Reducing Miss Penalty of Load Value Prediction using Load Address Prediction

Toshinori Sato

Toshiba Microelectronics Engineering Laboratory
580-1, Horikawa-Cho, Saiwai-Ku, Kawasaki 210-8520, Japan
toshinori.sato@toshiba.co.jp

Abstract. In this paper, we propose to combine load value prediction with load address prediction in order to reduce misprediction penalty. When a load value predictor can not predict a load value, it is supported by a load address predictor and data dependences can be speculated. We call this predictor a *cooperative predictor*. Furthermore, we propose to verify a value prediction by comparing the predicted value with the one obtained using the predicted load address. We call this scheme *speculative verification*. With the help of the speculative verification, a recovery action caused by a misprediction is initiated earlier than the time when the misprediction is really detected. Using a cycle-by-cycle simulator, we have evaluated the cooperative predictor and the speculative verification and found that data dependences are more aggressively speculated to exploit instruction level parallelism.

1 Introduction

Recently, data dependences are studied to be speculatively resolved in order to exploit more instruction level parallelism (ILP). An outcome of an instruction is predicted by value predictors and the instruction and its dependent instructions can be dispatched simultaneously, thereby highly ILP is exploited. The value predictors are classified into two types in terms of the predicted instruction class. One is that predicting execution results of all types of instructions which write values into registers[Gabbay et al.,1997;Lipasti,1997; Sazeides et al.,1997;Wang et al.,1997]. The other is that predicting only load data values[Gonzalez et al.,1997;Lipasti,1997;Moshovos et al.,1997;Sato,1998b; Sato, 1998c;Tyson et al.,1997;Widigen et al.,1996]. In this paper, I focus on the load value predictor.

In order to utilize value predictors effectively, the following two conditions must be considered. One is prediction accuracy and the other is prediction coverage. I define the prediction accuracy as the number of instructions whose outcome is correctly predicted over the total number of the predicted instructions and the prediction coverage as the number of correctly predicted instructions over the all load instructions. Since mispredictions cause penalties, the prediction accuracy should be high. The predictability is different according to individual instructions. Thus, the prediction accuracy is improved if only easily predictable

Proceedings of the Fourth Australasian Computer Architecture Conference, Auckland, New Zealand, January 18–21 1999.

instructions are predicted. However, this means the reduction of the prediction coverage. We have found that the dominant of performance improvement is not the prediction accuracy but the prediction coverage[Sato,1998a]. Therefore, we must improve both the prediction accuracy and the prediction coverage.

In order to reduce the misprediction penalty while the prediction coverage is increased, I propose a combination of the load value predictor and a load address predictor. Address prediction enjoys better accuracy than value prediction, but does not result in as big a win when correct. Therefore, address prediction is proposed to be used in cases where value prediction is unable to predict a load. I call this predictor a *cooperative predictor*. Furthermore, I investigate to verify predicted values using the address predictor, thereby misprediction penalty can be further reduced. A predicted value is compared with the one which is obtained using the predicted address. I call this scheme *speculative verification*. With the help of the speculative verification, a recovery action caused by a misprediction is initiated earlier than the time when the misprediction is really detected.

The organization of the rest of this paper is as follows. In Section 2, previously proposed related works are surveyed. The cooperative predictor and the speculative verification are explained in Section 3. In Section 4 evaluation methodology is described and simulation results are shown in Section 5. Finally, our conclusions are presented in Section 6.

