Implementing Fast Recovery for Register Alias Table in Out-of-order Processors

Jianqing Xiao, Mian Lou, Wei Li, and Yuanyuan Cui
Department of Computer Research and Development
Xi'an Microelectronics Technology Institute
Xi'an, China
E-mail: xjq_career@126.com

***Abstract*—Register renaming is an indispensable technique to cope with false data dependencies in out-of-order processors. The critical component for performing register renaming is a register alias table (RAT), which maintains the mappings between architecture and physical registers. Unfortunately, a potential misprediction may seriously slower the processor execution, because new instructions are not allowed to be renamed until the RAT has been restored to the previous correct status. Therefore, a fast RAT recovery is necessary to sustain high performance. As instruction windows size increases, the traditional recovery mechanisms such as using retirement map table and history buffer become too slow. In order to instantly restore RAT from a misprediction, the embedded checkpoint method is introduced but too costly due to creating a large number of checkpoints. In this paper, we propose a selective checkpoint policy which is based on branch confidence and decides when to assign a new checkpoint. Experimental results show that for a 2048-entry large instruction window, our proposal only uses 8 checkpoints to implement fast RAT recovery, reducing the misprediction overhead to just 3% of the ideal model.**

Keywords-register renaming; register alias table; checkpoint; misprediction recovery

I. INTRODUCTION

Modern high-performance microprocessors commonly implement out-of-order and speculative execution to increase level parallelism. In this execution mechanism, register renaming techniques are required to cope with false data dependencies such as write after read (WAR) and write after write (WAW). As the core of register renaming, Register Alias Table (RAT) maintains the proper mappings between architectural and physical registers, so that each instruction can obtain its register address by performing a RAT lookup. At the same time, in order to reduce control hazards, branch predictors are used in almost all of the deep pipelining processors [1, 2]. However, if a misprediction occurs, the processor may suffer serious penalties for misprediction recovery, especially in RAT recovery, because the correct path instructions cannot be renamed until the RAT corresponding to the mispredicted branch is restored. Apparently, implementing fast recovery from misprediction paths is quite important to processor performance. Some literatures even strongly suggest that reducing misprediction penalty is a better alternative solution than increasing prediction accuracy [3].

So far, several methods have been proposed to restore the RAT from branch misprediction. One is to start with current retirement map table [4] and walk from the head of the reorder buffer towards the mispredicted branch, thus incrementally incorporating the valid RAT rename information. Another is to use a history buffer that stores the speculative map table updates since the mispredicted branch was dispatched, thus incrementally obliterating the overwritten maps in RAT. However, both the two methods could contribute to a significant branch misprediction penalty, because there may be many instructions in reorder buffer or history buffer prior to the mispredicted branch and they need to be serially processed. To support instantaneous recovery from mispredictions, a third method is to embed global checkpoints (GCs) into the RAT [5, 6], which creates checkpoints periodically either at every branch or every few cycles, and use them to restore the RAT on a branch misprediction. The periodic checkpoint method is quite fast, but impractical to implement due to the requirement for a large number of checkpoints. In theory, the number of checkpoints is determined by the number of unresolved branches allowed in the instruction window. If only a few checkpoints are provided, it would perform worse than the other two methods in dealing with conditional branches. As instruction window sizes increase, these traditional recovery mechanisms are either too costly or may become too slow.

In this paper, we focus on implementing fast RAT recovery with only few checkpoints, and propose a selective checkpoint policy which uses a branch confidence estimator to determine whether to allocate a new checkpoint on current branch or not. Experimental results show that our proposal performs best among all the methods above.

The paper is organized as follows. First we introduce the implementation structure of checkpointed RAT in Section II and then depict the proposed selective checkpoint policy in Section III. Section IV gives the simulation methodology and experimental results. Related work is discussed in Section V and we conclude in Section VI.

II. THE STRUCTURE OF CHECKPOINTED RAT

Traditionally, the RAT implementations have used two types of memory structures: static RAM or CAM. Both of them present advantages and shortcomings, and neither shows a dominant trend. For example, the MIPS R10000 [6] and the Pentium4 [4] use the RAM structure, while the Alpha 21264 [5] and Power4 [7] use the CAM approach.

