IEICE TRANS. INF. & SYST., VOL.E95–D, NO.1 JANUARY 2012

PAPER Special Section on Trust, Security and Privacy in Computing and Communication Systems

A Fault-Tolerant Architecture with Error Correcting Code for the Instruction-Level Temporal Redundancy

Chao YAN[†a)], **Hongjun DAI**[†b)], ***Nonmembers***, ***and*** **Tianzhou CHEN**[††c)], ***Member***

SUMMARY Soft error has become an increasingly significant concern in modern micro-processor design, it is reported that the instruction-level temporal redundancy in out-of-order cores suffers an performance degradation up to 45%. In this work, we propose a fault tolerant architecture with fast error correcting codes (such as the two-dimensional code) based on double execution. Experimental results show that our scheme can gain back IPC loss between 9.1% and 10.2%, with an average around 9.2% compared with the conventional double execution architecture.

key words: *soft errors, fault tolerance, double execution, instruction reuse buffer, fast error correcting code*

1. Introduction

Soft error (also called transient fault) is a kind of temporary faults occurred in digital circuits. It is caused by internal noise such as power transients, or external source such as cosmic particle hits. The processor scaling characteristics, such as higher transistor integration densities, increasing clock frequencies, lower operating voltages and the shrinking of the feature size, make soft error tolerance an increasingly significant issue for future microprocessor design [1]. According to the reports from Intel Corporation, the mechanisms to enable fault tolerance will take up to 5%-10% of the total transistors over the next several years [2].

Most fault tolerant techniques are based two major types of redundancy: temporal redundancy (TR) and spatial redundancy (SR). TR means to execute each operation multiple times separately on the same hardware, performing the same calculation more than once to validate the results. It suffers the performance penalty imposed by the redundant workloads. SR means to use the replicated hardware to perform the same workload, but the additional hardware overhead is required.

In the out-of-order processors, the reliability is protected with the instruction-level TR. The most representative implementation of TR is Double Instruction Execution (DIE) [3], which entails to execute each instruction twice on the same hardware and the comparison among the results of the dual execution is taken before the instruction commit. The strategy can cover most soft errors. However, the performance penalty that imposed by the redundant workload is significant. It is reported that the performance penalty is up to 45% compared with the single instruction execution (SIE), the execution without redundancy [3].

Therefore, it needs additional mechanisms to improve the performance of DIE in order to satisfy the requirement of modern microprocessors, such as the instruction reuse [4] and the instruction pre-computation [5]. In [4], each executed instruction stores its input operands and the results into an instruction memorization table, which is called Instruction Reuse Buffer (IRB). Once the execution with the same input operands comes again, it can bypass the execute phase by reusing the results from IRB. In [5], prior to execution, the profiling data that includes op-codes with input operands and results as well as the application binary code, is loaded into the pre-computation table for reuse.

According to the statistic from the reports, DIE-IRB architecture has an improved performance compared with DIE. It can gain back nearly 50% of the IPC loss that occurred due to ALU bandwidth limitations for instruction-level temporal redundancy and 23% of the overall IPC loss [4]. However, there is a problem in the DIE-IRB architecture. According to [6], the soft errors caused by particle strikes will become increasingly serious for microprocessor design in future. The raw error rate per device in a bulk CMOS process is projected to remain roughly constant or decrease slightly for the next several technology generations [7]. Thus,unless we add more extensive error protection mechanisms, a processors error rate will grow in direct proportion to the number of devices we add to a processor in each succeeding generation. Since the soft error rate goes up, the error recovery will have significant impact on the performance of the fault tolerant system because the recovery process is taken more frequently. Thus, besides the DIE, the performance of error recovery also need to be enhanced.

The DIE-IRB architecture has an error coverage process with significant performance degradation. Once the soft error is detected, the coverage process is stimulated to correct the error. It firstly clears the error affected instruction and all its subsequences from ROB and evacuates the Instruction Fetch Queue, where instructions are waiting for decoding. The pipeline has to reload the instructions from the Instruction Cache and re-execute them. Such a process can greatly prevent the instruction pipeline.

Error Correcting Code (ECC) is a possible solution, which has been widely used to protect memory. It is a sys-

Manuscript received March 15, 2011.
Manuscript revised July 2, 2011.