2 Related Work

There are many studies predicting register values[Gabbay et al.,1997; Gonzalez et al.,1997;Lipasti,1997;Moshovos et al.,1997;Sato,1998b;Sato,1998c; Sazeides et al.,1997;Tyson et al.,1997;Wang et al.,1997;Widigen et al.,1996]. These predictors are classified into two types in terms of the predicted instruction class. One is that predicting execution results of all types of instructions which write values into registers[Gabbay et al.,1997;Lipasti,1997;Sazeides et al.,1997; Wang et al.,1997]. The other is that predicting only load values[Gonzalez et al.,1997;Lipasti,1997; Moshovos et al.,1997;Sato,1998b;Sato,1998c; Tyson et al.,1997; Widigen et al., 1996]. Last value predictor proposed by Lipasti[Lipasti,1997] introduces the value prediction concept, and is based on value locality. An instruction uses the same value which is executed at the last time, when the same instruction is emerged in the future. Operand prefetch cache[Widigen et al.,1996] also uses the last outcome of a load instruction. Stride predictor proposed by Gabbay et al.[Gabbay et al.,1997] keeps not only the last outcome of an instruction but also a stride which is the difference between last two outcomes of the instruction. The predicted value is the sum of the last outcome and the stride.

In order to improve prediction accuracy, several hybrid predictors[Sazeides et al.,1997;Wang et al.,1997] are proposed. The hybrid predictor is a combination of several value predictors with a selector choosing the probably most accurate one. An another approach to improve value prediction accuracy is utilization of program execution profiles[Gabbay et al.,1997]. Using the profiles, instructions are classified according to the predictability, and compiler provides the classified information to processors. The prediction accuracy of load values can be improved by utilizing store instruction information. The schemes proposed by Moshovos et al.[Moshovos et al.,1997], Sato[Sato,1998b;Sato,1998c], and Tyson et al.[Tyson et al.,1997] are very similar with each other, and thus I call them renaming-based value predictors in this paper. The renaming-based predictors speculatively streamline a stored value to a load instruction. The

load value predictor used in this paper is based on one of the renaming-based predictors[Sato,1998b;Sato,1998c].

There are many proposals predicting data addresses of load instructions [Gonzalez et al.,1997;Sato,1997a;Sato,1997b;Sazeides et al.,1996; Widigen et al., 1996]. The basic schemes they use are similar with each other. They keep histories of memory references and predict a data address as the sum of a previous data address and a stride which is the difference of the last two addresses generated by an instruction. The data address prediction method utilized in this paper is based on [Sato,1997a;Sato,1997b].

To the best of our knowledge, predictors combining a load value predictor and a load address predictor have not been investigated. The predictor proposed by Gonzalez et al.[Gonzalez et al.,1997] predicts both load values and addresses, but the load values are obtained using the predicted load addresses. Thus, the load address predictor and the load value predictor do not work independently. On the other hand, our proposed load value predictor does not use the predicted address and hence works independently of the load address predictor. Furthermore, the verification of predicted values can be accelerated using the predicted address.

3 Proposed Schemes

The cooperative predictor is independent of the kinds of load value and address predictors of which it consists. Firstly in this section, I briefly explain a load value and a load address predictors used in this paper. Next, I describe the cooperative predictor and the speculative verification.

3.1 Load Value Predictor

The renaming-based load value predictor used in this study consists of three tables which are data-indexed store table (DIST), store-indexed value table (SIVT), and load-indexed store table (LIST). The DIST is indexed by a data address and keeps histories of memory references generated by store instructions. When it is referred by a load instruction, it links the load and a store instructions which refer the same memory location. The LIST is indexed by an instruction address and keeps links. When the load instruction is fetched, the LIST is referred and supplies a tag which represents the link. The SIVT is indexed by the tag and keeps data values each of which is written by the store instruction specified by the tag. Thus, the load instruction can obtain the data value before calculating the data address. The detailed explanations are presented in [Sato,1998b;Sato,1998c].

3.2 Load Address Predictor

In order to predict data addresses, reference prediction table (RPT) [Chen et al.,1995] is utilized. The RPT, which has a similar structure with the instruction cache, is proposed by Chen et al. for hardware prefetching and keeps track of previous memory references. An entry of the RPT is indexed by an instruction address and holds a previous data address, a stride value, and a state information. The stride is the difference between last two data addresses generated by an instruction. The state information encodes the past history and indicates if next prediction is valid. The predicted address is the sum of the previous address and the stride. If the state information decides that the predicted address is valid, the load instruction speculates data dependences using the predicted address. The detailed explanations are presented in [Sato,1997a;Sato,1997b].