978-1-4799-2716-6/13/$31.00 ©2013 IEEE

Figure 1(a) depicts the RAT structure based on RAM. As we can see, the number of RAT entries is equal to the number of architectural registers. Each entry contains a corresponding physical register identifier. To find or update the current mapping information for an architectural register, the corresponding RAT entry is read out or written into respectively, using the architectural register identifier as the access index. A new global checkpoint containing current mappings is created by copying the whole RAT content to a shadow table which is so-called GC. Therefore, every GC contains as many bits as the RAT.

In the CAM-based RAT structure, shown in Figure 1(b), the number of RAT entries is equal to the number of physical registers. Each entry contains the corresponding architectural register identifier and a valid bit. To rename a source architectural register, its identifier as the lookup keyword is broadcasted to all entries and compared with them, and finally the corresponding physical register index is obtained from a matching entry. To allocate a new physical register, the architectural register identifier is written into the associated entry and the valid bit of the old mapping is cleared, which makes sure that at any time only one entry for each architectural register has the valid set. In this structure, every GC only contains the valid bits of the RAT.

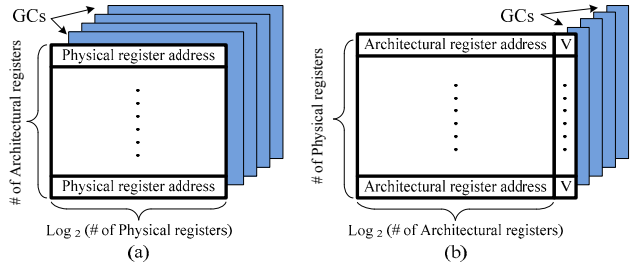

Figure 1. Structure of RAT: (a) SRAM-based, (b) CAM-based.

Regardless of the approach used, the RAT needs to incorporate more GCs to sustain performance. Unfortunately, RAT latency, area and energy will significantly increase with the number of GCs. Therefore, using only few checkpoints to implement fast RAT recovery from a misprediction become the key issue [8].

III. SELECTIVE CHECKPOINT POLICY

A. Branch Confidence Estimator

In order to reduce useless checkpoints, we propose a selective low-confidence branch checkpoint policy, which creates RAT checkpoints just on the branches with a high misprediction probability. In this policy, a branch confidence estimator is required firstly. Our estimator uses a saturation counter table, which is indexed by the *xor* result of the branch address and global branch history, as the same way the PHT is indexed in *gshare* predictor. Consequently, the confidence estimator table (CET) is implemented together with the PHT, as shown in Figure 2. If final prediction is correct, the CET counter is incremented. However, a misprediction will stoutly reset the counter to zero. Eventually, a counter with the maximal value indicates the corresponding branch is high confidence while the remaining values represent a low-confidence branch.

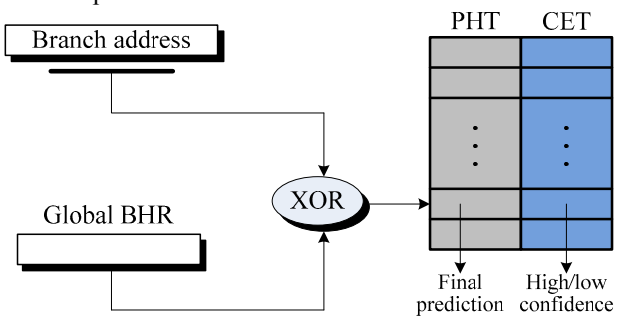

Figure 2. Gshare predictor with confidence estimator table embedded.

B. Checkpoint Creation

According to our selective policy, a low or high confidence branch determines whether a new checkpoint will be created or not, thus reducing the recovery overhead effectively. Besides the primary design motivation above, in fact, we need to take the following instances into account:

1) Creating a checkpoint at the first branch

In current instruction window, the executing processor eventually comes across an unresolved branch, and then enters "checkpoint mode" from previous "conventional mode". We must create a checkpoint at this branch no matter it is low or high confidence, as the start point used to recover processor status when a branch is found mispredicted.

2) Limiting the maximal branches for per checkpoint

Since checkpoints are not created at every branch, a branch misprediction may result in execution restarting from the nearest checkpoint prior to the mispredicted branch, though the branch is high confidence. This causes the valid instructions between the checkpoint instruction and the mispredicted branch to be re-executed. We call this re-execution overhead the checkpoint overhead. To minimize this overhead, an additional checkpoint should be created when the limit number of branches attached to a particular checkpoint is exceeded.