[†]The authors are with the Department of Computer Science and Technology, Shandong University, Jinan, China.
[††]The author is with the Department of Computer Science and Technology, Zhejiang University, Hangzhou, China.
a) E-mail: yanchao.chn@gmail.com
b) E-mail: dahogn@sdu.edu.cn (Corresponding author)
c) E-mail: tzchen@zju.edu.cn

DOI: 10.1587/transinf.E95.D.38

Copyright © 2012 The Institute of Electronics, Information and Communication Engineers

tem of adding redundant data, or parity data to the message. Furthermore, it can recover the protected data from a number of errors [8]. There are some typical ECC, such as hamming code [9] applied in DRAM controllers, SEC-DED code used in memory systems, the parity check code [10] used in storage systems. The advantage of ECC is that it can correct the faults within its capacity instead of re-executing the fault instructions. Thus, we introduce the ECC into instruction pipeline to reduce the performance degradation imposed by the error recovery process.

In [11], it proposed that the calculation of in-pipeline ECC, such as SEC-DED, requires resources in the timing critical paths. Thus, conventional ECC are not suitable for fault tolerant pipeline. However, there are some special ECC, such as the multi-dimensional code introduced in [12], which have stronger correcting capacity and lower VLSI overhead. In [13], the two-dimensional (2D) error coding techniques that enables fast common-case error-free operation with the combination of light-weight horizontal per-word error coding is introduced.

In this paper, we introduce a kind of efficient ECC, which is called fast ECC (FECC) into instruction pipelines. The in-pipeline storage capacity of IRB is adopted as the foundation of our scheme. we proposed to store the FECC into IRB to correct the soft errors (FECC-IRB). This approach is implemented by the following steps:

First, FECC is calculated by the FECC calculation units (FCU) before the instructions are committed. It is stored into IRB for possible reuses. Second, when instructions come into the pipeline, they perform a reuse test to find the entry in the IRB. Third, before the instruction commit, there is a comparison of the FECC for soft error detection. If any fault happens, FECC can correct the error, rather than re-executing each instruction from the error point.

We conducted the experiments based on an improved sim-outorder simulator from SimpleScalar 3.0 [14] and used programs from Mibench suite [15] for simulation. The results show that the ideal FECC-IRB approach can reduce the IPC loss from 9.1% to 10.2%, with an average about 9.2% compared with SIE. In addition, in programs with high proportion of ALU instructions, FECC-IRB has a 8% IPC improvement, which is close to the ideal FECC-IRB (100% IRB hit). This means that the FECC-IRB performs more excellent when it has serious ALU contention. On the other hand, the IPC loss caused by FECC calculation is controlled within 1.5% compared with the conventional IRB. Thus, the FECC-IRB approach is valuable to enhance the performance of the high reliability computing system.

2. Related Works

This paper introduces the architecture to enhance the performance of the DIE architecture by storing FECC into IRB. There were related works about DIE, IRB and ECC.

In [3], it introduces the time redundant technique, which suggests executing each instruction twice with multiple, pipelined functional units to detect unexpected soft errors in superscalar microprocessors. In [16], it also investigates a duplication-based fault tolerant technique: error-detection is achieved by verifying the redundant results of dynamically replicated threads of executions. These schemes can cover most soft errors in functional units. However, They suffer from a significant performance penalty imposed by the redundant workloads. This means that the auxiliary mechanism is needed to improve its performance.

In [4], it introduces the Instruction Reuse, which uses the instruction memorization technique to avoid the redundant computations of the conventional DIE system through reusing the results of the previously executed instructions. Work [5] combines instruction pre-computation with IRB to enhance the performance of the DIE system, which reduces the performance penalty of work [4] in a further step with more hardware supports. Since the IRB can store information for the executed instruction, there is still much potential in IRB that worth to have a further exploration.

ECC has been widely used on memory protection. In [9], it proposed a concatenated Reed-Solomon code combined with hamming code for dynamic random access memory controller. In [17], it used the BCH based DEC-TED code to protect L2-chache from soft errors. In [10], a high performance 2D ECC encoding technique with low VLSI overhead and strong correcting capability is introduced to protect the memory system from soft errors, it investigates the design of the high performance multidimensional ECC for dynamic memory systems. In general, ECC is effective to correct undesired soft errors, while few works introduce it into the out-of-order pipeline.

Above all, DIE is the basic method used as a guarantee of the instructions reliability. IRB strategy is the key point to improve the system performance of the conventional DIE system. It needs a new way to enhance the system performance of the traditional DIE system and exploit the potential capability of IRB. Different from the traditional ECC, FECC, such as Two-dimensional (2D) is a novel solution for the performance problem of the DIE system.