3.3 Cooperative Predictor

It is possible to improve the prediction coverage if the value predictor is supported by the address predictor when the value predictor can not predict load values. In such a case, the load address is predicted and data dependences are speculated. I call the value predictor supported by the address predictor a *cooperative predictor*. Note that the combination is not a hybrid one[Sazeides et al.,1997; Wang et al.,1997] meaning that there is no mechanism present to choose from either prediction method.

Examples are useful to understand how the cooperative predictor works. Fig. 1 shows an example of an instruction sequence and there is a data dependence between instructions `I1` and `I2`. It is assumed that register $r12$ is ready and $r10$ will be ready at one cycle later. It is executed as shown in Fig. 2 when the dependence is not speculated. Figs. between 3 and 5 explain how the instructions speculate the dependence. When the load value for `I1` is correctly predicted, the instruction sequence goes as shown in Fig. 3. `I2` is dispatched using the predicted load value (**r11**). The predicted value is in bold. When the actual load value is obtained, it is compared with the predicted one and the speculation finishes since the prediction is correct. The dependence length is decreased by three.

```
I1: load r11 ← r10(4)
I2: add  r13 ← r11 + r12
```

Fig. 1. Instruction sequence example

When the load value can not be predicted the instruction sequence is as same as that shown in Fig. 2. If both load value and address predictors are used, that is the cooperative predictor is used, the sequence goes as shown in Figs. 4 and 5 according to the address prediction result. The load value is obtained using the predicted address (**ad**). The predicted address and the value obtained using the predicted address are in bold. The comparison of the predicted address with the actual one can be done simultaneously with the address calculation[Cortadella et al.,1992], and thus the data fetching using the actual address is suppressed when two addresses match. `I2` speculates data dependence using the load value which is read using the predicted address, and when the prediction is correct the speculation finishes as shown in Fig. 4. Otherwise, `I1` fetches data again using the actual address and `I2` is invalidated and reissued as shown in Fig. 5.

In summary, when a load value can not be predicted, the cooperative predictor decreases the data dependence length by two if the predicted address is correct, and does not increase it even if the prediction fails.

3.4 Speculative Verification

For the cooperative predictor, the value predictor is supported by the address predictor only when the value prediction is not initiated. If the value prediction fails, the processor suffers whole miss penalty. In order to reduce the penalty, the cooperative predictor is extended using the *speculative verification*. The predicted value is compared with the value obtained using the predicted address. If they match, the value prediction goes as normally. Otherwise, a load value misprediction is speculatively detected and the instructions dependent upon the probably mispredicted load instruction are invalidated and reissued. With the

	time →			
I1		ad←r10+4	r11←mem[ad]	
I2				r13←r11+r12

Fig. 2. Without any predictions

	time →			
I1		ad←r10+4	r11←mem[ad]	*compare* r11
I2	r13←**r11**+r12			

Fig. 3. Correct value prediction

	time →			
I1	**r11**←mem[**ad**]	ad←r10+4 *compare* ad		
I2		r13←**r11**+r12		

Fig. 4. Correct address prediction

	time →			
I1	**r11**←mem[**ad**]	ad←r10+4 *compare* ad	r11←mem[ad]	
I2		r13←**r11**+r12 *invalidated*		r13←r11+r12

Fig. 5. Failed address prediction

help of the speculative verification, a recovery action is initiated earlier than the time when the misprediction is really detected, and thus the miss penalty is reduced.

Figs. 6 and 7 explain how the speculative verification works. The instruction sequence is as same as that used in the previous subsection. When a misprediction occurs in the load value predictor, the instruction sequence goes as Fig. 6. The predicted value is compared with the actual one when it is obtained, and `I2` is invalidated since they do not match. After the misprediction is detected, `I2` is reissued and executed again at the next cycle. In such a case, the processor suffers the miss penalty of one cycle since the reissue of the `I2` is delayed due to the comparison. If the verification of the predicted value is accelerated, the miss penalty is removed and the dependence length is reduced by two as shown in Fig. 7. The comparison between the predicted value and the one obtained using the predicted address is in bold.

