Decoupling Recovery Mechanism for Data Speculation from Dynamic Instruction Scheduling Structure

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 decouple the recovery mechanism for data speculation from dynamic instruction scheduling structure. Instruction reissue mechanism for data speculation has a serious impact on processor performance. The effective capacity of instruction window is reduced since instructions dependent upon a speculated instruction must remain in instruction window until they are committed. The decoupling of the recovery and scheduling mechanisms solves the problem. A small instruction window schedules instructions and its entry is released immediately when an instruction is dispatched. A large instruction buffer is active only when a misspeculation occurs and is used to reissue instructions dependent upon the misspeculated instruction. Using a cycle-by-cycle simulator, we evaluated the proposal and found that the decoupling is useful.

1 Introduction

Recently, there are many studies which try to speculate data dependences in order to extract 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 ILP is exploited aggressively. If a speculation is mispredicted, it is necessary to recover processor state. A straightforward implementation of the recovery mechanism is instruction squashing which is already used for branch prediction. The instruction squashing is not adequate for the data speculation, because it throws away execution results of instructions which are independent of the mispredicted instruction and because these instructions should be fetched again from instruction memory. This wastes useful computations. Moreover, since penalty caused by data dependence misprediction is quite larger than that caused by control dependence misprediction, the overhead including instruction squashing and re-fetching is very serious. Instruction reissue is one of the promising solutions for this problem and several studies of the instruction reissue are done[7, 8, 10]. However, the instruction reissue mechanism for data speculation has a serious impact on processor performance. The effective capacity of instruction window is reduced since instructions dependent upon a speculated instruction must remain in instruction window until they are committed. In this paper, we propose to decouple the recovery mechanism for data speculation from dynamic instruction scheduling structure. The decoupling solves the problem. A small instruction window schedules instructions and its entry is released immediately after an instruction is dispatched. A large instruction buffer is active only when a misspeculation occurs and is used to reissue instructions dependent upon the misspeculated instruction.

The organization of the rest of this paper is as follows. In Section 2, previously proposed related works are surveyed. Section 3 explains the decoupling of the recovery and scheduling mechanisms. Section 4 presents the evaluation methodology and Section 5 evaluates the proposed mechanism. Finally, our conclusions are presented in Section 6.

2 Related Work

Data speculation[4, 5] is a technique which executes instructions speculatively using predicted data values. Data dependences are speculatively resolved and thus ILP is increased. When a misspeculation occurs, it is necessary to recover processor state.

There are two mechanisms for the recovery action. One is instruction squashing and the other is instruction reissue. The instruction squashing flushes all instructions following a mispredicted instruction. It is easy to implement the instruction squashing since the same mechanism is already implemented for branch prediction. However, it is found that data speculation relying upon the instruction squashing sometimes diminishes processor performance due to large misprediction penalties[8, 10]. On the other hand, the instruction reissue re-executes only instructions dependent upon a misspeculated instruction. Those instructions are detected selectively and reissued inside instruction window. In order to realize the instruction reissue, the dependent instructions are forced to retain in instruction window. When the predicted instruction produces an actual value, the predicted value must be compared with the actual one. If they match, the prediction is correct and the dependent instructions release instruction window. If the prediction fails, the dependent instructions are invalidated and reissued. Lipasti et al.[5] proposed the concept of the instruction reissue but did not mention its implementation.

Tyson et al.[10] evaluated the usefulness of the instruction reissue. They proposed a renaming-based load value predictor and found that the instruction reissue proposed in [5] can improve processor performance even for the applications whose performance is degraded when the instruction squashing is used. However, the practical implementation of the instruction reissue was not discussed. Rotenberg et al.[7] investigated an instruction reissue scheme on Trace Processor architecture and found it is useful for dataflow speculation. However, they did not evaluate it on superscalar processors which are currently in mainstream.

We proposed a practical implementation of the instruction reissue[8]. Register update unit (RUU)[9] is extended to realize the instruction reissue. The RUU is very suitable for the instruction reissue, since it forces each instruction to retain until the instruction has been committed. Our instruction reissue mechanism serially detects and reissues all instructions dependent upon a misspeculated instruction, and thus a lot of comparators working in parallel to detect the dependent instructions can be removed.