3. FECC-IRB Architecture

3.1 System Architecture

FECC-IRB architecture with IRB and the specialized FCU is shown in Fig. 1. IRB is used to store the previous executed instructions for reuse, which can enhance the performance of DIE. FCU is used to calculate the FECC for the instruction. Reorder Buffer (ROB) is used to store the results of the instruction pairs to enforce the in-order commit. Dispatch is to decode and allocate ROB unit for the instruction. Issue window and FU are used to execute instructions stored in ROB. Check& Retire is to compare the result of the instruction pair to ensure the reliability and commit the reliable instruction.

DIE is implemented by duplicating each instruction at the decode/dispatch stage. Thus, each instruction has two copies: the first copy is corresponding to the original in-

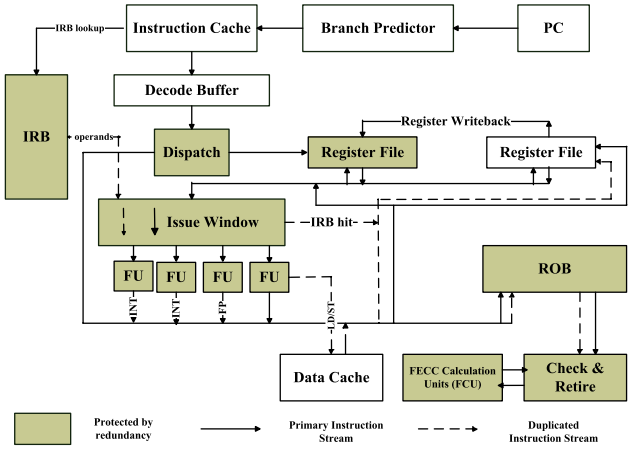

Fig. 1 The fault-tolerant architecture with FECC-IRB. FCU is used to assist the Check/Retire in order to ensure the instruction reliability.

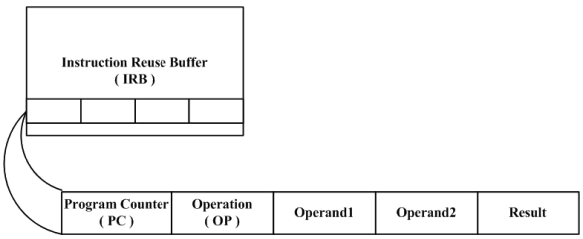

Fig. 2 The content of the IRB item, including op-code, PC, two operands and FECC.

struction (O-Ins), while the second copy is corresponding to the duplicated instruction (D-Ins). Both O-Ins and D-Ins are allocated items in ROB.

IRB is a small hardware table storing the previous executed instructions. After the same instruction comes again with the same input operands, it can directly skip the FU by reusing the FECC stored in IRB. The instruction that without a reuse item is executed in FUs.

When the instruction PC is sent to the instruction cache, another copy is sent to IRB to perform a lookup. If there is a PC hit, further reuse test is motivated before the issue window. If there is no hits, the IRB can not produce any change to the DIE pipeline. Otherwise, the D-Ins can bypass the execution and reach the Check& Retire stage.

3.2 FECC-IRB Details

The committed instructions update IRB with its op code, PC, input operands and the FECC code. The item of IRB is shown in Fig. 2. The op code and the operands are used to perform the reuse test. PC is used to perform the IRB lookup concurrently with the instruction fetch. FECC is used to check and ensure the instruction reliability.

As shown in Table 1, there are different strategies for different kinds of instructions. All ALU instructions including integer operations and float point operations are stored in IRB for reuse. For branch target instructions, the IRB is only used for the target address calculation. For load/store

Table 1 Different strategies for different kinds of instructions.

Instruction	Method
ALU instruction	Store the ALU instruction itself
Brach Target Instruction	Store the branch the branch target adress calculation operation
Load/Store instruction	Store the adress calculation operation

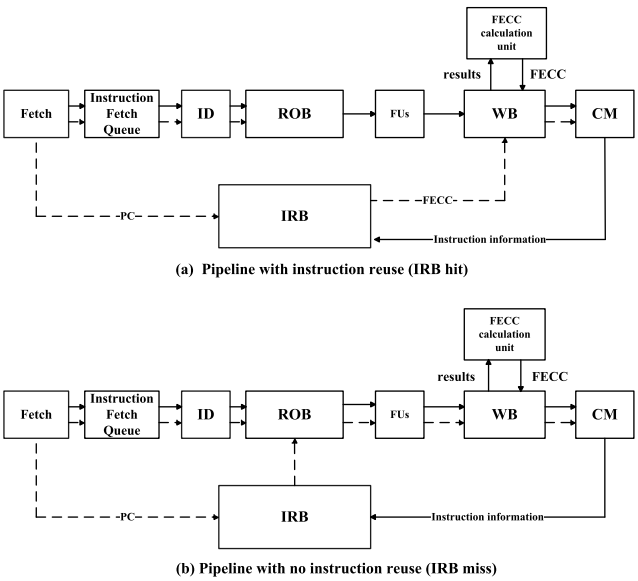

Fig. 3 Instruction execution pipeline under different circumstances. The solid lines show the flow of the original instructions in the primary stream and the dashed lines show the flow of duplicated instructions, which have possibility to skip the issue window (reuse test success).

instructions, the IRB is used for the address calculation.