There may be a question what happens when the value prediction succeeds and the address prediction fails. Does a processor suffer miss penalty caused by address misprediction even though the predicted value is correct ? The answer is that it does not always suffer the miss penalty. The speculative verification is harmless as long as the comparison between the predicted address and the actual one is done before the comparison of the load values. That is, if the actual address is calculated before a data value is obtained using the predicted address, the speculative verification is canceled as shown in Fig. 8. If the verification

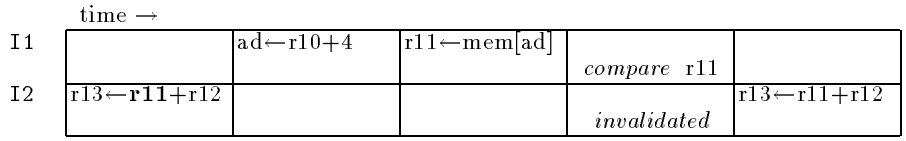


Fig. 6. Failed value prediction

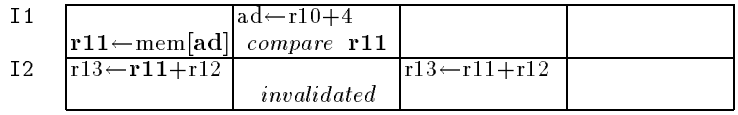


Fig. 7. Speculative verification

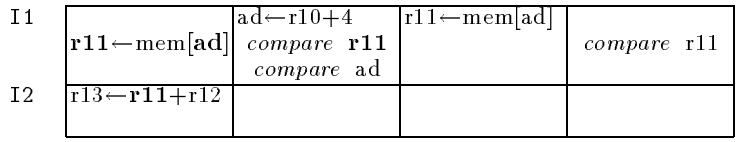


Fig. 8. Canceled speculative verification

of the predicted address is later, the speculative verification might invalidate a correct value prediction by an incorrect address prediction and reissue dependent instructions using a wrong data value. In such a case, the dependent instructions are reissued again after the address misprediction is detected.

In summary, whenever a load instruction is issued, both load value and address are predicted. If the value prediction is not initiated, the address prediction contributes to data dependence speculation. On the other hand, if the value prediction fails, the address prediction reduces the miss penalty caused by the value misprediction.

4 Evaluation Methodology

In this section, I describe my evaluation methodology.

The baseline model is an out-of-order execution superscalar processor based on register update unit[Sohi,1990]. Following the discussion explained in [Jourdan et al.,1995], I decide the configuration of the baseline processor summarized in Table 1. I evaluate the effect of the proposed mechanism by using SimpleScalar tool set[Burger et al.,1997]. The SimpleScalar architecture is based on MIPS architecture, and a fully execution-driven and cycle-by-cycle simulator is executed on a SPARCstation. For the load value predictor, the SIVT, LIST, and DIST are assumed to be direct-mapped and to have 4096 entries, respectively. For the load address predictor, the RPT is assumed to be direct-mapped and to have 4096 entries.

The SPEC95 CINT benchmark suite is used for this study. The `test` input files which are provided by SPEC are used. I use the object files provided by University of Wisconsin Madison[Burger et al.,1997]. Each program is executed to completion or for the first 100 million instructions. I count only committed instructions.

Table 1. Processor configuration

Fetch Width	8 instructions
Br. Predictor	512 entry 2way set-associative BTB, gshare scheme, 12b BHR, 4096 entry PHT, speculatively updated in ID stage, 8 entry return address stack, 3 cycle miss penalty
Insn. Windows	64 entry instruction queue, 8 entry load/store queue
Issue Width	8 instructions
Commit Width	8 instructions
FUs	5 iALU's, 1 iMUL/DIV, 4 Ld/St, 2 fALU's, 2 fMULs, 2 fDIVs
Latency (total/issue)	iALU 1/1, iMUL 3/1, iDIV 35/35, Ld/St 2/1, fADD 2/1, fMUL 3/1, fDIV 6/6
Register Files	32 32b fixed point registers, 32 32b floating point registers
Insn. Cache	64K 4way set-associative, 32 byte blocks, 4-port, 6 cycle miss penalty
Data Cache	64K 4way set-associative, 32 byte blocks, 4-port, writeback, non-blocking load, hit under miss, 6 cycle miss penalty
Unified L2 Cache	256K 4way set-associative, 64 byte blocks, 32 cycle miss penalty

