# Analyzing Overhead of Reissued Instructions on Data Speculative Processors

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 investigate the impact of instructions reissued due to mispredicted dataflow speculation on processor performance. Recently, data dependences are studied to be speculatively resolved using value prediction techniques. When a speculation is mispredicted the recovery action for the processor state should be initiated, and the instruction reissue mechanism is utilized for that purpose. The instruction reissue suffers less miss penalties than the instruction squashing used in current generation processors for a mispredicted control speculation, but occurs redundant instruction dispatching, that is multiple copies of an instruction are in flight in functional units. The effectiveness of dataflow speculation is diminished, if reissued instructions cause serious structural hazard. Therefore, we evaluate the impact of the instruction reissue on processor performance using an execution-driven simulator. From the experimentation, overheads due to the instruction reissue is so small that the dataflow speculation contributes to processor performance.*
*Keywords: instruction level parallelism, out-of-order execution, dynamic speculation of data dependence, instruction reissue, value prediction*

## 1 Introduction

Dependences are the obstacles disturbing the processor performance. They include control, name, and data dependences. The control dependence attracts many researchers and is attacked by the techniques such as branch prediction and speculative execution. The name dependence is caused by the resource shortage and can be eliminated using register renaming. However, the data dependence can not be removed by such techniques, as it is called *true dependence*. Hence, the data dependence is the serious obstacle limiting instruction level parallelism (ILP).

Recently, several researchers study data dependence speculation using value prediction [2,7,8,10,13, 15–17]. With the help of a value predictor, data dependences are speculatively resolved and a predicted instruction and its dependent instructions can be dispatched simultaneously. If a speculation is mispredicted, the processor has to recover its state. A straightforward implementation of the recovery mechanism is instruction squashing which is currently used to correct processor state when a speculation of control dependence fails. The instruction squashing is not adequate for the data dependence speculation, because it throws away execution results of instructions which are independent of the misspeculated instructions 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 refetching is very serious. Instruction reissue seems to be one of the promising solutions for this problem [11, 15]. The instruction reissue suffers less miss penalties than the instruction squashing, but occurs redundant instruction dispatching, that is multiple copies of an instruction are in flight in functional units. The effectiveness of dataflow speculation is diminished, if reissued instructions cause serious structural hazard. Therefore, we investigate the impact of the instruction reissue on processor performance using an execution-driven simulator.

The organization of the rest of this paper is as follows. Sections 2 and 3 present the execution model and the evaluation methodology. We explain the microarchitecture of the evaluated processor and benchmark programs used in this study. Sections 4 and 5 show the evaluation results. In Section 6, previously studied related works are surveyed. Finally, our conclusions are presented in Section 7.

## 2 Execution Model

The baseline model is an out-of-order execution superscalar processor based on register update unit (RUU) [14]. Following the discussion explained in [6], we decide the configuration of the baseline processor summarized in Table 1 as the representative of a practical processor in the coming generation. This decision is made because our interest focuses on the usefulness of dataflow speculation utilized in *realistic* processors.

In this section, we explain a microarchitecture attached with data dependence speculation mechanism using data value prediction. We have to consider two components. One is a value predictor, and the other is dynamic instruction scheduling mechanism enable to recover processor state when a misspeculation oc-

Table 1: Processor Configuration

| | |
|---|---|
| Fetch Width | 8 instructions |
| Branch Predictor | 512 entry 2way set-associative BTB, gshare scheme, 12-bit 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 |
| Functional Units | 5 iALU's, 1 iMUL/DIV, 4 Ld/St, 2 fALU's, 2 fMULs, 2 fDIV/SQRT's |
| Latency(total/issue) | iALU 1/1, iMUL 3/1, iDIV 35/35, Ld/St 2/1, fADD 2/1, fMUL 3/1, fDIV/SQRT 6/6 |
| Register Files | 32 32-bit fixed point registers, 32 32-bit 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, write-back, non-blocking load, hit under miss, 6 cycle miss penalty |
| L2 Cache | unified, 256K 4way set-associative, 64 byte blocks, 32 cycle miss penalty |

curs. We use two types of hypothetical predictors, one of which predicts execution results of all types of instructions writing back the results into register files, and the other of which predicts execution results of load instructions. We call the former a general value predictor and the latter a load value predictor. For dynamic instruction scheduling, the instruction reissue mechanism proposed in [10] is used.

## 2.1 Instruction Reissue Mechanism

When data value prediction is performed, the predicted value should be kept in instruction window. When the instruction finishes and an actual value is obtained, the predicted value must be compared with the actual one. If they match, the prediction succeeds and the speculation is useful. Otherwise, the prediction fails and the processor state must be corrected. All instructions dependent upon the mispredicted instructions should be reissued. Figure 1 depicts an instruction window extending the RUU in order to support 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. This means that dataflow graph is kept inside the RUU during a speculation is performed.

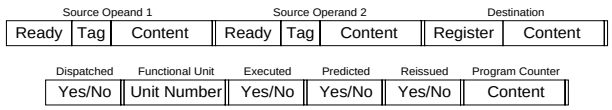


Figure 1: Extended RUU Entry for Value Prediction

An entry of the instruction window consists of two source operand fields, a destination field, a dispatched bit, a functional unit field, an executed bit, a predicted bit, a reissued bit, and a program counter field. If a source operand is not ready, the ready bit is reset to indicate that the source operand is not available[1] and a tag for the operand is obtained. When the operand is ready, the content of the source register is held in the source operand field and the ready bit is set. The destination register number with a renaming tag is held in the destination field, and an execution result is also held in the destination field when the execution completes. The dispatched bit indicates if the instruction is dispatched into a functional unit which is specified by the functional unit field. The executed bit is set when the execution finishes. When the instruction finishes, the execution result is held in the destination field and the executed bit is set. The predicted bit indicates if the corresponding instruction predicts an execution result. Furthermore, the destination field is used to keep the predicted value. Namely, the predicted bit indicates if the content in the destination field is the predicted value. If one of the executed and predicted bits is set, the following instructions which are dependent upon the instruction can obtain a source operand. The reissued bit indicates if the corresponding instruction is reissued. Lastly, the program counter field is used for the correction of misprediction and precise interrupts.