As shown in Fig. 3, the FECC-IRB pipeline is an improved five stage pipeline, including fetch stage, dispatch stage, issue stage, Writeback stage and commit stage. Each instruction is duplicated in ROB, respectively corresponding to the O-Ins and D-Ins. O-Ins is always executed in FUs. After entering the pipeline (fetch stage), the D-Ins performs a lookup with its PC to find a relevant entry in IRB. If there is a PC hit, further reuse test is stimulated to test the operands. According to whether there is a IRB hit, there are two different situations: (a) IRB hit. D-Ins skips the FU by reusing the FECC of the former execution from IRB at the Writeback stage. (b) IRB miss (PC miss or reuse test failed). D-Ins has to be executed in ALU.

To ensure the instruction reliability, a FCU is used to calculate the FECC for the result of the O-Ins at the Writeback stage, which is used to update the IRB for future reuse when instructions are committed.

FECC-IRB can decrease the certain redundant executions imposed by DIE. In addition, the FECC stored in the IRB that is a potential factor to correct the results of the error instructions within its correcting capacity if soft errors are detected at the Writeback stage. Obviously, if the instruction is damaged so serious that the error is beyond FECC's correcting capacities, both O-Ins and D-Ins have to

be re-executed.

3.3 Pipeline Process

The FECC-IRB pipeline is shown in Fig. 4.

O-Ins is always executed in ALU. At the Writeback stage, FCU calculates the FECC for the O-Ins. However, it has to wait for commitment until the instruction pair is checked with each other.

D-Ins has a different execution process. In the Fetch stage, it performs a lookup of IRB with its PC to find whether there is a possible reuse entry. If there is no entry (PC miss), the instruction has to be executed in ALU. Otherwise (PC hit), further reuse test is performed before the issue stage. If the execution has the same input operands with the one stored in the IRB, it can reuse the FECC from the corresponding item and bypass the execution. If the reuse test is failed, the D-Ins must be executed in ALU.

When both O-Ins and the D-Ins reach the Writeback stage, a comparison is performed according to whether there is a IRB hit:

(a)Hit in IRB

The comparison is performed between the reused FECC and the FECC calculated for the O-Ins. If they are equal, the results of the O-Ins are reliable enough to be committed. Otherwise, if they are different, the reused FECC from the IRB can be used to correct the results of the O-Ins within its ability rather than to stimulate the error recover process, which has a considerable performance penalty. If the errors are so serious that the results cant be repaired with the FECC, the instruction pair has to be re-executed.

(b)Miss in IRB

D-Ins has to be executed without skipping the execution. The comparison is performed between results of the O-Ins and the D-Ins at the Writeback stage. If their results are equal, the O-Ins is committed. Otherwise, because there is no reused FECC can be utilized to correct the results, a conventional recover penalty is inevitable.

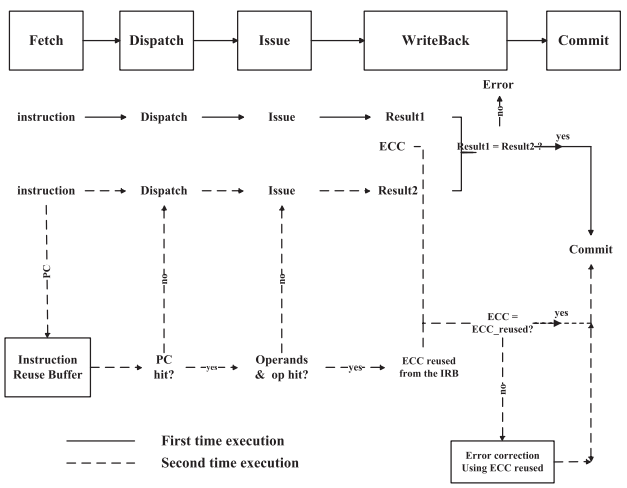

Fig. 4 FECC based IRB double execution process.