5 Experimental Results

This section presents experimental results. For measuring performance, I use the committed instructions per cycle (IPC). Only useful instructions are considered for counting the IPC. I define the performance improvement rate as the increased IPC over the IPC of the baseline model. Sometimes I use a term "average" which is the mean of all programs used.

Firstly in this section, I evaluate a relationship between the prediction coverage and performance improvement. Next, I present the prediction coverage of the load value and the load address predictors in order to confirm our motivation. Then, I evaluate the cooperative predictor. And last, I present the effect of the speculative verification.

5.1 Preliminary Evaluation

Fig. 9 shows performance improvement rate when a synthetic load value predictor whose prediction accuracy is 100% is used. Predicted instructions are randomly picked up. For each group of six bars, the bars from left to right

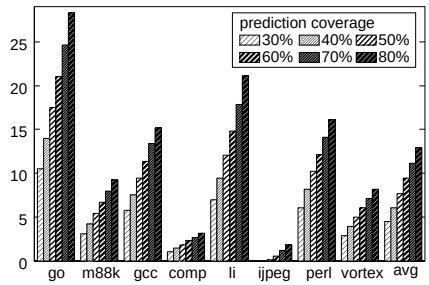
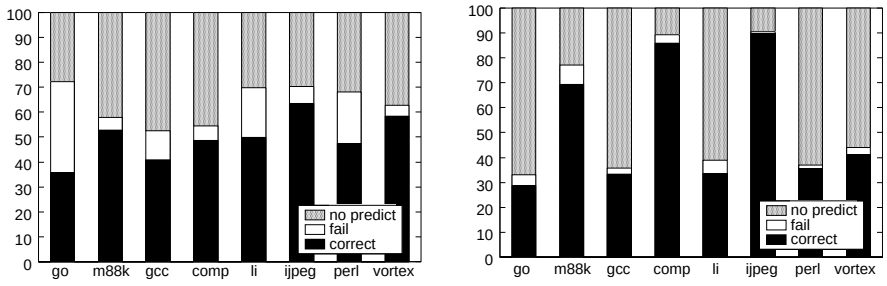


Fig. 9. (%)Performance improvement of ideal value predictor

indicate the performance improvement rate when the prediction coverage is varied between 30% and 80%. Naturally, the higher the prediction coverage is, the larger the performance improvement is. It can be also observed that the prediction coverage should be more than 70% in order to achieve the performance improvement of 10% on average.

Fig. 10(i) presents the prediction coverage when the load value predictor is utilized. Each bar is divided into three parts. The bottom part (black) indicates the percentage of the load instruction whose data value is correctly predicted. The middle part (white) indicates the percentage that is mispredicted. And the top part (gray) indicates the percentage that is not predicted. The prediction coverage depicted as the bottom part is approximately 50%, and hence it may not be enough to improve processor performance over 10% when penalties caused by mispredictions are considered. It is necessary to improve the prediction coverage without diminishing the prediction accuracy.