When a value prediction is initiated, the predicted bit is set and the predicted value is held in the content slot of the destination field. When an instruction produces an execution result, its destination tag is broadcasted as the original RUU does. Simultaneously, the signal indicating if a prediction is correct, is also broadcasted when the predicted bit is set. Here, we call this signal the misprediction signal. When the prediction is not correct, the misprediction signal is asserted. The correctness of the prediction is decided by comparing the predicted and actual values. In the case that the misprediction signal is not asserted, the following instructions obtain their source operands if their source and the broadcasted destination tags match. The instructions whose source operands are obtained are ready to be dispatched. In the case that the misprediction signal is asserted, the following instructions should be invalidated and reissued if their source and the broadcasted destination tags match and if the corresponding dispatched bit is set. These instructions have been already dispatched using a wrong operand value. The corresponding dispatched and executed bits should be reset, and the reissued bit is set. The mispredicted instruction has produced

[1] In [14], the ready bit is set if a source operand is not ready. However, in this paper, we set the ready bit when the source operand is obtained for easy understanding.

the actual value, and thus the reissued instructions have obtained the source operands and are ready to be dispatched. When a reissued instruction finishes, the mispredicted signal is asserted. Namely, the mispredicted signal indicates if the mispredicted or reissued instructions have finished their execution. The following process is as same as above. The instructions dependent on the reissued instruction should also be reissued. In this manner, all instructions dependent upon a mispredicted instruction are searched and reissued gradually.

This scheme introduces only one signal into the dynamic instruction scheduling mechanism. The signal decides the action resulted from the associative search which examines if the source and destination tags match. The signal is asserted when mispredicted or reissued instruction finishes. If the signal is not asserted, the dynamic instruction scheduling mechanism conducts usual process. Otherwise, it detects instructions which should be reissued. This mechanism is very simple, and hence it requires only slight hardware overhead and it does not increase the processor cycle time. This is the reason why we propose the scheme. It is logically possible to search and detect in parallel all instructions dependent on a mispredicted instruction at a time. However, it requires expensive hardware to implement such a mechanism without causing the increase of the processor cycle time, which degrades the processor performance.

It is true that an instruction might be dispatched before it is detected to be reissued even if a misprediction has been detected, and hence it is also true that the effectiveness of functional units usage might be reduced. However, we expect that such cases were very rare due to the following two reasons. One is that the predicted instruction is statically located far away from its dependent instructions by sophisticated compilers so that the outcome of the predicted instruction is ready when the dependent ones are issued [3]. The other is the limited number of functional units. Namely, instructions which are ready to be dispatched using predicted values are not always be dispatched before their source operands are ready. Therefore, we do not worry about the performance degradation caused by the redundant dispatching and we evaluate this redundant instruction dispatching in this paper.

## 2.2 Dynamic Scheduling Example

An example is useful to understand the process of dynamic instruction scheduling using the RUU. Figure 2 shows an instruction sequence. For easy understanding, each operation of $f_1 - f_4$ is assumed to refer only 1 source operand. The execution latency of each instruction is 1 cycle except that that of instruction `I1` is 4. Instruction `I1` is such an instruction as a load

`I1`: $r11 \leftarrow f_1(r1)$
`I2`: $r12 \leftarrow f_2(r2)$
`I3`: $r13 \leftarrow f_3(r11)$
`I4`: $r14 \leftarrow f_4(r13)$

Figure 2: Instruction Sequence Example

instruction suffering a data cache miss. There exist two data dependences. One is between instructions `I1` and `I3`, and the other is between instructions `I3` and `I4`. This sequence finishes in seven cycles when any data dependences are not speculated.

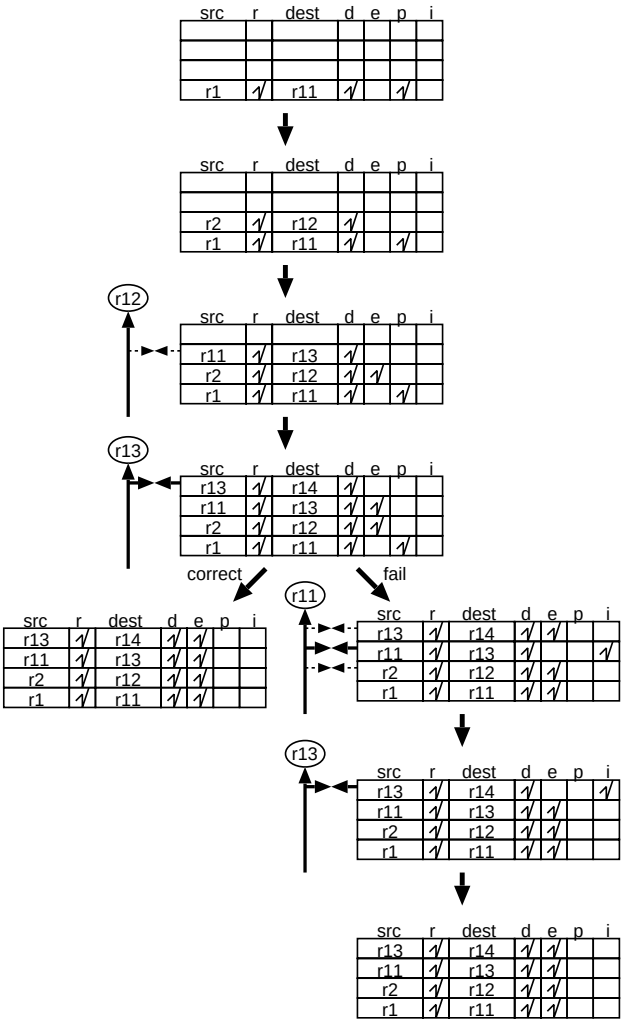


Figure 3: Instruction Reissue with RUU

Figure 3 illustrates an example of the instruction reissue. In order to make explanations simple, the processor is single-issued and the dispatch width is one. We omit the instruction committing process. It is assumed that the destination register number with the renaming tag is same to the architectural register number. It is also assumed that registers $r1$ and $r2$ are available before instruction `I1` is issued. It is assumed that only instruction `I1` can predict a data value and its dependent instructions speculate data dependence. Now, let us explain dynamic scheduling using the RUU. First, `I1` is issued and dispatched. The associated dispatched bit (`d`) is set. `I1` predicts the value of $r11$ and sets the predicted bit (`p`). At