While the instruction reissue considerably reduces misspeculation penalty by selectively re-executing instructions, it has a negative impact on processor performance[3]. As explained above, in order to reissue only dependent instructions, those instructions must remain in instruction window until they are committed. This reduces utilization of instruction window and thus its effective capacity becomes small. The instruction window size affects processor performance significantly. Therefore, it is necessary to increase the instruction window size in order to maintain its effective capacity. However, the instruction window is one of the dominant of processor cycle time and its size severely affects its speed[6].

Recently, Akkary et al.[1] proposed two level instruction window structure. There are two instruction windows. One is small and its entry is released immediately when an instruction allocated to the entry is dispatched. The other is large and works as the backup of the small one when a misspeculation occurs. The backup process works just like cache refill process. Dependent instructions which should be reissued are injected to the small instruction window from the large one. Since most of the time the small instruction window is utilized, the negative impact on processor performance is reduced. They evaluated the two

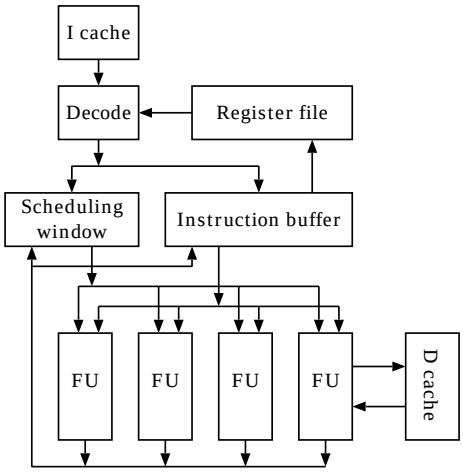


Fig. 1. Processor diagram

level instruction window only on a multithreading processor.

3 Decoupled Instruction Window

In this section, we propose a decoupled instruction window. As explained above, implementing the instruction reissue mechanism causes the following problem. Since every instruction must remain in instruction window until it is committed, the effective capacity of instruction window is reduced. In order to keep processor performance, it is necessary to increase the instruction window size. However, it is difficult to maintain processor cycle time for large instruction window. Thus, the wakeup and select logic of the large window should be pipelined, degrading processor performance[6].

For the purpose of solving the problem, we propose to decouple the recovery mechanism for data speculation from dynamic instruction scheduling structure. Fig.1 depicts a processor utilizing the decoupled instruction window. The decoupled window consists of a small instruction window for the scheduling and a large instruction buffer for the instruction reissue. After an instruction is fetched and decoded, it enters both the instruction scheduling window and the instruction buffer. When the instruction is dispatched to a functional unit, it leaves the instruction window and release its entry but remains in the instruction buffer. When it is committed, it leaves the instruction buffer and releases its entry. In the case that either the window or the buffer is full, issuing instruction into the decoupled window stalls.

The small instruction window works for dynamic instruction scheduling. Thus, most of the times each instruction is dispatched from the small window. Since instructions are aggressively deallocated from the scheduling window when they are dispatched, the problem reducing its effective capacity is solved. In addition, its small size does not have serious impact on processor cycle time. However, it is impossible to reissue misspeculated instructions inside the scheduling window. The large instruction buffer works as the backup for the small window and performs the instruction reissue. Each instruction remains in the buffer until it is committed. When a misspeculation occurs, reissued instructions are obtained from the instruction buffer. In order to aggressively speculate instructions, the instruction buffer should be very large. Since it is difficult to access such a large

buffer in one cycle, the wakeup and select logic of the buffer is pipelined to maintain high processor cycle time. It is expected that the pipelining does not degrade processor performance, since the instruction buffer is active only when misspeculations are detected.

The decision from which structures an instruction is dispatched to a functional unit works straightforwardly. When an instruction is misspeculated, its dependent instructions which have already dispatched and thus should be reissued are obtained from the instruction buffer only. They have already left the scheduling window. The dependent instructions which have not been dispatched remain in both the scheduling window and the buffer. Thus, both structures can dispatch the instructions. However, note that the instruction buffer is pipelined, and the instructions are obtained from the scheduling window earlier than from the buffer. And then, the same instruction provided by the buffer can be canceled to dispatch. Therefore, those instructions are scheduled by the window and release their entries.