(i) value predictor (ii) address predictor

Fig. 10. (%)Prediction coverage

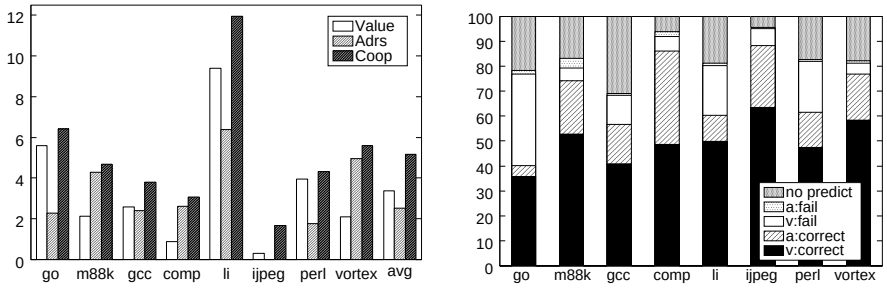
Fig. 10(ii) shows the prediction coverage when the load address predictor is utilized. Just like Fig. 10(i), each bar is divided into three parts. As can be seen, the prediction coverage is significantly high for `124.m88ksim`, `129.compress`, and `132.ijpeg`, and thus it is expected that the value predictor is supported by the address predictor when it can not predict the load. For the remaining programs, the prediction coverage is small. However, if the address predictor covers the loads which can not be covered by the value predictor, total prediction coverage may be increased.

5.2 Cooperative Predictor

Fig. 11(i) shows performance improvement rate. For each group of three bars, the bars from left to right indicate the improvements of the load value (`Value`), load address (`Adrs`), and cooperative (`Coop`) predictors, respectively. For all programs, the cooperative predictor exploits more ILP than both the load value and address predictors. Compared with the value predictor, performance improvement rate on average is increased by 1.8 point and is 5.2%.

Fig. 11(ii) shows the prediction coverage of the cooperative predictor. Because the address predictor is utilized only when the value predictor can not predict data values, only the top part, which indicates the percentage of the load instruction whose data value is not predicted, of each bar in Fig. 10(i) is divided into three parts and each bar in Fig. 11(ii) is divided into five parts. The

bottom part indicates the percentage of the load instruction whose data value is correctly predicted. The next part indicates the percentage that of the load instruction whose data value is not predicted but whose data address is correctly predicted. The next part indicates the percentage that of the load instruction whose data value is mispredicted. The next part indicates the percentage that of the load instruction whose data address is mispredicted. And the top part indicates the percentage that of the load instruction whose data value and address are not predicted. Note that the parts `v:correct` and `v:fail` in Fig. 11(ii) equal the parts `correct` and `fail` in Fig. 10(i) respectively and that the sum of the parts `a:correct`, `a:fail`, and `no predict` in Fig. 11(ii) equals the part `no predict` in Fig. 10(i). For the cooperative predictor, the prediction coverage consists of the bottom two parts. Compared with the coverage of the load value predictor, the average prediction coverage is improved by 18.4 point and is 68.1%. This is the reason why performance improvement rate is increased. Referring back to Fig. 9, the improvement rate is approximately 10% on average when the prediction coverage is 68.1%. However, it is actually 5.2%. This is due to the following reasons. First, the synthetic predictor used in Fig. 9 is ideal and does not occur mispredictions, thereby it does not suffer from mispredictions. Second, while the real predictor captures only instructions which is predicted easily, the synthetic predictor predicts easy and hard values equally well. Since hard predictions will give more benefit, the performance improvement rate of the synthetic predictor is larger than that of the real predictor when the prediction coverage is equal.



(i) (%)performance improvement (ii) (%)prediction coverage

Fig. 11. Simulation results for cooperative predictor

I also evaluate the load value predictor whose LIST is doubled; i.e. it has 8192 entries. It is interesting to compare the base value predictor and the larger one, since the hardware budget of the cooperative predictor is larger than that of the base value predictor. Note that the hardware budget of the added 4096 entries in the LIST is approximately as same as that of the address predictor (the RPT) which has 4096 entries, and hence the hardware budget of the larger value predictor is approximately as same as that of the cooperative predictor. Fig. 12 shows the simulation results. For each programs, the left bar is for the base value predictor and the middle one is for the larger one. The right bar is for the cooperative predictor. As can be easily observed, the difference between two value predictors is very small. Performance improvement rate is increased by only 0.1 point over the base value predictor and is 3.5% on average. Therefore,

the cooperative predictor is more effective than the larger value predictor if their hardware budgets are approximately same.