next cycle, `I2` is issued and dispatched. Next, `I3` is issued. `I3` can be dispatched since the predicted bit of `I1` which produces the $r11$ value is set. `I2` produces an execution result $r12$, but it is not used. The dashed arc indicates that the destination and source tags do not match. `I2` sets the executed bit (`e`). At the forth cycle, `I4` is issued and dispatched, since `I3` has finished the execution and $r13$ is ready. The thick arc indicates that the destination and source tags match. At next cycle, there are two cases. One is that the prediction of `I1` is correct, and the other is that `I1` fails the prediction. If the prediction is correct, `I1` resets the predicted bit and sets the executed bit. `I4` has finished its execution and also sets the executed bit. The example instruction sequence has finished in five cycles. As mentioned above, the example sequence finishes in seven cycles when any data dependences are not speculated. By comparison with this, the execution cycle is reduced by two cycles when the prediction is correct. On the other hand, if the prediction fails, the instruction reissue process is initiated. `I3` should be reissued since it has been dispatched using a wrong operand. The corresponding dispatched and executed bits are reset and the reissued bit (`i`) is set. The source operand for `I3` is ready and thus `I3` is ready to be dispatched. Since `I3` is the only instruction ready to be dispatched, it is dispatched into a functional unit and sets the dispatched bit. Next, `I3` finishes its execution, sets the executed bit, and resets the reissued bit after asserting the misprediction signal. `I4` is detected to be reissued since it has been dispatched using an operand produced by the instruction which is dependent upon the mispredicted instruction. The dispatched and executed bits corresponding `I4` are reset and the reissued bit is set. Because there are not any instructions except `I4`, which are ready to be dispatched, `I4` is dispatched and sets the dispatched bit. And lastly at the next cycle, `I4` finishes, sets the executed bit, and resets the reissued bit. The number of the execution cycle when the misprediction occurs is seven, which is as same as that in the case when any data dependences are not speculated. Fortunately in this example, misprediction does not occur any miss penalties. In actual computing, there should be cases when reissued instructions can not be dispatched immediately after detected to be reissued, and hence the processor may suffer from some miss penalties.

# 3 Evaluation Methodology

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

## 3.1 Simulation Environment

In order to evaluate the effect of speculations, an execution-driven simulation is more desirable than a trace-based simulation, since trace-based simulators can not simulate misspeculated instructions. We evaluated the effect of the proposed mechanism by using SimpleScalar tool set [1]. The SimpleScalar architecture is based on MIPS architecture, and a fully execution-driven and cycle-by-cycle simulator is executed on a SPARCstation. The SimpleScalar simulator implements the original RUU [14], and we have modified it in order to support the instruction reissue mechanism described in the previous section. The dynamic instruction scheduler is implemented in detail and the reissued instructions might be dispatched several times. If an instruction were redundantly dispatched as explained in the previous section, multiple copies of the instruction would be in flight in functional units.

## 3.2 Workload

The SPEC95 benchmark suite is used for this study. The `test` input files which are provided by SPEC are used. We used the object files provided by University of Wisconsin Madison [1]. Each program was executed to completion or for the first 100 million instructions. We counted only committed instructions.

# 4 General Value Predictor Results

This section presents experimental results when a general predictor is utilized. First, we present performance improvement. Next, we show the number of useless instructions caused by mispredicted speculations and evaluate how they affect processor performance. For measuring performance, we used the committed instruction per cycle (IPC). Only useful instructions are considered for counting the IPC. We define the performance improvement rate as the increased IPC over the IPC of the baseline model.

We evaluate two cases. In the first case, the total prediction rate is fixed and the prediction accuracy is varied. We define the total prediction rate as the number of all predicted instructions including mispredicted ones over the all general instructions, and the prediction accuracy as the number of instructions whose outcome is correctly predicted over the total number of the predicted general instructions. In the second case, the prediction rate is fixed and the prediction accuracy is varied. We define the prediction rate as the number of the correctly predicted instructions over the all general instructions.

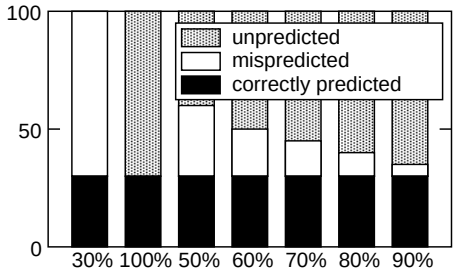


Figure 4: Prediction Accuracy and Prediction Rate

The relationship between the prediction accuracy, the prediction rate, and the total prediction rate is depicted in Figure 4, where the prediction rate is 30%. The bars from left to right indicate the relationship when the prediction accuracy is 30%, 100%, 50%, 60%, 70%, 80%, and 90%, respectively. Each bar is divided into three parts. The bottom part (black) indicates the percentage of the instruction which is correctly

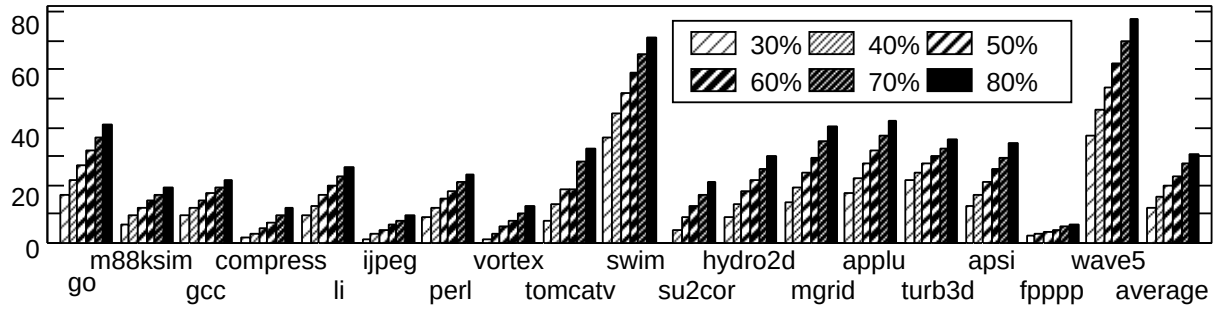


Figure 5: (%)Performance Improvement vs. Prediction Accuracy (General)