3) Assigning the last available checkpoint

Naturally, only few checkpoints are supported. Once the last available checkpoint is allocated, all branches that have sequentially entered the current instruction window are assigned to the last checkpoint. At this point, the processor continues fetch, dispatch and execution without any stall. Notice that this unbounded assignment occurs just when there is no available checkpoint. On becoming available, a new checkpoint will be allocated immediately if the number of branches attached to the last checkpoint has exceeded the maximal limit, or a low-confidence branch is met.

C. Checkpoint Management

All RAT checkpoints are allocated and reclaimed in a FIFO order. It is clear that a new checkpoint will be allocated only if there are free available checkpoints. On the other hand, the oldest checkpoint is reclaimed when either all its associated instructions have been allocated and have

completed execution successfully, or any of its associated branches is mispredicted. In the former case, the processor smoothly gets back to conventional mode and continues to execute those instructions attached to the next checkpoint. In the latter case, the processor discards the execution after the checkpoint, and uses the checkpoint to restore the RAT mappings.

Figure 3 gives an example of this selective checkpoint policy. In the example, up to four checkpoints are supported, and a maximum of four branches can be assigned to a checkpoint. The dots on the timeline represent unresolved branches in current instruction window. On the first such branch, regardless of its confidence, a global checkpoint (GC1) is always created, which initiates checkpoint mode and sets the start point for potential misprediction recovery. The second branch is naturally provided with a second checkpoint (GC2), because it is regarded as low-confidence. For the subsequent high-confidence branches, they are still attached to this second checkpoint, until the maximal limit of four branches per checkpoint is met. That is, on the fifth branch from the last allocated checkpoint, a third checkpoint (GC3) is created regardless of prediction confidence. Later, on the next low-confidence branch, a fourth and the last checkpoint (GC4) is allocated. Because all free available checkpoints have been exhausted, and in order not to stall execution in checkpoint mode at this point, subsequent branches (both branch 9 and 10) are assigned to this last checkpoint regardless of their confidence. When branch 1 is predicted correctly and retired successfully, GC1 is reclaimed at once. Then it is allocated again on subsequent branch 11 which is regarded as low-confidence.

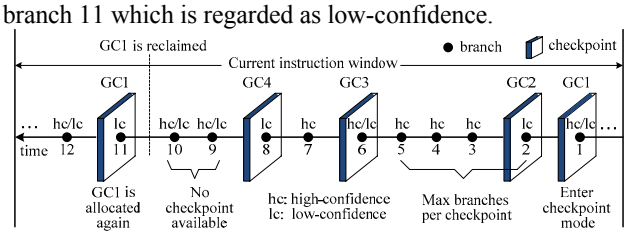

Figure 3. Example of checkpoint allocation and reclaimation.

IV. EXPERIMENT AND ANALYSIS

A. Simulation Environment

A performance evaluation has been carried out on the SimpleScalar toolset [9], which is widely used to simulate cycle-accurate, out-of-order microarchitectural processors. Up to now, lots of modern processors have been successfully tested in many aspects by modifying parts of original source codes in SimpleScalar. In our experiment, a baseline out-of-order 4-way superscalar processor has been modeled with the architectural parameters, as summarized in TABLE I. Experimental results will be obtained from the execution of the entire SPEC2000 benchmark suite including both integer and floating-point benchmarks.

B. Results and Analysis

We evaluate the RAT recovery schemes under five different conditions, as shown in Figure 4. The *BASE* represents the baseline parameters of TABLE I, containing a 128-entry instruction window and using retirement map table restoration method. For remaining four schemes, each of them implements a 2048-entry large instruction window, and the only difference is their recovery mechanisms. *RMAP* and *HBMAP* respectively indicate retirement map table and history buffer method from Section I. *SEL-CKPT* is our selective checkpoint proposal, which uses 8 checkpoints, limits the maximum of 256 branches per checkpoints, and employs a branch confidence estimator for deciding when to create a new checkpoint. *IDEAL* means an ideal model without any recovery overhead. All performance numbers in this experiment are reported as normalized instruction per cycle (IPC) relative to the performance in BASE condition.

