be avoided by not using predictions suffering from hash-aliasing.
Using simulations we show a significant improvement in confidence estimation over known confidence mechanisms, whereas no additional hardware is needed. The combination of independent hashing with saturating counters performs better than pattern recognition, the best confidence mechanism in literature, and it does not need profiling.

1 Introduction

Nowadays computer architects are using every opportunity to increase the IPC, the average number of Instructions executed Per Cycle. The upper bound on achievable IPC is generally imposed by data dependencies. To overcome these data dependencies, the outcome of instructions is predicted such that dependent instructions can be executed in parallel using this prediction.

As correct program behaviour has to be guaranteed, mispredictions require recovery techniques to undo the speculation by restarting the execution from a previous processor state. This recovery takes some cycles and therefore every predictor design tries to avoid mispredictions. To further prevent mispredictions, applying selective prediction [3] or predicting only for a subset of instructions is recommended as over 40% of predictions made may not be useful in enhancing performance [8]. The selection of appropriate predictions can be done by using

a confidence mechanism based upon history information. Common techniques include saturating counters and pattern recognition [2].

We propose a new confidence mechanism for prediction schemes using a hashing function. Specifically for these predictors many mispredictions occur due to hashing: collisions or hash-aliasing occurs when different unhashed elements are mapped on a same hashing value. Our confidence mechanism tries detecting this interaction by using a second hashing function, independent on the original one. If hash-aliasing is detected the corresponding prediction will be ignored resulting in higher prediction accuracies. We evaluate it for a Sazeides predictor [10], the most accurate non-hybrid [11] value predictor known today.

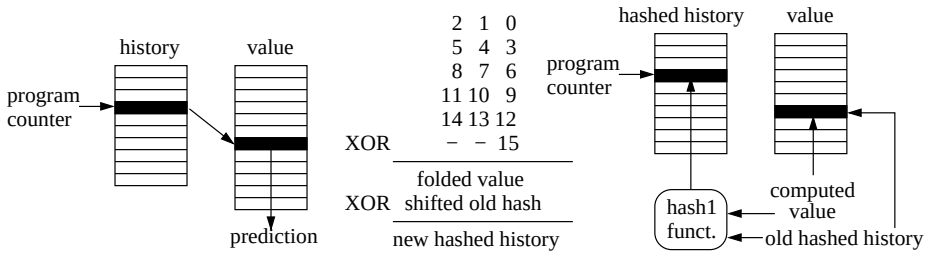
This paper starts with an introduction to value prediction and the problem of aliasing. In section 3 we discuss the need for a confidence mechanism, explain previously proposed confidence mechanisms and describe metrics for comparing different confidence mechanisms. The use of an independent hashing function for detecting hash-aliasing is introduced in section 4. In section 5 we evaluate our independent hashing mechanism. Section 6 summarises the main conclusions.

2 Value prediction

Most instructions need the outcome of preceding instructions and therefore have to wait until the latter are finished before their execution can be started. These so-called data dependencies can be eliminated by predicting the outcome of instructions so that dependent instructions can be executed earlier using the predicted value. This predicted value is simply referred as *prediction* to distinguish it from the *computed value* which verifies the prediction. The prediction is made during fetching while the computed value is available when the execution is completed. In case of a misprediction the speculation has to be undone according to a recovery policy: re-fetching or selective recovery. *Re-fetching* is used in branch prediction and involves all instructions following the misprediction to be re-fetched. This is a very costly operation and makes high prediction accuracies necessary. It is however easy to implement since the branch recovery hardware can be reused. *Selective recovery* only re-executes those instructions that depend on the misprediction resulting in lower misprediction penalties but requires additional hardware to keep track of dependency chains.

2.1 FCM predictor

The finite context method (FCM) is a context-based prediction scheme [7] using recently computed values, called *history*, to determine the next prediction. The number of computed values forming the history is the *order* of the prediction scheme. One of the most accurate FCM predictors was introduced by Sazeides [10] and the actions taken during prediction are illustrated in Figure 1(a). Two prediction tables are managed. The first one is the *history table* and is indexed by the program counter. It contains the history, which is hashed in order to reduce the total number of bits to store. The hashed history is then used as an



(a) Predicting (b) Hashing (c) Updating

Fig. 1. FCM predictor: Sazeides

index in the *value table*, where the prediction is found. An accuracy up to 78% (order 3) is reached by the Sazeides predictor when using infinite tables [10].