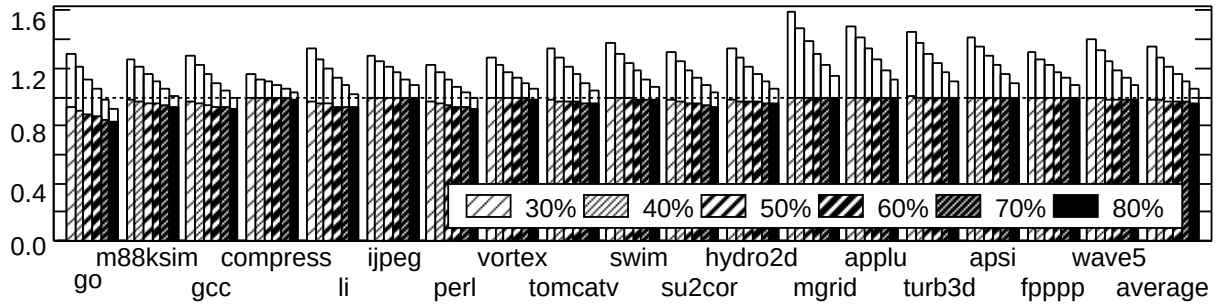


Figure 6: Total Instruction Count vs. Prediction Accuracy (General)

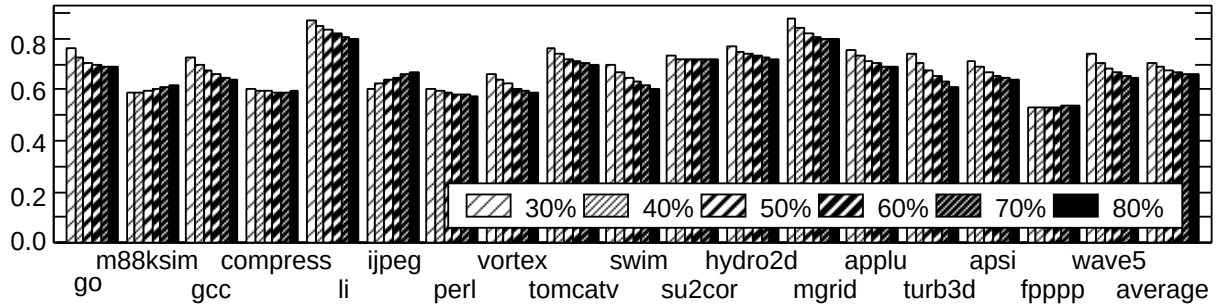


Figure 7: Reissued Instruction Rate vs. Prediction Accuracy (General)

predicted. The middle part (white) indicates the percentage that is mispredicted. And the top part (gray) indicates the percentage that is not predicted. Therefore, the first bar explains that the total prediction rate is 100% and the prediction accuracy is 30%. This case will be evaluated in Section 4.1. The second bar shows that the prediction accuracy is 100% and the prediction rate is 30%. The cases of the remaining bars will be evaluated in Section 4.2.

### 4.1 Impact of Prediction Accuracy

First, we evaluate the general predictor under the situation that the total prediction rate is 100% and the prediction accuracy is varied between 30% and 80%. Namely, all general instructions are candidates for value prediction.

Figure 5 shows processor performance improvement. For each group of six bars, the bars from left to right indicate the results when value prediction accuracy is 30%, 40%, 50%, 60%, 70%, and 80%, respectively. Naturally, the processor performance increases as the prediction accuracy becomes higher. When prediction accuracy is 80%, processor performance is improved by 31.0% on average with a maximum of 75.6%. It is surprising that processor performance is improved by 12.0% on average when the prediction accuracy is only 30%. This means that 30% of successful prediction compensates the penalty caused by 70% of misprediction and furthermore contributes to processor performance. It can be seen that floating-pont programs are more improved in general than integer programs. That is, value prediction is more effective to floating-point programs than integer programs.

Figure 6 shows the executed instruction count, which includes squashed and reissued instructions[2]. The count is normalized by that of the baseline model without a value predictor. Layout of Figure 6 is the

[2]The instruction squashing is performed when a control dependence speculation fails.

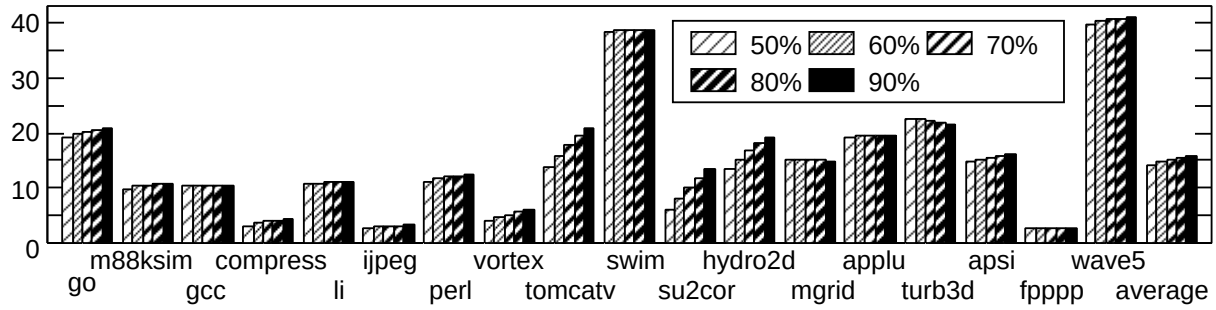


Figure 8: (%)Performance Improvement vs. Total Prediction Rate (General)