TABLE I. BASELINE PROCESSOR PARAMETERS

Parameter	Configuration
Issue Policy	Out of order
Machine Width	Fetch, issue, commit: 4 instructions/cycle
Instruction Window	128-entry size
Branch Predictor	Hybrid 2K gshare, 10-bit global history, 2K bimodal, 1K selector; 2048-entry, 4-way BTB
Branch Penalty	5 cycles
Recovery Scheme	Using retirement map table
Register File	96 Integer/ 96 FP
Functional Units	4 INT ALU, 1 INT Mult/Div, 2 FP ALU, 1 FP Mult/Div
L1 I/D-Cache	32KB, 4 way, 64 Byte line, 2 cycles hit time
L2 Unified Cache	512KB, 8 way, 128 Byte line, 10 cycles hit time
Memory Latency	100 cycles

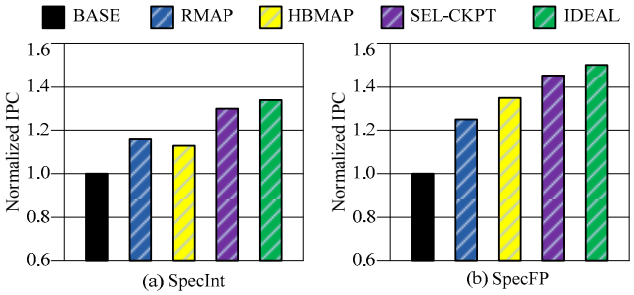

Figure 4. The performance of misprediction recovery mechanisms.

As we can see from the graph, the processor performance has been improved significantly when instruction window gets large enough. In the two sequential RAT restoration schemes, *HBMAP* performs better than *RMAP*. Nevertheless, for the SPECINT benchmark with frequent branch mispredictions, *HBMAP* still suffers an 11% performance reduction relative to *IDEAL*. As an instant recovery mechanism, our proposal *SEL-CKPT* performs best of all, only reducing performance within 3% of the ideal model.

To better understand the behavior of our selective checkpoint policy, TABLE II presents some related statistics.

On average, there are about 796 instructions executed successfully between two branch mispredictions. The average percentage of mispredicted branches is 81%, which are predicted to be low-confidence and checkpointed. For per misprediction, the average number of valid instructions re-executed because of rolling back to a prior checkpoint is 15, just 1.9% of the misprediction distance. In a word, we achieve a high checkpoint coverage and a very low checkpoint overhead by using the selective checkpoint policy.

TABLE II. STATISTICAL RESULT OF SELECTIVE CHECKPOINT POLICY

	SpecInt	***SpecFP***	***Average***
Misprediction Distance	325	1267	796
Checkpoint Coverage	84%	78%	81%
Checkpoint Overhead	12(3.7%)	18(1.4%)	15(1.9%)

V. RELATED WORK

Some other research works have addressed the reduction of the recovery penalty. [10] proposes an out-of-order release checkpoint mechanism which reduces the number of RAM checkpoint to about one third. In [11], a ROB-like structure is proposed to accelerate checkpoint recovery. Such structure allows the recovery from selected branches. Similarly, a selective checkpoint mechanism to recover mispredictions and support large instruction window is proposed in [12, 13]. [14] gives a key idea that brings stall when speculation is likely to deteriorate performance, thus minimizing recovery cost while conserving good speculation opportunities.

With respect to the energy and latency of the RAT, [15] compares the RAM and CAM approaches. It is concluded that when the number of checkpoint exceeds a limit, CAM approaches become more efficient and faster. To achieve high efficiency, a hybrid RAM-CAM register renaming scheme is presented in [16], which combines the best of both approaches.

In addition, checkpoints are used on low-confidence loads to restore the processor status if a load value is mispredicted, and [17] proposes checkpointed early load retirement for reducing the execution stall of dependent instructions.

VI. CONCLUSIONS

Register Alias Table (RAT) is the critical component to perform register renaming in out-of-order microprocessors. However, a branch misprediction may stall the rename process for lots of cycles for the RAT recovery.Therefore, a fast RAT recovery mechanism is necessary to sustain high performance. In this paper, we present a selective checkpoint policy which creates a new checkpoint just on the branch with low confidence to reduce the number of checkpoints. Experimental results show that as instruction window size increases, our proposal performs best of all, achieving both a fast recovery and a very low checkpoint overhead.