From the explanations, it can be observed that the decoupled instruction window solves the problem caused by the instruction reissue and will maintain processor performance.

4 Evaluation Methodology

In this section, we describe the evaluation methodology by explaining a processor model and benchmark programs.

4.1 Experimental model

An execution-driven simulator which models wrong path execution caused by misspeculations is used for this study. We implemented the simulator using the SimpleScalar tool set[2]. The SimpleScalar/PISA instruction set architecture (ISA) is based on the MIPS ISA.

The simulator models a realistic 8-way out-of-order execution superscalar processor based on RUU[9] which has 128 entries. We model two RUUs. One is a one cycle latency RUU and the other is a two cycle latency pipelined RUU. Each functional unit can execute all operations and has a latency of 1 cycle except for multiplication (4 cycles) and division (12 cycles). A 4-port, non-blocking, 128KB, 32B block, 2-way set-associative L1 data cache is used for data supply. It has a load latency of 1 cycle after the data address is calculated and a miss latency of 6 cycles. It has a backup of a 8MB, 64B block, direct-mapped L2 cache which has a miss latency of 18 cycles for the first word plus 2 cycles for each additional word. No memory operation can execute beyond a store whose data address is unknown. A 128KB, 32B block, 2-way set-associative L1 instruction cache is used for instruction supply and also has the backup of the L2 cache which is shared with data supply.

For control prediction, a 1K-entry 4-way set associative branch target buffer, a 4K-entry gshare-type branch predictor, and an 8-entry return address stack are used. The branch predictor is updated at instruction commit-time.

Value predictor used in this study is a 4096 entry direct-mapped stride predictor[4]. Fig.2 depicts the predictor. It is indexed by instruction address and each entry has a tag field (`tag`), a previous value field (`prev_value`), a stride field (`stride`), and a confidence field (`conf`). The tag field is used for distinguishing individual instructions from each other. The previous value field holds the last value generated by the instruction. The stride field keeps a difference of the last two values. The predicted value is predicted as the sum of the previous value and the stride. The confidence field is a 2-bit saturated counter and decides if the speculation using the predicted value should be initiated. When the count is

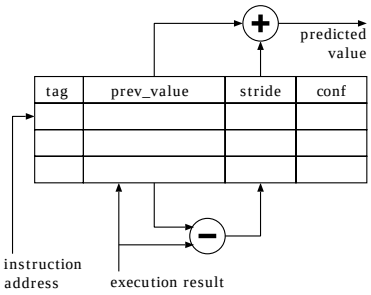


Fig. 2. Stride value predictor

larger than two, the speculation is initiated. In this paper, we call the processor model using the value predictor value prediction model.

The decoupled instruction window consists of a 64 entry centralized reservation station and a 128 entry RUU performing the instruction reissue[8]. The wakeup and select logic of the RUU is pipelined and have two cycle latency.

4.2 Workloads

The SPEC95 CINT benchmark suite is used for this study. The `test` input files which are provided by SPEC are used. All programs are compiled by GNU GCC (version 2.6.3) with the optimization option, `-O3`. Each program is executed to completion or for the first 100 million instructions. We count only committed instructions.

5 Simulation Results

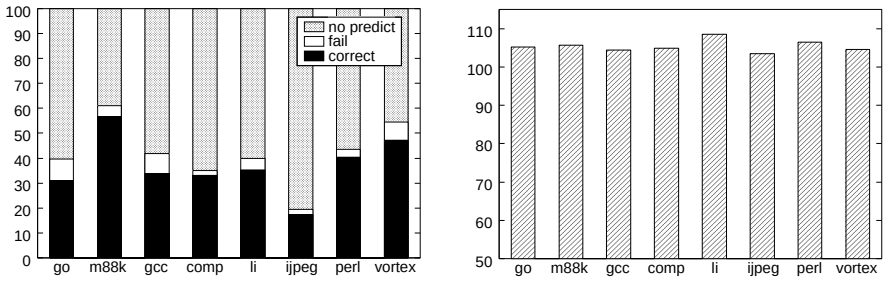
In this section, we present simulation results. First, we evaluate the value predictor. Second, we show how the pipelined RUU affects processor performance. And last, the effect of the decoupling is presented.