Throughout this paper, the history is hashed according to Sazeides' FS R-5 hashing function because it provides high prediction accuracy for a wide range of predictor configurations [9]. This hashing function incrementally calculates a new hashed history, by only using the old hashed history and the computed value to add to it. For a value table of 2^b entries we need a hashing function that maps the history consisting of *order* values to b bits. The construction is illustrated in Figure 1(b) for a 16-bit computed value and $b = 3$. The computed value is folded by splitting into sequences of b consecutive bits and combining these sequences by XORing. According to the definition of the order, we shift the old hashed history over $\lceil \frac{b}{order} \rceil$ bits. By XORing the shifted old hashed history and the folded value we obtain the new hashed history.

All actions that have to be taken during update, i.e. when the computed value is known, are shown in Figure 1(c). They include storing the computed value in the entry pointed by the old hashed history, calculating the new hashed history and storing it in the history table.

2.2 Instruction aliasing

The discussed prediction scheme uses the program counter to index the history table. Using infinite tables every instruction has its own table entry. For finite tables however, only part of the program counter is used as an index resulting in many instructions sharing the same entry, which is called *instruction aliasing*. Although the interaction between instructions could be constructive, it is mostly destructive [4].

3 Confidence mechanism

Basically, a value predictor is capable to make a prediction for each instruction. However, sometimes the prediction tables do not contain the necessary infor-

mation to make a correct prediction. In such a case, it is better not to use the prediction because mispredictions incur a penalty for undoing the speculation, whereas making no prediction does not. From this point, we can influence the prediction accuracy of value predictors by selectively ignoring some predictions. To perform this selection we associate to each prediction a degree of reliability or *confidence*. Along this gradation a confidence mechanism assigns *high confidence* or *low confidence* such that assigning high confidence goes together with little chance of making a misprediction. High-confident predictions will be used whereas for low-confident ones it behaves as if no value predictor is available. Confidence mechanisms are based on *confidence information*, stored in the prediction tables together with the prediction. We first describe saturating counters and patterns as types of confidence information and we then explain how different confidence mechanisms can be compared.

3.1 Saturating counters

A *saturating counter* directly represents the confidence of the corresponding prediction [6]. If the counter value is lower than a certain *threshold* the prediction is assigned low confidence, otherwise high confidence. The higher the threshold, the stronger the confidence mechanism. Regardless of the assigned confidence, the counter is updated at the moment the computed value is known. For this update we increment the counter (e.g. by one) for a correctly predictable value saturating at the maximum counter value and decrement (e.g. by one) the counter down to zero if the value was not correctly predictable. This way a saturating counter is a metric for the prediction accuracy in the recent past.

3.2 Patterns

Pattern recognition as proposed in [2] is based on *prediction outcome histories* keeping track of the outcome of the last value predictions. To identify the predictable history patterns, a wide range of programs are profiled (i.e. looking at their behaviour). Patterns precisely represent the recent history of prediction outcomes and do not suffer from saturating effects. Typically, patterns require more confidence bits than saturating counters and perform slightly better.

3.3 Metrics for comparing confidence mechanisms

For a given value predictor the predictions are divided into two classes: predictions that are correct and those that are not. Confidence assignment does not change this classification. The improvement of adding a confidence mechanism is thus limited by the power of the underlying value predictor. For each prediction, the confidence mechanism distinguishes high-confident predictions from low-confident ones. Bringing together the previous considerations we categorise each prediction in one of the quadrants shown in Figure 2 [5]. A perfect confidence mechanism only puts predictions into classes HCcorr and LCntcorr. In a

	Correctly predictable	Not correctly predictable
High Confidence	HCcorr	HCntcorr
Low Confidence	LCcorr	LCntcorr

Fig. 2. Classification of predictions

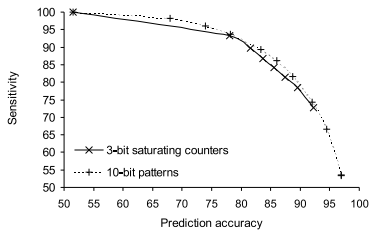


Fig. 3. Counters and patterns

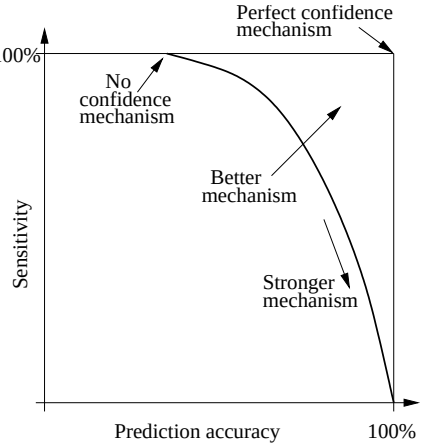


Fig. 4. Sensitivity versus prediction accuracy

realistic situation all quadrants are populated even the classes LCcorr and HCntcorr. We note that these 'bad' classes are not equivalent because the impact of a misprediction is usually different from that of missing a correct prediction.