3.4 Correctness of the FECC

IRB does not need any extra protection even if errors could happened on the FECC reused from it. According to the pipeline process, the P-Ins is always executed in the FU. The FECC for its result value is always calculated by the FCU. For the D-Ins, if there is no IRB hit, the D-Ins has to be treated the same way as the P-Ins. Thus, any error occurred during the FECC calculation can be detected by the comparison of FECC. If the is a IRB hit, the D-Ins directly reuse the FECC from IRB. Errors occurred both during the FECC calculation process of the P-Ins and on the IRB can be detected by the comparison of FECC. The difference is that the invalid FECC can not be used to correct the error. Thus, all instructions from the error point have to be re-executed.

4. FECC Design

4.1 Limitation of the Conventional ECC

According to [12], it is a performance bottleneck to compute the in-pipeline ECC bits for error detection. To avoid this drawback, some microprocessors periodically sweep through the array, checking the data integrity. This technique (called scrubbing) has lower error coverage than checking ECC [18]. The characteristics of typical ECC are listed in Table 2.

It can be found that in the conventional ECC techniques, there are limited correcting performance. For example, if a SECDED is used to protect multi-bit errors, each word must store extra bits for the SECDED code-word, and the instructions have to suffer the delay, calculating, and comparing the SECDED code-words. From Table 2, SECDED has only 1-bit error correction and 2 bit detection capability with a hamming distance 4. Thus, it is unsuitable to introduce conventional ECC techniques into pipeline to protect multi-bit failures because of their low effieciency.

2D error coding techniques [13], which enables fast common-case error-free operation while still maintaining high error coverage with low VLSI overheads, is ideal to be used in pipeline. The key point of the 2D error coding is the combination of lightweight horizontal per-word error coding with vertical column-wise error coding. The types of codes used in the horizontal and vertical directions allow us to trade-off the error coverage against the VLSI overheads

Table 2 Characteristics of the conventional ECC.

ECC	Characters
SECDED	1-bit error correction, 2-bit error detection (Hamming distance = 4)
DECTED	2-bit error correction, 3-bit error detection (Hamming distance = 6)
QECPED	4-bit error correction, 5-bit error detection (Hamming distance = 10)
OECNED	8-bit error correction, 9-bit error detection (Hamming distance = 18)

to tailor the design to the types and frequency of errors expected. The vertical codes enable correction of multi-bit errors along rows up to and including entire row failures.

4.2 Two Dimensional ECC

A set of information bits is used to form a $q \times l$ square array. A parity check bit is added to each row which is the parity sum of the information bits in that row. Then a parity check bit is added to each column that is the parity sum of the bits in that column. The array is then $(q+1) \times (l+1)$, with $(q+l+1)$ parities added to the original information bits. The resulting code has minimum distance four since changing a single information bit requires changing three of the parity bits to maintain all the parity equations.

To correct a single error anywhere in the array, the parities of all of the rows are computed, as are the parities of all of the columns. A parity failure in exactly one row and exactly one column indicates an error in the bit in that row and column; more than one row parity failure or more than one column parity failure indicates the presence of multiple errors. This permits any double error to be detected.

An error detection and correction example is given in Fig. 5. The area colored with white is used to store data, while the area colored with gray is used to store 2D ECC. (a) shows the normal situation without error. When there is one bit error happened in data area, as shown in (b), the parity failure in exactly one row and exactly one column can locate the exact error coordination. Thus, the error can be recovered according to the error location provided by parity failures in rows and columns as (c).

Let the information bits:

$b(i, j), i = 0 \ldots q-1, j = 0 \ldots l-1$

The row parities are the bits $b(i, l), i = 0 \ldots q$, and the column parities are the bits $b(q, j), j = 0 \ldots l$. $b(q, 1)$ is the parity sum of the information bits, whether computed as a sum of row parities or a sum of column parities.

Suppose that the values of the instruction result form a codeword in the product code in which at most a single error has occurred. To read bit $b(i, j)$ three values are needed:

$$\sum_{t=0}^{q} b(t, j) + b(i, j) \tag{1}$$

$$\sum_{t=0}^{l} b(i, t) + b(i, j) \tag{2}$$

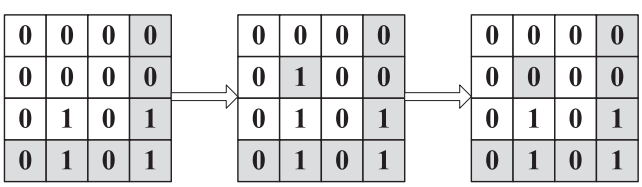

Fig. 5 If single bit error happened, a parity failure in exactly one row and exactly one column indicates an error in the bit in that row and column. Then the error bit is corrected as (c).