ACKNOWLEDGMENT

This work has been supported by National High Technology Research and Development Program of China under Grant No. 2011AA120204. We would like to thank Xunying Zhang, Ruxia Pei and Qiang Wang for their comments and suggestions on this work.

REFERENCES

[1] Wenbing Jin, Jing Dong, Keqin Lu, and Yanghua Li, "The Study of Hierarchical Branch Prediction Architecture," Proc. IEEE 14th International Conference on Computational Science and Engineering (CSE 11), IEEE Press, Aug. 2011, pp. 16-20.

[2] Mingyan Yu, Xiangjian Zhang, and Bing Yang, "Low Power Branch Target Buffer Design based on Hopping Access," Journal of Computer-Aided Design & Computer Graphics, vol. 22, Apr. 2010, pp. 695–702.

[3] M. Shakeri. A. T. Haghighat, and M. K. Akbari, "Modeling and Evaluating the Scalability of Instruction Fetching in Superscalar Processors," Proc. Fourth International Conference on Information Technology (ITNG 07), IEEE Press, Apr. 2007, pp. 972-972.

[4] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal, 2001.

[5] D. Leibholz and R. Razdan, "The Alpha 21264: A 500 MHz Out-of-order Execution Microprocessor," Proc. the 42nd IEEE Computer Society International Conference (COMPCON 97), IEEE Press, Feb. 1997, pp. 28-36.

[6] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," IEEE Micro, vol. 16, Apr. 1996, pp. 28-41.

[7] T. N. Buti, R. G. McDonald, Z. Khwaja, A. Ambekar, H. Q. Le, W. E. Burky, and B. Williams, "Organization and Implementation of the Register-renaming Mapper for Out-of-order IBM POWER4 Processors," IBM Journal of Research and Development, vol. 49, Jan. 2005, pp. 167–188.

[8] E. Safi, A. Moshovs, and A. Veneris, "On the Latency and Energy of Checkpointed Superscalar Register Alias Tables," IEEE Transaction on Very Large Scale Integration (VLSI) Systems, vol. 18, Mar. 2010, pp. 365-377.

[9] http://www.simplescalar.com/

[10] A. Moshovos, "Checkpointing Alternatives for High-Performance, Power-Aware Processors," Proc. the International Symposium on Low Power Electronics and Design (ISLPED 03), IEEE press, Aug. 2003, pp. 318-321.

[11] P. Akl and A. Moshovos, "Turbo-ROB: A Low Cost Checkpoint/Restore Accelerator," Lecture Notes in Computer Science, vol.4917, Jan. 2008, pp. 258–272.

[12] H. Akkary, R. Rajwar, and S. T. Srinivasan, "An Analysis of a Resource Efficient Checkpoint Architecture," ACM Transactions on Architecture and Code Optimization, vol. 1, Dec. 2004, pp. 418–444.

[13] H. Akkary, R. Rajwar, and S. T. Srinivasan, "Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors," Proc. the 36th International Symposium on Microarchitecture (MICRO 36), IEEE press, Dec. 2003, pp. 423-434.

[14] P. Akl and A. Moshovos, "BranchTap: Improving Performance with Very Few Checkpoints through Adaptive Speculation Control," Proc. International Conference on Supercomputing (ICS 06), ACM press, Jun.2006, pp. 36-45.

[15] E. Safi, A. Moshovos, and A. Veneris, "A Physical Level Study and Optimization of CAM-based Checkpointed Register Alias Table," Proc. the 13th International Symposium on Low Power Electronics and Design (ISLPED 08), IEEE press, Aug. 2008, pp.233-236.

[16] S. Petit, R. Ubal, J. Sahuquillo, and P. Lopez, "A Power-aware Hybrid RAM-CAM Renaming Mechanism for Fast Recovery," Proc. IEEE International Conference on Computer Design (ICCD 09), IEEE press, Oct. 2009, pp. 150-157.

[17] N. Kirman, M. Kirman, M. Chaudhuri, and J. F. Martinez, "Checkpointed Early Load Retirement," Proc. the 11th International Symposium on High-Performance Computer Architecture (HPCA 05), IEEE press, Feb. 2005, pp. 16-27.