5.1 Effect of value prediction

Fig.3(i) shows the characteristics of the value prediction. Each bar is divided into three parts. The bottom part (black) indicates the percentage of the 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. It is observed that 36.8% of dynamic instructions on average are correctly predicted.

Fig.3(ii) presents performance improvement when the value predictor is utilized. We use committed instructions per cycle (IPC) as a metric for evaluating processor performance. For each program, performance of the value prediction model is normalized by that of the baseline model. The processor performance is improved by 5.38% on average. This is a modest improvement but study of value predictors is beyond the scope of this paper.

Table 1 shows the utilization of the instruction window. The columns between 2 and 5 are for the baseline model and the columns between 6 and 9 are for the value prediction model. For each group of four columns, the first two columns indicate the numbers of instructions remaining in the instruction window. They include instructions dispatched to functional units. Note that the instruction window evaluated in this paper is the RUU, which forces each instruction to retain until the instruction has been committed. The remaining two columns show the numbers of instructions waiting for dispatch. They are candidates for



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

Fig. 3. Characteristics of value prediction

Table 1. Instruction window utilization

program	w/o data prediction				with data prediction			
	total		waiting		total		waiting	
	avg	max	avg	max	avg	max	avg	max
099.go	21.6	128	11.0	126	21.5	128	9.6	126
124.m88ksim	18.3	128	8.3	121	16.2	128	5.6	127
126.gcc	21.7	128	11.9	127	21.6	128	11.1	127
129.compress	87.6	128	68.4	121	88.5	128	72.0	121
130.li	17.9	128	8.2	121	17.7	128	6.8	121
132.ijpeg	71.8	128	37.5	121	71.7	128	36.5	121
134.perl	23.8	128	10.9	121	22.0	128	9.1	121
147.vortex	27.8	128	15.0	121	24.8	128	12.8	121

dynamic scheduling. Furthermore, for each group of two columns, the left column presents the average number and the right one presents the maximum number. It is found that when data prediction is used, the average number of waiting instructions decreases except for `129.compress` while that of total instructions changes little. This means that the effective capacity of the instruction window is reduced even if it is compared with the RUU which also holds dispatched instructions. Thus it is necessary to increase the window size to maintain the scheduling ability when data prediction is used. It is also found that the average utilization is significantly small compared with the total capacity. For most of the programs, it is less than 1/3.

5.2 Performance impact of pipelined instruction window

Fig.4 shows how pipelining of RUU affects processor performance. The wakeup and select logic is pipelined by two cycles. For each group of two bars, the left bar indicates the performance of the value prediction model and the right bar indicates that of the processor whose RUU is pipelined. Note that the processor model whose RUU is pipelined also utilizes data prediction. As can be easily seen, the performance degradation is very severe. The improvement gained by data prediction is lost, and furthermore the processor performance is lower than that of the baseline model. For example, comparing the baseline model, performance of `132.ijpeg` is reduced by approximately 25%. In order to fill the performance gap, it is necessary to improve cycle time by 25%. But this is almost impossible to realize. On the other hand, it is an another answer to increase the instruction window size but it will cause further degradation of processor cycle time.

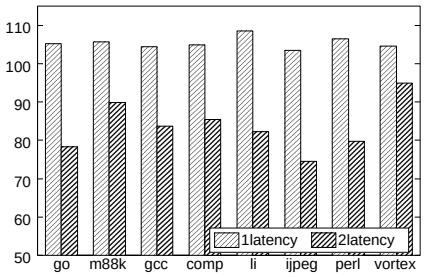


Fig. 4. (%)Performance degradation by pipelined wakeup and select logic

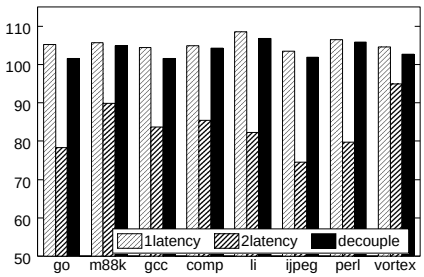


Fig. 5. (%)Performance contribution of decoupled instruction window

5.3 Effect of instruction window decoupling

In this section, we investigate the effect of the decoupling. Since the average utilization of the RUU is less than 1/3 as we have seen in Section 5.1, we decide that the size of the scheduling window is half of the instruction buffer.