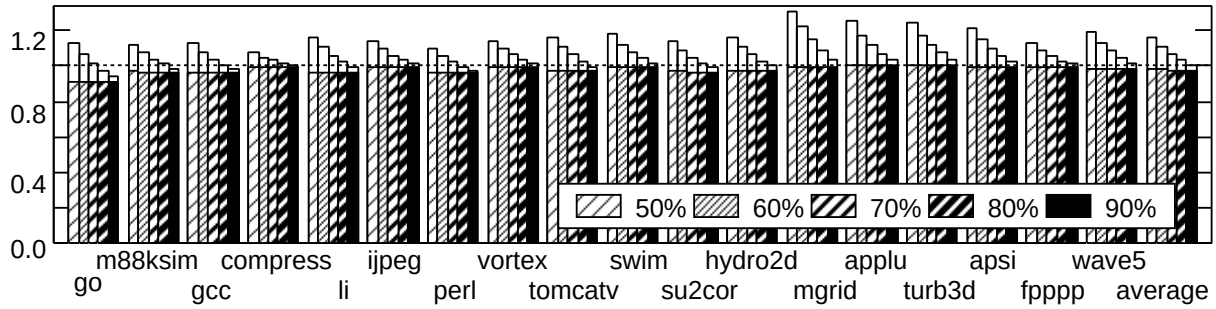


Figure 9: Total Instruction Count vs. Total Prediction Rate (General)

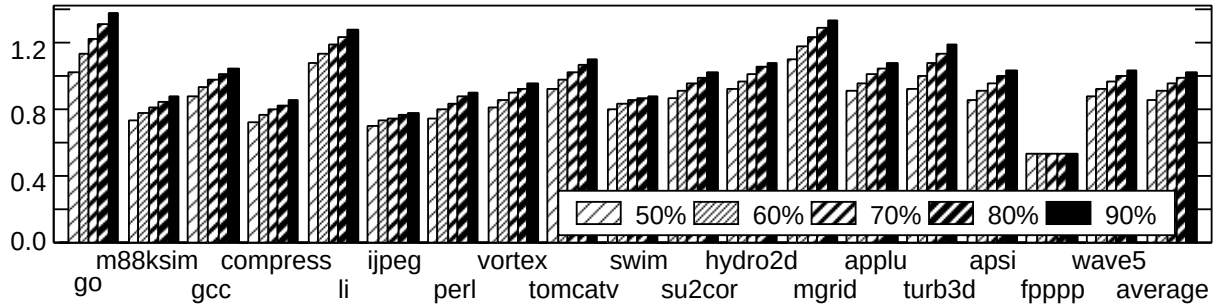


Figure 10: Reissued Instruction Rate vs. Total Prediction Rate (General)

same as for Figure 5. Each bar is divided into two regions. The top region (white) indicates the count of the reissued instructions. The bottom region indicates that of the remaining instructions. Note that the reissued instructions does not consume instruction fetch bandwidth. The rate of the fetched instructions depicted as the bottom region is lower than 1.0 in most cases. Thus, the instruction fetch traffic is not affected by the reissued instructions.

Figure 7 shows reissued instruction rate. We define the reissued instruction rate as the number of the reissued instructions over the number of the mispredicted instructions. Layout of Figure 7 is the same as for Figure 5. Before performing experimental simulations, we expected that the number of the reissued instructions was larger than that of the mispredicted ones, since the number of instructions dependent upon a mispredicted instruction seemed to be more than one. However, it is not. As can be seen in Figure 7, the reissued instruction rate is smaller than 0.9 for all programs. This is because the instructions which are ready to be dispatched using predicted values can obtain their source operands before dispatched into functional units. That is, many of the speculations which are to fail are not invoked due to structural hazard.

### 4.2 Impact of Total Prediction Rate

In this subsection, we investigate the impact of the reissued instructions under the condition that the prediction rate is fixed and the prediction accuracy is varied, and hence the total prediction rate is varied according to the prediction accuracy. We try to make the total prediction rate as small as possible. Therefore, the prediction accuracy should be relatively high. The decision is made since we believe that the predictors whose total prediction rate is large are much more difficult to be implemented than those whose prediction accuracy is high. The predictability is different according to individual instructions. Since misspeculations cause penalties, easily predictable instructions

only should be predicted. This means the reduction of the total prediction rate.

The prediction rate is fixed to be 30% and the prediction accuracy is varied between 50% and 90%. The accuracy cannot be less than 30% since the prediction rate is 30%. According to the prediction accuracy, the total prediction rate is varied between 33.3% and 60.0%. The reason why we decided the prediction rate presented above is as follows. In the previous subsection, the performance improvement rate is more than 10%, when the prediction accuracy is 30% and the prediction rate is also 30% since total prediction rate is 100%. We think the performance improvement of over 10% up is enough effective to be considered implementation. Therefore, we decided the prediction rate as above.

Figure 8 presents processor performance improvement rate when the prediction rate is fixed to be 30%. For each group of five bars, the bars from left to right indicate the results when the prediction accuracy is 50%, 60%, 70%, 80%, and 90%, respectively. The corresponding total prediction rate is 60.0%, 50.0%, 42.9%, 37.5%, and 33.3%, respectively. As can be seen from Figure 8, the difference of the improvement rate according to the prediction accuracy is very small. This implies that the dominant of the performance improvement is not the prediction accuracy but the prediction rate. For `turb3d`, the performance improvement rate decreases as the prediction accuracy increases. This is due to the unexpected characteristics of the hypothetical value predictor. The predicted instructions are different according to simulation runs whose configurations are different. Thus, a instruction predicted in a simulation run may be very effective to improve processor performance, and the other instruction predicted in the other simulation run may not. Furthermore, a predicted instruction dose not always speculate data dependence due to structural hazard. Therefore, there are the cases in which the performance improvement rate decreases as the prediction accuracy increases.

Figure 9 shows the executed instruction count. The count is normalized by that of the baseline model without a value predictor. Layout of Figure 9 is the same as for Figure 8. As same with Figure 6, each bar is divided into two regions. Also in this case, the number of the fetched instructions of the evaluated model is lower than that of the baseline model. Hence, the instruction fetch traffic is not affected by the reissued instructions.