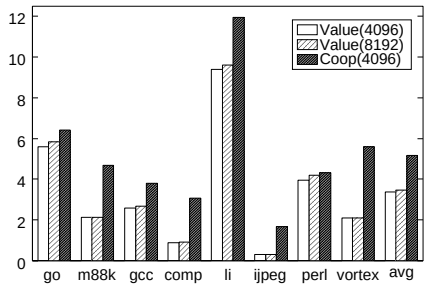
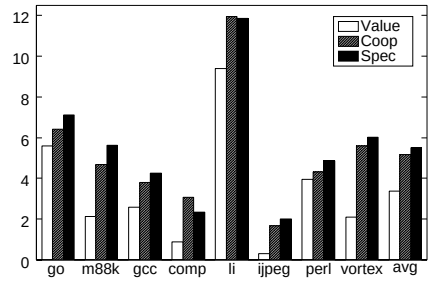


Fig. 12. (%)Performance improvement of larger value predictor

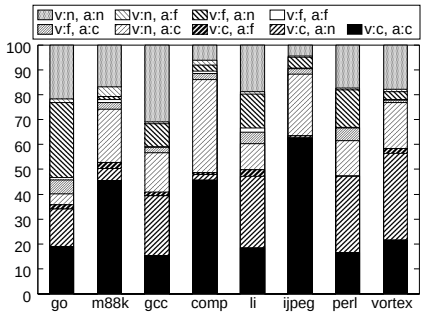
Anyway, the cooperative predictor improves the prediction coverage, and thereby the performance improvement rate is increased.

5.3 Speculative Verification

Fig. 13(i) presents performance improvement rate of speculative verification. For each group of three bars, the bars from left to right indicate the improvements of the load value predictor (`Value`), the cooperative predictor (`Coop`), and that using the speculative verification (`Spec`), respectively. For six of eight programs, the cooperative predictor using the speculative verification exploits more ILP than the base cooperative predictor. Compared with the base cooperative predictor, performance improvement rate on average is increased by only 0.32 point and is 5.5%. For two programs, processor performance is degraded because correct value predictions are invalidated by incorrect address predictions and thus data dependences are not be speculated.



(i) (%)performance improvement



(ii) (%)prediction coverage

Fig. 13. Simulation results for speculative verification

Fig. 13(ii) shows the prediction coverage when the speculative verification is used. Each bar is divided into nine parts. They are combinations of three cases (correct, fail, not predict) of the value prediction and those of the address prediction. In Fig. 13(ii), the value and address predictions are denoted with `v:` and `a:`, respectively. Also in Fig. 13(ii), the correct, fail, and not predict cases

are denoted with `c`, `f`, and `n`, respectively. When the speculative verification is utilized, the prediction coverage consists of the bottom five parts in Fig. 13(ii). Compared with the coverage of the cooperative predictor, the average prediction coverage is increased by 3.2 point and is 71.3%. This contributes to the increasing performance improvement of 0.32 point.

From these simulation results, the speculative verification little contributes to the enhancement of the cooperative predictor, hardly justifying the cost of the implementation.

6 Concluding Remarks

In this paper, I have proposed to combine load value prediction with load address prediction in order to reduce misprediction penalty. Since the prediction coverage dominates processor performance improvement, it is important to increase the prediction coverage. When a load value predictor can not initiate a value prediction, a load address predictor supports it by predicting a load address and a processor speculates data dependences. I have also proposed the speculative verification. Using the speculative verification, the predicted load value is verified by the one obtained using the predicted load address, and the value prediction is verified earlier.

From experimental simulations, it is found that the cooperative predictor improves the prediction coverage and increase performance improvement rate. The prediction coverage is 68.1% on average, and processor performance is improved by 5.2% on average with a maximum of 11.9%. Using the speculative verification, the average prediction coverage is further increased by 3.2 point and is 71.3% and it improves processor performance by 5.5% on average with a maximum of 11.9%. These results confirms that the cooperative predictor enhances data dependence speculation and exploit more ILP. However, it can not be said that the speculative verification improves the performance of the cooperative predictor, when its implementation cost is considered.