Fig.5 presents the performance contribution of decoupled instruction window. For each group of three bars, the first bar (see from left to right) indicates the performance of the value prediction model. The second bar indicates that of the processor whose RUU is pipelined. And the last bar indicates that of the processor which utilizes the decoupled window. It is observed that the performance degradation due to the pipelined instruction window is compensated by its decoupling. For all cases, processor performance is improved over the baseline model. Compared with the value prediction model, performance of the decoupled model is lower. However, it is expected that the clock cycle speed of the decoupling model is faster than that of the value prediction model. Therefore, the processor performance of the decoupling model will be comparable to that of the value prediction model.

Table 2 shows the utilization of the scheduling window and the instruction buffer. The columns 2 and 3 are for the scheduling window and the columns 4 and 5 are for the instruction buffer. For each group of two columns, layout and subject matter of Table 2 are the same as for Table 1. Since the scheduling window size of the decoupling model is smaller than the RUU size of the baseline and the value prediction models, it is natural that the window utilization of the decoupling model is lower than those of the other models. However, it is important to note that the utilization is improved over the value prediction model for `099.go` and `130.li`.

Table 2. Instruction buffer/window utilization

program	buffer		window	
	avg	max	avg	max
099.go	23.9	128	10.0	64
124.m88ksim	17.4	128	5.6	64
126.gcc	21.3	128	9.4	64
129.compress	57.8	128	40.7	64
130.li	18.7	128	7.0	64
132.ijpeg	68.1	128	32.4	64
134.perl	22.8	128	9.1	64
147.vortex	24.3	128	10.8	64

6 Concluding Remarks

In this paper, we have proposed to decouple instruction window. The decoupled window consists of a small instruction window for the dynamic instruction scheduling and a large instruction buffer for the instruction reissue. Since the large buffer is active only when misspeculations are detected, it is expected that the instruction buffer size is increased while the processor cycle time is maintained by pipelining the wakeup and select logic of the buffer. Using a cycle-by-cycle simulator, we have evaluated the decoupled instruction window. While processor performance is significantly diminished by pipelining instruction window, it is compensated by decoupling instruction window. In addition, since it is expected that the clock cycle speed of the decoupling model is fast, there is a possibility to increase processor performance.

Acknowledgment

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

References

1. Akkary,H., Driscoll,M.A.: A dynamic multithreading processor. 31st Int'l Symp. on Microarchitecture (1998)
2. Burger,D., Austin,T.M.: The SimpleScalar tool set, version 2.0. ACM SIGARCH Computer Architecture News, 25(3) (1997)
3. Chrysos,G.Z., Emer,J.S.: Memory dependence prediction using store sets. 25th Int'l Symp. on Computer Architecture (1998)
4. Gabbay,F.: Speculative execution based on value prediction. Technical Report #1080, Dept. of Electrical Eng., Technion (1996)
5. Lipasti,M.H., Wilkerson,C.B., Shen,J.P.: Value locality and load value prediction. Int'l Conf. on Architectural Support for Programming Languages and Operation Systems VII (1996)
6. Palacharla,S., Jouppi,N.P., Smith,J.E: Complexity-effective superscalar processors. 24th Int'l Symp. on Computer Architecture (1997)
7. Rotenberg,E., Jacobson,Q., Sazeidas,Y., Smith,J.: Trace processors. 30th Int'l Symp. on Microarchitecture (1997)
8. Sato,T.: Data dependence speculation using data address prediction and its enhancement with instruction reissue. Euromicro'98 Conf., Workshop on Digital System Design (1998).
9. Sohi,G.S.: Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. IEEE Trans. Comput., 39(3) (1990)
10. Tyson,G. Austin,T.M.: Improving the accuracy and performance of memory communication through renaming. 30th Int'l Symp. on Microarchitecture (1997)