For $b(i, j)$ itself, all sums are mod 2. The first is the sum of all bits in the column except the bit itself; the second is the sum of all bits in the row except the bit itself. By virtue of the parity check condition, if no error has occurred, all three values will be the same. The three sets of bits are disjoint, and a single error can affect at most one of the three. Consequently, if the value output is the majority vote of these three, it is correct. However, a double error clearly causes the emitted value to be in error. The presence of a double or multiple errors would have to be detected by a systematic computation of the parity sets that looks for two or more row or two or more column parity violations.

5. Experiments and Results

5.1 Simulation Platform

Our experiments were carried out using a modified sim-outorder simulator from SimpleScalar 3.0 tool suite [14]. Besides the normal execution, we also extended it for DIE, DIE-IRB, and the FECC-IRB scheme. The machine configurations and simulation parameters are given in Table 3. According to the characters of the 32-bit SimpleScalar instruction sets, each IRB item holds one 32-bit PC, one 8-bit op-code, two 32-bit input operands, one 32-bit result. So the size of the IRB item is 17 bytes, and the number of IRB items is 1024, which means the IRB size is 17 KB. For workloads, we used 5 benchmarks from the Mibench suite [15] for simulations.

5.2 IPC Analysis

IPC is the most important parameter, which is used to evalu-

Table 3 Processor parameters of the simulator.

Feature	Simulation Parameters
Fetch/Decode/Issue/Commit Width	8
Fetch-Queue Size	8
Register Update Unit (RUU) Size	256
Load Store Queue (LSQ) Size	128
Integer Adder	4
Integer Multiplier	2
FP Adder	2
FP Multiplier	1
L1 Data Cache	8 KB, 1-way, LRU, 1 cycle latency
L1 Instruction Cache	8 KB, 1-way, LRU, 1 cycle latency
L2 Unified Cache	256 KB, 4-way, LRU, 6 cycle latency
Instruction TLB	64 entries, 4-way, LRU, 30 cycle latency
Data TLB	128 entries, 4-way, LRU, 30 cycle latency
Memory Access Latency	18 cycles and 2 cycles (first, rest)

ate the high performance computer system. Compared with the ordinary computer system. High reliability means inevitable IPC loss imposed by redundancy. The IPC loss can be divided into two distinct parts£normal execution delay and error recovery penalty. Only normal execution delay is calculated in this section to find the IPC loss imposed by DIE workload. The analysis including error recovery penalty is taken in the following section.

The performance is compared without error occurrence to observe the execution IPC gap (IPC difference between different methods) easily. As shown in Fig. 6, the percentage of IPC loss of the considered DIE, IRB and the FECC-IRB compared with SIE system.

The first item indicates the IPC loss of DIE compared with SIE. The average IPC gap between SIE and DIE is up to 24.9%, which is a considerable performance loss caused by the redundant workload. The second item presents the IPC loss of DIE-IRB compared with SIE. It has an improvement on performance, with the average 26.4% IPC enhancement over the DIE. The third item presents the IPC loss of FECC-IRB. It is higher than the conventional IRB, because the FECC-IRB scheme has a FECC calculation penalty.

As shown in Fig. 6, the DIE approach has an IPC loss compared with SIE alleviated by IRB scheme. Compared with IRB, the FECC-IRB approach has higher IPC loss because only normal execution delay is calculated to find the IPC loss imposed by DIE workload. However, the effect of our FECC-IRB approach can be reflected in the following analysis including error recovery penalty because it has an improvement over the error recovery loss.

5.3 FECC-IRB Benefits

The comprehensive analysis including normal execution delay and error recovery penalty is taken in this section in order to observe the advantage of our FECC-IRB approach over DIE-IRB in enhancing the system performance.

In Fig. 7, it presents the IPC of different schemes under different soft error rates. The second and the third items give the IPC gap between the IRB and the FECC-IRB scheme. The third and the forth items present the IPC, where it is artificially set the IRB hit rate to be 100%, denoted as ideal IRB and ideal FECC, which means all D-Ins skip the FU and all soft errors are corrected by FECC respectively.