We now describe a way for comparing different confidence strategies against the same value predictor without fixing the architecture as proposed in [5] for comparing confidence mechanisms in branch predictors. We will use the following independent metrics which are both "higher-is-better": *prediction accuracy* representing the probability that a high confidence prediction is correct and *sensitivity* being the fraction of correct predictions identified as high confidence.

$$\textbf{Prediction accuracy} = Prob[\text{correct prediction}|HC] = \frac{HCcorr}{HCcorr+HCntcorr}$$

$$\textbf{Sensitivity} = Prob[HC|\text{correctly predictable}] = \frac{HCcorr}{HCcorr+LCcorr}$$

We will plot figures of sensitivity versus prediction accuracy as sketched in Figure 4. Values closer to the upper right corner are better as perfect confidence assignment reaches 100% sensitivity and 100% prediction accuracy. A value predictor without confidence mechanism uses all predictions and achieves the highest possible sensitivity in exchange for lower prediction accuracy. A *stronger* confidence mechanism ignores more predictions by assigning low confidence to them and necessarily reaches lower sensitivities because the number of predictions in class HCcorr decreases (stronger mechanism) and the number of correctly predictable predictions is constant (fixed value predictor). The same reasoning in terms of the prediction accuracy is impossible, but a stronger mechanism should avoid more mispredictions than it looses correct predictions so that the prediction accuracy increases. In the limit when the sensitivity decreases down to 0% by using none of the predictions, prediction accuracy is strictly undefined, but we assume it approaches 100%. Figure 3 shows the sensitivity versus prediction accuracy for confidence mechanisms with 3-bit saturating counters and 10-bit patterns (threshold is varied along the curve).

4 Independent hashing

Using a hashing function in the Sazeides predictor causes different unhashed histories to be mapped on the same level-2 entry. This interaction is called *hash-aliasing* and occurs in 34% of all predictions, for a predictor of order 3 and both tables with 2^{12} entries. Only in 4% this results in a correct prediction whereas the other 30% end up in a misprediction [4]. In order to avoid these mispredictions we propose detecting hash-aliasing, assigning low confidence to the corresponding predictions and so eliminating predictions suffering from hash-aliasing.

First, the detection can be done perfectly by storing the complete unhashed history in both prediction tables. This requires a hardware budget that exceeds many times that of the value predictor itself and is not acceptable, but it gives an upper limit for sensitivity and prediction accuracy. Figure 5 shows a sensitivity of 96% and a prediction accuracy of more than 90%, a considerable improvement over counters and patterns. Note that only hash-aliasing is detected and that this technique does not try to estimate predictability.

Secondly, we perform the detection by a *second hashing function*, independent on the one used in the value predictor. This second hashing function maps the history on a *second hashing value*. The actions taken to locate the prediction and to compute the corresponding confidence are illustrated in Figure 7(a). The history table contains two independent hashing values based on the same complete history, while *val_hash2* corresponds to the history on which the value stored in the value field follows. High confidence is assigned when the second hashing values match, otherwise the prediction is of low confidence. Confidence information is spread over both prediction tables.

The second hashing function has to satisfy the following requirements:

1. If the Sazeides hashing function computes the same hashing value for two different unhashed histories, the second hashing function should map these histories to different values with a good chance. In other words, the hashing functions have to be *independent* meaning that none of the hashing bits can be derived by XORing any other combination of hashing bits.
2. All history bits should be used.
3. A new hashing value must be computable from the old hashing value and the computed value.

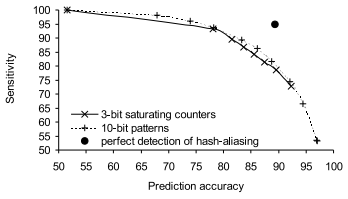


Fig. 5. Perfect detection of hash-aliasing compared to counters and patterns

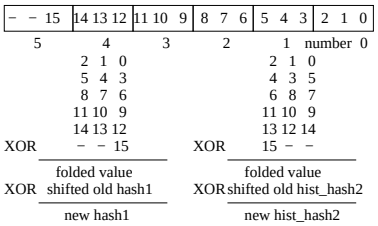
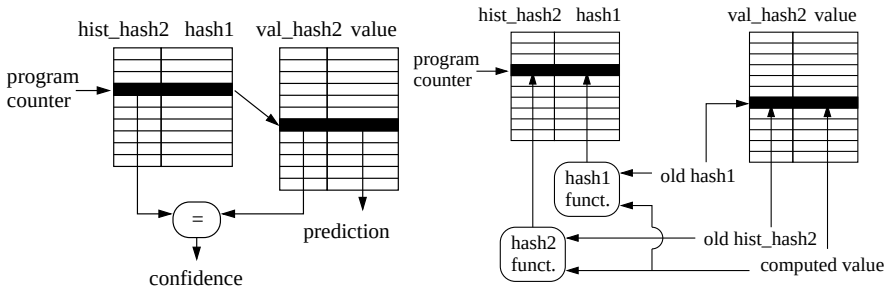


