A Minimal Roll-Back Based Recovery Scheme for Fault Toleration in Pipeline Processors

Jun YAO, Ryoji Watanabe, Takashi Nakada,
Hajime Shimada, Yasuhiko Nakashima
Graduate School of Information Science,
Nara Institute of Science & Technology,
Takayama-Cho 8916-5, Ikoma 630-0192, Japan
{*yaojun, ryoji-w, nakada, shimada, nakashim*}*@is.naist.jp*

Kazutoshi Kobayashi
Graduate School of Science & Technology,
Kyoto Institute of Technology,
Matsugasaki, Sakyo-ku, Kyoto 606-8585, Japan
kobayasi@kit.ac.jp

***Abstract*—In this paper, we proposed a light-weighted recovery scheme for fault tolerable pipeline processors after error has been detected by redundant executions. A minimal rolling back procedure is designed to schedule the re-execution based recovery in a one-cycle delay. This scheme makes full use of in-fly pipeline working status to aid the recovery, which relieves the recovery from a large checkpoint buffer.**

***Keywords*-Fault tolerance; System recovery;**

I. Introduction

Generally, the error detection in electronic devices can be achieved by proper replication. Dual modular redundancy (DMR) based execution in IBM series including z990 [1] and replicated thread execution AR-SMT [2] are typical ways of error checking mechanisms in microprocessors. However, conventional recovery methods in these architectures usually require a large checkpoint buffer to cache correct processor status. After error detection, long distance rolling back is performed to restore system from erroneous executions. Both the additional checkpoint buffer and the required state machine to control the recovery will add to the area cost and the large extra units may cause further vulnerability issues for the system robustness.

In this paper, we proposed a light-weighted roll-back based error recovery scheme. The proposed method uses the correct in-fly status inside the pipeline processor, which is expected to largely reduce the recovery complexity and cost.

II. Proposal of a Light-Weighted Recovery

A recovery procedure is necessary to restore processor status after detecting error in runtime executions. Figure 1 gives our baseline pipeline processor with error detection supports, following a typical dual modular redundancy (DMR) structure. The DMR processor contains two identical pipelines and each pipeline follows a textbook prototype that has six stages, including IF, ID, RR, EX, MA, and WB. The instructions are replicated at the IF stage by feeding the two pipelines with identical program counters (PCs). Data integrity and validity checks are performed per each stage by the "==?" units.

A. Basic Roll-back Based Recovery Procedure

To reduce the recovery cost from checkpoint buffers and control logics, we proposed a fast recovery scheme based on a minimized roll-back concept. The basic idea of recovery is given in Fig. 2.

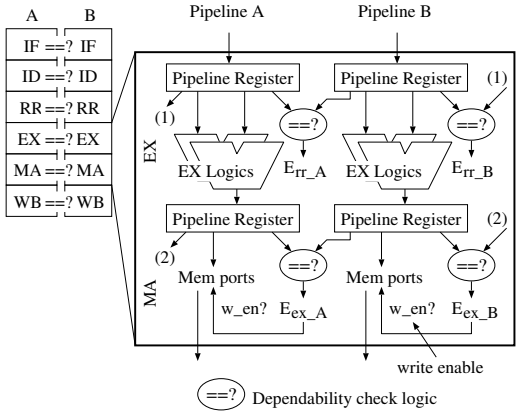

Fig. 1. DMR execution scheme in EX and MA stages

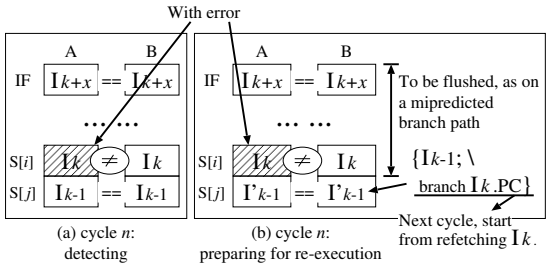

Fig. 2. Restarting execution by inserting a branch style instruction.

The re-execution based recovery is triggered by unconditionally jumping to the correct restart point via citing the information stored in the last correct stage. Fig. 2(a) shows an example of two consecutive instructions as I_{k-1} and I_k. Assuming that at cycle n, the comparators indicate that one of the executions of I_k is problematic while the executions of I_{k-1} are verified to be correct, the information of I_{k-1} can thus be used to instruct the recovery.