In Fig. 8, it presents the percentage of IPC improvement of different schemes over different soft error rates. It can be found that the ideal FECC-IRB has a recovery of the IPC loss between 9.1% and 10.2%, reducing an average 9.2% of the IPC loss caused by redundant execution of the DIE system. On the other hand, the ideal IRB scheme can only reduce the IPC loss with the average being around 6.9%. As the soft error rate increased, the advantage of the FECC-IRB is more obvious over the conventional IRB, because the FECC is used to reduce the error recovery penalty, which is decided by soft error rates. However, the IPC gap between normal FECC scheme and IRB is not so obvious.

Three factors contribute to the effect of FECC-IRB on the IPC loss over the DIE approach:

(a) Soft error rate

Higher soft error rate can provide more opportunities using FECC to correct the error affected instructions rather than stimulate the conventional timing-critical recovery process. Thus, the advantage of our FECC over the conventional IRB is more obvious.

(b) IRB hit rate

Higher IRB hit rate can provide more error correcting

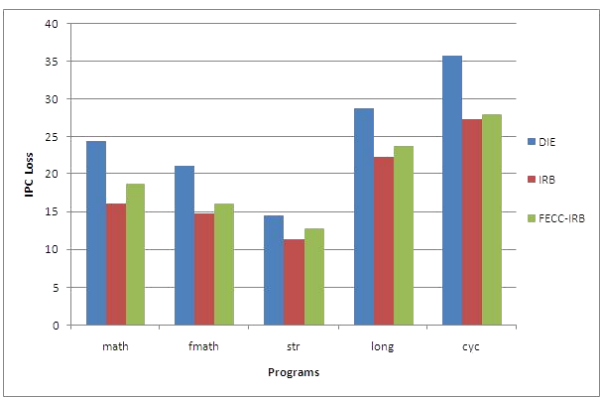

Fig. 6 The IPC loss of DIE, IRB and FECC-IRB.

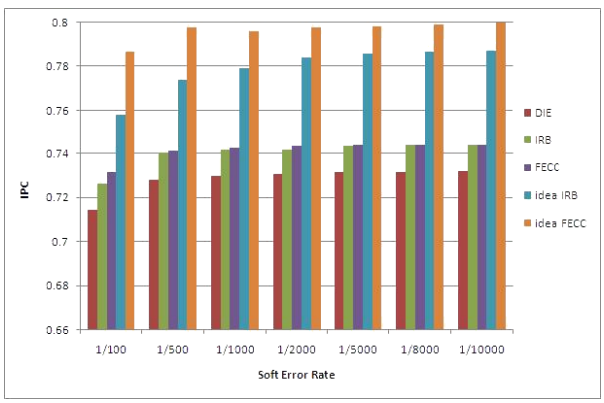

Fig. 7 The IPC of different schemes under different soft error rates.

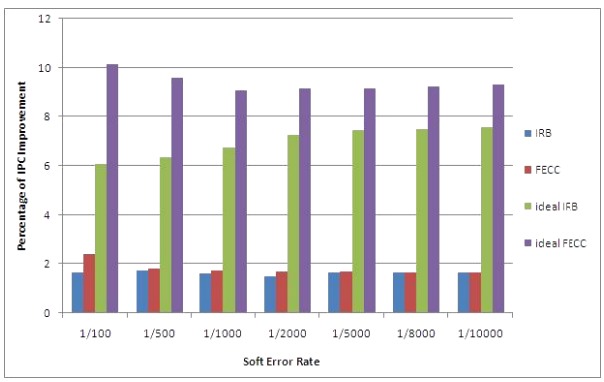

Fig. 8 The percentage of IPC improvement of different schemes over different soft error rates.

Table 4 Types of instruction stored in IRB and the percentage of each kind of instruction in different programs.

Types	math	fmath	long	str	cyc
ALU	28.97%	32.63%	35.83%	54.37%	30.91%
Branch	20.04%	21.15%	21.22%	18.00%	24.82%
Load/Store	59.85%	53.95%	48.69%	36.35%	53.63%

opportunities, too. Because only the FECC reused from the IRB can be used to correct error affected instructions. Thus, increasing the IRB hit rate is a factor to magnify the advantages of our FECC-IRB approach.

(c) IPC of DIE system

It is another factor that can decide the percentage of IPC improvement of the FECC-IRB over the conventional IRB. The IPC loss of conventional DIE can be divided into two distinct parts, the first part is normal execution delay, and the second part is error recovery penalty. The FECC mainly enhances the IPC loss of the error recovery penalty.