Figure 10 shows reissued instruction rate. Layout of Figure 10 is the same as for Figure 8. Different from Figure 7, the reissued instruction rate is larger than 1.0 for several programs. It can be also seen that the reissued instruction rate for most programs increases as the prediction accuracy increases, while in Figure 7 the reissued instruction rate for most programs decreases as the prediction accuracy increases. This is due to the number of mispredicted instructions executed. As the mispredicted instructions dispatched into functional units are reduced, the effectiveness of functional units usage improves and the number of instructions dependent upon an instruction increases. Thus, the chain of dispatched instructions which are dependent upon a mispredicted instruction becomes longer. This means the increase of the reissued instruction rate.

# 5 Load Value Predictor Results

In this section, we present the simulation results of the load value predictor. As same with the previous section, the processor performance improvement and the instruction overhead caused by the instruction reissue are investigated under the two situations evaluated in the previous section. In this section, the total prediction rate is defined as the number of all predicted instructions including mispredicted ones over the all load instructions, and the prediction rate is defined as the number of the correctly predicted instructions over the all load instructions.

## 5.1 Impact of Prediction Accuracy

Figure 11 shows performance improvement rate when the prediction accuracy is varied. Layout and subject matter of Figure 11 are the same as for Figure 5. It can be seen that the improvement rate is significantly reduced when candidates for prediction are limited to load instructions. The contribution of the value prediction is reduced to be half of the general instruction case for integer programs. The processor improvement rate for floating-point programs is more severely decreased than the integer program case. It is less than one forth of that for the general instruction case. Therefore, value predictors are required highly accurate predictions of arithmetic instruction results in order to contribute to processor performance when floating-point programs are executed. The performance improvement rate is 10.0% on average with the maximum of 28.1% when the prediction accuracy is 80%.

Figure 12 shows the executed instruction count. Layout and subject matter of Figure 12 are the same as for Figure 6. As same with the general instruction case, the number of the fetched instructions is lower than that of the baseline model. This also confirms that the instruction fetch traffic is not affected by the reissued instructions.

Figure 13 shows reissued instruction rate. Layout and subject matter of Figure 13 are the same as for Figure 7. Different from the general instruction case, the reissued instruction rate is larger than 1.0 for several programs. The reason is as follows. The latency for load instructions are larger than that for the other instructions with the consideration of data cache misses. Thus, the number of dispatched instructions during the period before a speculated load instruction produces the actual results after the load instruction is dispatched. This means the number of instructions dependent upon the speculated load instruction increases.

## 5.2 Impact of Total Prediction Rate

Due to the same reason explained in the previous section, the prediction rate is fixed to be 80% and the prediction accuracy is varied between 82% and 90%. The accuracy cannot be less than 80% since the prediction rate is 80%. Figure 14 presents the proces-

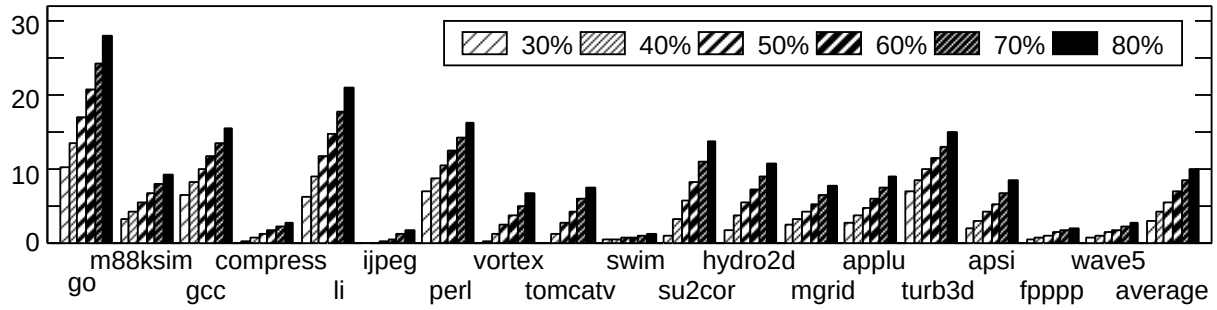


Figure 11: (%)Performance Improvement vs. Prediction Accuracy (Load)

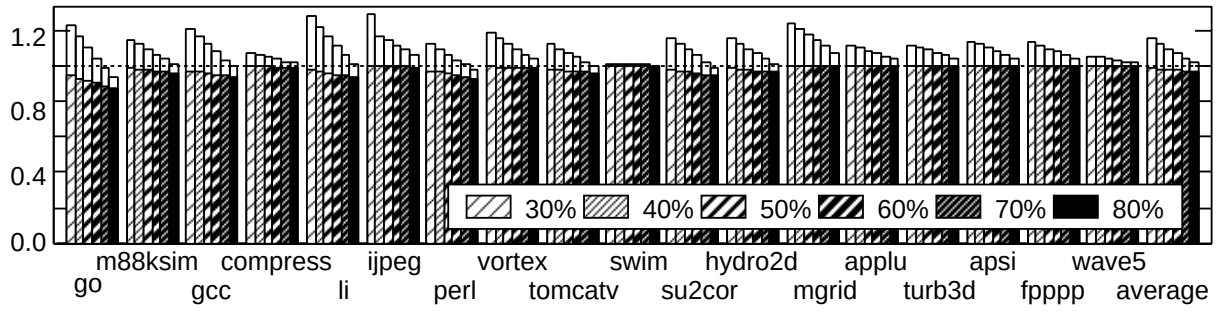


Figure 12: Total Instruction Count vs. Prediction Accuracy (Load)