Fig. 6. Second, independent, hashing function on a 16-bit computed value and $b = 3$



(a) Predicting (b) Updating

Fig. 7. Independent hashing

By analogy with the hashing function from Sazeides we propose a second hashing function based on the fold-and-shift principle. Again we assume a hashing function mapping the history on a value of b bit illustrated in Figure 6 for $b = 3$. After splitting the computed value into sequences of b consecutive bits, the second hashing function first rotates the sequences to the left before XORing. If we number these sequences starting by zero, the rotation of each sequence is done over (*number* MOD b) bits. Once the folded value is computed, the calculation of both hashing values is similar.

The above-described second hashing function is easy to compute (shifting, rotating and XORing) and uses all history bits. We also examined the independence of the second hashing function upon the original one. Therefore we use a matrix that represents the hashing functions such that after multiplication with a column representing the unhashed history, both hashing values are computed. By verifying the independence of rows in this matrix we prove the independence of both hashing functions.

When the computed value is known the content in the prediction tables is updated. This update phase is shown in Figure 7(b).

5 Evaluation

For each configuration, we use trace-based simulations using traces generated on-the-fly by a SimpleScalar 2.0 simulator (sim-safe) [1]. The benchmarks are taken from the SPECint95 suite which are compiled with gcc 2.6.3 for SimpleScalar with optimisation flags "-O2 -funroll_loops". We use small input files (Figure 8) and simulate only the first 200 million instructions, except for m88ksim where we skip the first 250M. Only integer instructions that produce an integer register value are predicted, including load instructions. For instructions that produce two result registers (e.g. multiply and divide) only one is predicted. Finally value prediction was not performed for branch and jump instructions and the

Program	options, input	predictions
cc1	cccp.SS.i	140M
compress	test.in	133M
go	30 8	157M
ijpeg	-image_file	155M
	vigo_ref.ppm -GO	

Program	options, input	predictions
li	7queens.lsp	123M
m88ksim	-c ctl.raw.lit	139M
perl	scrabbl.pl	126M
	scrabbl7_train.in	
vortex	vortex.ref.lit	122M

Fig. 8. Description of the benchmarks

presented results show the weighted average over all SPECint benchmarks. When not explicitly mentioned we consider a FCM-based Sazeides value predictor of order 4 with 2^{12} entries in both tables. The original hashing function in the value predictor then folds each history into 12 bits. First we evaluate adding the confidence mechanism to the value predictor and not its embedding in an actual processor. Afterwards we check if the higher accuracy and higher sensitivity translate in an actual speedup.

5.1 Independent hashing

In this section we evaluate our second hashing function as confidence mechanism and we compare it to saturating counters and patterns, placed at the history table since this provides the best results. We found that using 4 bits in the second hashing value is a good choice as it assigns in 90% of the predictions the same confidence as perfect detection of hash-aliasing. Using more bits in the second hashing value do slightly better but require more hardware. The result of a 4-bit second hashing function is shown in Figure 9.

Our independent hashing function performs well in the sense that interaction between histories is detected and assigned low confidence. Nevertheless this technique does not account for predictability itself as high confidence is assigned every time no interaction occurs or can be detected. To correct this we propose combining detection of hash-aliasing with other confidence mechanisms. In a combined mechanism high confidence is assigned if both confidence mechanisms indicate high confidence. We can put the additional confidence in either of the two tables. If we add a simple 2-bit saturating counter with varying threshold, we get Figure 10. We also show the combination with a perfect detection system as well as the two possibilities for placing the saturating counter. The second hashing function approaches perfect detection when both are combined with a saturating counter. It gets even closer for higher thresholds. In the situation where we placed the counters at the value table only the highest threshold could be a meaningful configuration.

For a fair comparison in terms of hardware requirement we should compare 10-bit pattern recognition against the combination of a 4-bit second hashing function with 2-bit saturating counters. The difference is significant and moreover patterns need profiling, while our technique does not.