Future studies dealing with the cooperative predictor include comparing the cooperative predictor with hybrid predictors and investigating alternative cooperative predictors. A hybrid predictor selects a predictor whose prediction accuracy is probably higher, and thus the prediction coverage might be improved at the hardware cost of a predictor selecting counter. If the selection is statically decided at compile time using execution profiles, the hybrid predictor need not have the counter and the hardware costs of the cooperative predictor and the hybrid one are same. Thus, it is interesting to compare these predictors. One of the alternative cooperative predictors is a load address predictor supported by a load value predictor. In other words, the address predictor has priority. If the cases that a value prediction is incorrect and an address prediction is correct frequently occur, the alternative predictor is favorable. And then, I have a plan to evaluate the alternative cooperative predictor.

Acknowledgment

The author is grateful to Dr. Mitsuo Saito and Dr. Shigeru Tanaka for their continuous encouragements. He also thanks the anonymous reviewers whose comments and suggestions helped to improve the quality of this paper.

References

Burger,D. and Austin,T.M. (1997). The SimpleScalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3).

Chen,T-F. and Baer,J-L. (1995). Effective hardware-based data prefetching for high-performance processors. *IEEE Trans. Comput.*, 44(5).

Cortadella,J. and Llaberia,J.M. (1992). Evaluation of A+B=K conditions without carry propagation. *IEEE Trans. Comput.*, 41(11).

Gabbay,F. and Mendelson,A. (1997). Can program profiling support value prediction?. *30th Int'l Symp. on Microarchitecture.*

Gonzalez,J. and Gonzalez,A. (1997). Speculative execution via address prediction and data prefetching. *11th Int'l Conf. on Supercomputing.*

Jourdan,S., Sainrat,P., and Litaize,D. (1995). Exploring configuration of functional units in an out-of-order superscalar processor. *22nd Int'l Symp. on Computer Architecture.*

Lipasti,M.H. (1997). Value locality and speculative execution. *PhD thesis*, Department of Electrical and Computer Engineering, Carnegie Mellon University.

Moshovos,A.I. and Sohi,G.S (1997). Streamlining inter-operation memory communication via data dependence prediction. *30th Int'l Symp. on Microarchitecture.*

Sato,T. (1997a). Data dependence speculation combining memory disambiguation with address prediction. *10th Summer United Workshop on Parallel, Distributed, and Cooperative Processing*, IPS Japan SIG Notes 97-ARC-125-1.

Sato,T. (1997b). Speculative resolution of ambiguous memory aliasing. *Int'l Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, IEEE Computer Society Press.

Sato,T. (1998a). Analyzing overhead of reissued instructions on data speculative processors. *Workshop on Performance Analysis and its Impact on Design*, held in conjunction with 25th Int'l Symp. on Computer Architecture.

Sato,T. (1998b). Load value prediction using reference address renaming. *10th Joint Symp. on Parallel Processing* (In Japanese).

Sato,T. (1998c). Load value prediction using two-hop reference address renaming. *Int'l Conf. on Computer Science & Infomatics.*

Sazeides,Y., Vassiliadis,S., and Smith,J.E. (1996). The performance potential of data dependence speculation & collapsing. *29th Int'l Symp. on Microarchitecture.*

Sazeides,Y. and Smith,J.E. (1997). The predictability of data value. *30th Int'l Symp. on Microarchitecture.*

Sohi,G.S. (1990). Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. *IEEE Trans. Comput.*, 39(3).

Tyson,G. and Austin,T.M. (1997). Improving the accuracy and performance of memory communication through renaming. *30th Int'l Symp. on Microarchitecture.*

Wang,K. and Franklin,M. (1997). Highly accurate data value prediction using hybrid predictors. *30th Int'l Symp. on Microarchitecture.*

Widigen,L., Sowadsky,E., and McGrath,K. (1996). Eliminating operand read latency. *ACM SIGARCH Computer Architecture News*, 24(5).