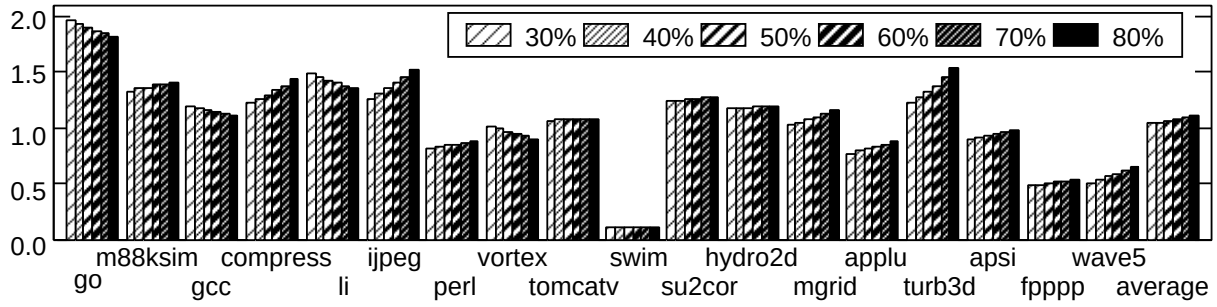


Figure 13: Reissued Instruction Rate vs. Prediction Accuracy (Load)

sor performance improvement rate. For each group of five bars, the bars from left to right indicate the results when the prediction accuracy is 82%, 84%, 86%, 88%, and 90%, respectively. The corresponding total prediction rate is 97.6%, 95.2%, 93.0%, 90.9%, and 88.9%, respectively. As same with the general instruction case, the difference of the improvement rate according to the prediction accuracy is very small. The average performance improvement rate is 10.0% when the prediction accuracy is 82% and 10.2% when the accuracy is 90%. Thus, it is confirmed again that the dominant of the improvement rate is not the prediction accuracy but the prediction rate.

Figure 15 shows the executed instruction count. Layout of Figure 15 is the same as for Figure 14. As same with the previous results, the number of fetched instructions are lower than that of the baseline model and hence the instruction fetch traffic is not affected by the reissued instructions.

Figure 16 shows reissued instruction rate. Layout of Figure 16 is the same as for Figure 14. The tendency resembles with the results presented in the previous section.

## 6 Related Work

This section surveys the related works.

Lipasti [7] introduced the instruction reissue concept. Instructions dependent upon a predicted instruction are forced to retain in the reservation station. 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 the reservation station. If the prediction fails, all dependent instructions are concurrently invalidated and reissued. However, he only has proposed the concept of the instruction reissue. He has not presented any practical implementation of the scheme. If the scheme were implemented, the processor cycle time would increase since it would be very difficult to find in parallel all dependent instructions using moderate hardware cost. Gonzalez et al. [4] also proposed the instruction reissue

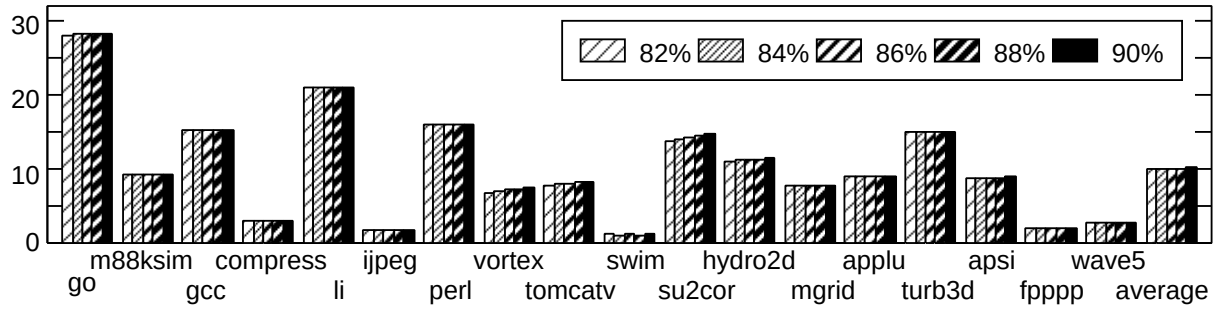


Figure 14: (%)Performance Improvement vs. Total Prediction Rate (Load)

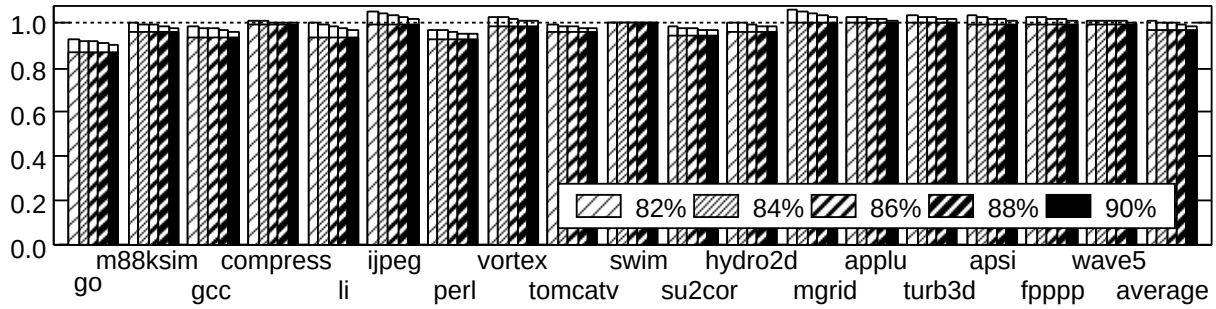


Figure 15: Total Instruction Count vs. Total Prediction Rate (Load)

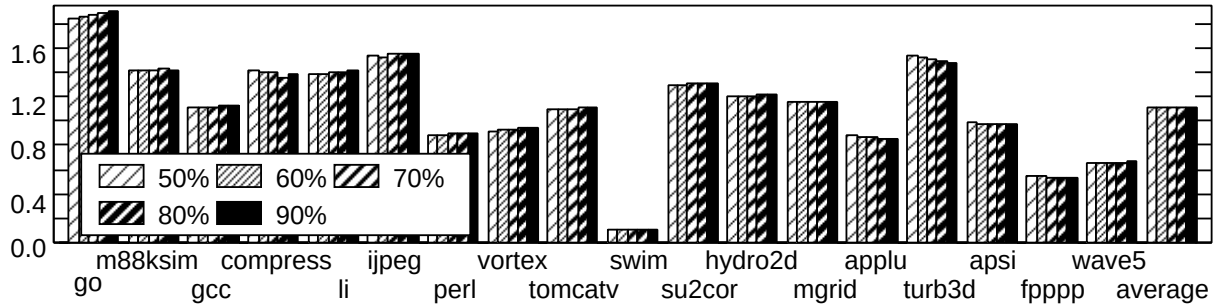


Figure 16: Reissued Instruction Rate vs. Total Prediction Rate (Load)

scheme, which is very similar to [7]. When a prediction is verified, all dependent instructions are committed if the prediction is correct. Otherwise, they are reissued. Gonzalez et al. also have not presented any practical mechanism but only described the concept.