As depicted in Fig. 2(b), I_{k-1} will be extended a little to compound with a dummy branch instruction, as $branch\ I_k.PC$. I_k's program counter (PC) will be filled into

978-0-7695-4289-8/10 $26.00 © 2010 IEEE
DOI 10.1109/PRDC.2010.44

```
/* function of get_restart_PC() */
enum {IF, ID, RR, EX, MA, WB} i, j, r;

/* 1) Locate the last stage with error */
for (i=WB;i≥IF;i--)
    if (pipeA_stage[i].error || pipeB_stage[i].error)
        break;
if (i<IF) return NULL; /* No Error */

/* 2) Locate the next stage, skipping NOPs from hazard. */
for (j=i+1;j≤WB+1;j++)
    /* HAZARD_NOPs are not in original programs. */
    if (pipeA_stage[j].OP != HAZARD_NOP) break;
/* I_good in stage[j], I_error in stage[i] */

/* 3) Flushing erroneous info, like a branch mispred. */
for (r=IF;r<j;r++)
    pipeA_stage[r].OP = pipeB_stage[r].OP = HAZARD_NOP;

/* 4) Jump to the correct restarting point
 * by using the correct value in stage[j] */
if (pipeA_stage[j].flag & IS_BRANCH)
    if (pipeA_stage[j].flag & IS_TAKEN)
        /* (I) restart from branch target */
        return pipeA_stage[j].brTarget;
/* (II) restart from next PC of I_good */
return ++pipeA_stage[j].PC;
```

Fig. 3. Basic roll-back based recovery

the branch instruction as the branch target. If the jump is correctly handled, I_k will be re-fetched from the cache and the re-execution will thereby start in cycle n+1. Here, we assume that caches are covered by ECC-like technologies so that a re-fetch can obtain the correct data.

The algorithm to extract restart PC from the last correctly executed instruction I_{k-1} is listed in Fig. 3. Specifically, as the step "4)" in Fig. 3 shows, according to the instruction type of last correct instruction I_{good}, the restart PC can either be the next PC of I_{good}, or I_{good}'s branch target. Since the branch target of I_{good} is also under the comparison in the DMR processor, it is safe to use this verified value under this condition.

B. Necessary Extensions for Delay Branch Instruction

Some instruction set architectures (ISAs) allow branches to have one or more delay slot instructions to reduce the branch misprediction penalty. Since the PC of a delay-slot instruction can not be directly used to calculate its successive instruction, a special treatment is required, as shown in Fig. 4.

The basic idea is that when the last correctly executed I_{good} is in the delay slot of a delay branch, a further lookup in the pipeline will be performed to locate the delay branch itself, as I_{good2}. Combining the information of both I_{good} and I_{good2}, the correct restarting address of I_{error} can be calculated, shown as step "3.5)" in Fig. 4.

Another special case is that the I_{error} itself is in the delay slot. Two cycles may be required to restart from this situation. The two cycles respectively restart the delay-slot instruction and the possible branch target of I_{good}. This two-cycle recovery will complicate the recovering controller. Assuming that the branch instruction only affects the PC in the next IF stage, no data will be actually updated into the register file or

```
/* between (3) and (4), in function of get_restart_PC() */
enum {IF, ID, RR, EX, MA, WB} m;
/* 3.5) Locate a further stage containing non-HAZARD_NOP. */
for (m=j+1;m≤WB+1;m++)
    if (pipeA_stage[m].OP != HAZARD_NOP) break;
/* Continuous I_good and I_good2 in stages [j], [m],
  I_error in stage[i] */

if (pipeA_stage[j].flag & IS_DELAYSLOT)
    /* stage[j] inst is delay slot,
     cannot be directly used for determine restart_PC */
    if (pipeA_stage[m].flag & IS_TAKEN)
        restart_PC = pipeA_stage[m].brTarget;
    else
        return ++pipeA_stage[j].PC;
else if (pipeA_stage[j].flag & IS_BRANCH)
    if (pipeA_stage[j].flag & IS_DELAYBRANCH)
        /* Error in delay slot, restart delay branch */
        return pipeA_stage[j].PC;
```

Fig. 4. Extensions to guarantee correct roll-back on path after branch with a delay-slot

memory[1]. Under this consideration, it is safe to restart from the delay branch instruction. By this means, the recovery can still be scheduled within one cycle.

C. Checkpoint Buffer

An ECC-covered checkpoint buffer will be added after WB stage to ensure that there is always a correctly executed instruction inside the pipeline processor. This buffer contains PC, instruction type, and branch target. The NOP instructions generated by pipeline hazards will not update this buffer. In addition, if a delay slot instruction passes through the commit phase, it will be merged with the delay branch in this buffer to indicate the correct restarting point.

III. Conclusion

We presented a recovery scheme based on a minimal distance roll-back for fault tolerable pipeline processors with proper redundancy. After error detecting, the recovery can be scheduled within one cycle. By making use of the already redundant units for error detection, this method largely alleviates the necessities for checkpoint buffers. Compared to long-distance recovery which requires additional buffers to cache register file data and memory modification logs, the proposed method only needs a buffer to store a PC, an instruction type, and a branch target after the commit phase. This cost is relatively negligible.

Acknowledgment

This work is supported by VLSI Design and Education Center (VDEC), University of Tokyo with the collaboration of Synopsys Corporation. This work is supported by JST CREST.

References

[1] P. Meaney, S. Swaney, P. Sanda, and L. Spainhower, "IBM z990 Soft Error Detection and Recovery," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, pp. 419–427, Sept. 2005.

[2] E. Rotenberg, "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors," in *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing*, pp. 84–91, 1999.

[1]If the branch instruction attempts to modified register file, it can be decomposed into instructions of a register update and a pure branch.