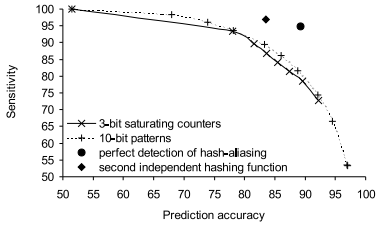


Fig. 9. 4-bit independent hashing

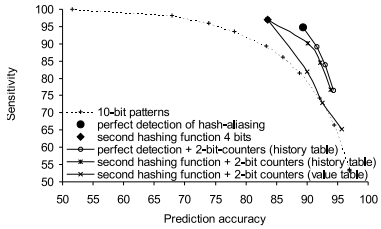


Fig. 10. 4-bit independent hashing combined with 2-bit saturating counters

Fetch, decode, issue, commit: 8
RUU/LSQ queue: 64/16
Functional units: 4
Branch predictor: perfect
L1 Icache: 128KB
L1/L2 latency: 3/12
L1 Dcache: 128
L2 cache (shared): 2MB
Recovery policy: selective

Fig. 11. Out-of-order architecture

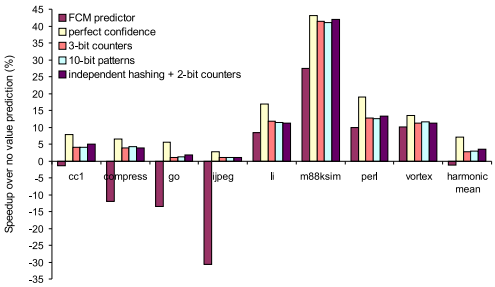


Fig. 12. Speedup over no value prediction

5.2 IPC

In this section we test if the higher prediction accuracy and higher sensitivity reached by independent hashing translate in actual speedup. Simulations are done by an out-of-order architecture (sim-outorder) as shown in Figure 11. In Figure 12 speedup over using no value prediction is plotted in 4 different cases: value prediction without confidence mechanism, perfect confidence mechanism, 3-bit saturating counter, 10-bit patterns and finally the combination of independent hashing with saturating counters. Independent hashing reaches a speedup that is only a slight improvement over patterns. An important aspect to increase performance by value prediction is *criticality* [3, 8]. Only correct predictions on the critical path can increase the performance while mispredictions are not dramatic when not on the critical path. None of the described confidence mechanisms consider about criticality of instructions and hence it is not evident that using more correct predictions do augment the IPC.

6 Conclusion

This paper studies confidence mechanisms for a context based Sazeides value predictor. We explain that many mispredictions are a result of using a hashing function and that detecting hash-aliasing can avoid a lot of mispredictions. Detection of hash-aliasing is done through a second, independent hashing function

as confidence mechanism. In case of detecting hash-aliasing the confidence mechanism assigned low confidence forcing the processor not to use the prediction. We evaluate our confidence mechanism and show a significant improvement according to saturating counters and patterns. Especially the combination of our technique with saturating counters translates in a slight speedup, needs the same storage as patterns and eliminates the use of profiling.

References

1. D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: The SimpleScalar Tool Set. Technical report, Computer Sciences Department, University of Wisconsin-Madison, July 1996.
2. M. Burtscher and B. G. Zorn. Prediction outcome history-based confidence estimation for load value prediction. *Journal of Instruction-Level Parallelism*, 1, May 1999.
3. B. Calder, G. Reinman, and D. M. Tullsen. Selective value prediction. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 64–74, May 1999.
4. B. Goeman, H. Vandierendonck, and K. D. Bosschere. Differential FCM: Increasing value prediction accuracy by improving table usage efficiency. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture*, pages 207–216, Jan. 2001.
5. D. Grunwald, A. Klauser, S. Manne, and A. Pleszkun. Confidence estimation for speculation control. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 122–131, 1998.
6. M. H. Lipasti and J. P. Shen. Exceeding the dataflow limit via value prediction. In *Proceedings of the 29th Annual International Symposium on Microarchitecture*, Dec. 1996.
7. T. N. Mudge, I.-C. K. Chen, and J. T. Coffey. Limits to branch prediction. Technical Report CSE-TR-282-96, The University of Michigan, Ann Arbor, Michigan, 48109-2122, 1996.
8. B. Rychlik, J. Faistl, B. Krug, and J. P. Shen. Efficacy and performance impact of value prediction. In *Parallel Architectures and Compilation Techniques (PACT)*, Oct. 1998.
9. Y. Sazeides and J. E. Smith. Implementations of context based value predictors. Technical Report ECE97-8, Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Dec. 1997.
10. Y. Sazeides and J. E. Smith. The predictability of data values. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, Dec. 1997.
11. K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 281–290, Dec. 1997.