Gonzalez et al. [5] have evaluated the potential of data value speculation and found that the speedup of processor performance is a linear function of the prediction accuracy. However, the simulator they used is a trace-based, and thus stress on functional units caused by instruction reissue is not considered. Gabbay et al. [3] have explored the limitation of value prediction in realistic processors and showed that the instruction fetch bandwidth and the issue rate have a significant impact on effectiveness of the prediction. They also used a trace-based simulator, and hence ignored the structural hazard caused by instruction reissue. We have also found that wide instruction fetch bandwidth enhances data dependence speculation [12] but the recovery mechanism for misspeculations used in the evaluated model is not the instruction reissue but the instruction squashing.

Tyson et al. [15] have evaluated the usefulness of the instruction reissue. They have proposed a streamlining load value scheme and found that the instruction reissue proposed by Lipasti [7] can improve processor performance even for the applications whose performance is degraded when the instruction squashing is used. Rotenberg et al. [9] have investigated an instruction reissue scheme in Trace Processor architecture and found it is useful for dataflow speculation. However, they do not evaluated it in superscalar processors which are currently in mainstream.

In this paper, we have evaluated the effect of the instruction reissue on superscalar processors using an execution-driven simulator.

## 7 Concluding Remarks

In this paper, we have investigated the impact of the reissued instructions on the effectiveness of

dataflow speculation. The effectiveness of dataflow speculation is diminished, if reissued instructions cause serious structural hazard. Using an execution-driven simulator, we have evaluated the performance improvement rate and the overhead caused by the reissued instructions. From the experimental results, the total number of executed instructions is significantly increased by the instruction reissue, but the processor performance is not degraded. This is because the increased instructions executed do not waste the instruction fetch bandwidth while the instruction fetching efficiency affects processor performance severely [6]. The reissued instructions increase the resource shortage of the instruction window and functional unit, but the performance is not degraded for almost all the cases.

In this paper, we have considered only one processor configuration, and it is true that the impact of the instruction reissue on processor performance becomes different when the configuration changes. However, the processor configuration used in this paper is relatively realistic for the coming generation processors, and hence we hope that the evaluation results presented in this paper are useful for studying data dependence speculation.

Future study dealing with the dataflow speculation will investigate the requirements under the situation that the hardware of the value predictor is constrained. For example, it is interesting to limit the number of predictions which are initiated in parallel. Other hardware resources should be also evaluated. We are now analyzing the hardware constraints such as the number of functional units and the instruction issue width.

## Acknowledgment

The author is grateful to Dr. Mitsuo Saito and Dr. Shigeru Tanaka for their continuous encouragements.

## References

[1] D.Burger and T.M.Austin, "The SimpleScalar tool set, version 2.0," ACM SIGARCH Computer Architecture News, vol.25, no.3, pp.13-25, 1997.

[2] F.Gabbay and A.Mendelson, "Can program profiling support value prediction?," Proc. of 30th Ann. Int'l Symp. on Microarchitecture, pp.270-280, 1997.

[3] F.Gabbay and A.Mendelson, "The effect of instruction fetch bandwidth on value prediction," 25th Ann. Int'l Symp. on Computer Architecture, 1998.

[4] J.Gonzalez and A.Gonzalez, "Speculative execution via address prediction and data prefetching," Proc. of 11th Int'l Conf. on Supercomputing, pp.196-203, 1997.

[5] J.Gonzalez and A.Gonzalez, "The potential of data value speculation to boost ILP," 12th Int'l Conf. on Supercomputing, 1998.

[6] S.Jourdan, P.Sainrat, and D.Litaize, "Exploring configuration of functional units in an out-of-order superscalar processor," Proc. of 22nd Ann. Int'l Symp. on Computer Architecture, pp.117-125, 1995.

[7] M.H.Lipasti, "Value locality and speculative execution," Ph.D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, April 1997.

[8] A.I.Moshovos and G.S.Sohi, "Streamlining inter-operation memory communication via data dependence prediction," Proc. of 30th Ann. Int'l Symp. on Microarchitecture, pp.235-245, 1997.

[9] E.Rotenberg, Q.Jacobson, Y.Sazeidas, and J.Smith, "Trace processors," Proc. of 30th Ann. Int'l Symp. on Microarchitecture, pp.138-148, 1997.

[10] T.Sato, "Load value prediction using reference address renaming," 10th Joint Symp. on Parallel Processing, pp.15-22, June 1998 (In Japanese).

[11] T.Sato, "Data dependence speculation using data address prediction and its enhancement with instruction reissue," Euromicro'98 Conf., Workshop on Digital System Design: Architectures, Methods and Tools, August 1998.

[12] T.Sato, "A microprocessor architecture utilizing histories of dynamic sequences saved in distributed memories," IEICE Trans. on Electronics, vol.E81-C, no.9, September 1998.

[13] Y.Sazeides and J.E.Smith, "The predictability of data value," Proc. of 30th Ann. Int'l Symp. on Microarchitecture, pp.248-258, 1997.

[14] G.S.Sohi, "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers," IEEE Trans. on Computers, vol.39, no.3, pp.349-359, 1990.

[15] G.Tyson and T.M.Austin, "Improving the accuracy and performance of memory communication through renaming," Proc. of 30th Ann. Int'l Symp. on Microarchitecture, 218-227, 1997.

[16] K.Wang and M.Franklin, "Highly accurate data value prediction using hybrid predictors," Proc. of 30th Ann. Int'l Symp. on Microarchitecture, pp.281-290, 1997.

[17] L.Widigen, E.Sowadsky, and K.McGrath, "Eliminating operand read latency," ACM SIGARCH Computer Architecture News, vol.24, no.5, pp.18-22, 1996.