5.4 ALU Contention

In some programs such as cyc, the FECC-IRB does not provide as much IPC improvement (only 1.73% on average), because the ALU bandwidth may not be the significant bottleneck of the pipeline, which can be concluded from Table 4. Since the ALU bandwidth is not that much of a problem (The ALU instructions in cyc is lower compared with other testing programs), there is not enough contention that can be released by FECC-IRB, which makes the effect seems not obvious. In contrast, some programs such as str, the FECC-IRB provides a IPC improvement by 8.21%, which is much higher than cyc. This is because the ALU instructions in the str is much more than in the cyc.

In general, the ALU contention level varies according to testing programs, which is a important factor that decide the effect of our FECC-IRB scheme. However, from the ideal FECC-IRB, we can observe that this scheme can enhance the performance by reduce the error recovery penalty.

5.5 IRB Hit Rate

It is difficult to quantify the fault coverage of the considered DIE based schemes. However, IRB hit rate can indirectly evaluate the error detection and fault coverage of the FECC-IRB approach. So, techniques for enhancing IRB hit rate can be useful for alleviating the ALU bandwidth even further. There are three different methods:

(a) More entries are held to reduce the conflict and the IRB miss rate. Figure 9 shows the hit rate under different IRB size. The number of sets in the IRB is controlled from 8 to 1024. It indicates that larger IRB size can insure a higher hit rate and longer instruction life time in IRB.

(b) Even if the IRB size is limited, it can only keep the useful entries in a smaller IRB, so that non-useful entries do not conflict with the useful ones.

(c) It can proactively insert entries into the IRB to reduce the miss rate.

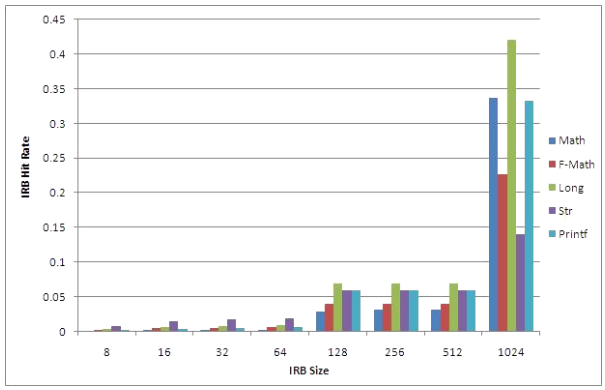

Fig. 9 IRB hit rate with different table size.

6. Conclusions

This paper proposed a novel solution to alleviate the performance loss of instruction-level temporal redundancy in an out-of-order superscalar microprocessor. It's an average up to 24.9% IPC loss on DIE over SIE for several benchmarks from Mibench. An important contributor to this performance gap is the in-sufficient ALU bandwidth since the same number of resources provisioned for SIE need to handle the workload imposed by DIE. The performance of the conventional DIE system is enhanced by storing the FECC into the IRB. D-Ins is directed to IRB, where it can directly pick up the FECC as long as that instruction has been previously encountered with the same set of operands, instead of execution on FUs. Otherwise, D-Ins is directed to the ALUs as the normal DIE. The temporal redundancy is provided by instruction reuse rather than re-execution. In addition, FECC stored in IRB can help to correct the results rather than cover them with re-execution.

According to the experiments, the ideal IRB hit rate provides a 9.1%-10.2% reduction in the IPC loss between SIE and DIE, with around 9.2% reduction on the average over several benchmarks form Mibench suite.

Furthermore, there are several ways to enhance the effect of the FECC-IRB approach for our further research. (a) Introducing the new encoding techniques which have high efficiency and stronger error correcting capacity can make FECC more powerful. (b) Enhancing the IRB hit rate, such as larger IRB, can provide more opportunities to correct the inaccurate results. (c) FECC calculation and error correcting process can be integrated into the superscalar pipeline as an extend execution stage.

References

[1] R. Baumann, "Soft errors in advanced semiconductor devices-part i: the three radiation sources," IEEE Trans. Device and Materials Reliability, vol.1, no.1, pp.17–22, March 2001.

[2] S. Borkar, P. Dubey, K. Kahn, D. Kuck, H. Mulder, S. Pawlowski, and J. Rattner, "Platform 2015: Intel processor and platform evolution for the next decade," Intel White Paper, Intel, March 2005.

[3] M. Franklin, "A study of time redundant fault tolerance techniques for superscalar processors," Defect and Fault Tolerance in VLSI Systems, 1995. Proceedings., 1995 IEEE International Workshop on,, pp.207–215, Nov. 1995.
[4] A. Parashar, S. Gurumurthi, and A. Sivasubramaniam, "A complexity-effective approach to alu bandwidth enhancement for instruction-level temporal redundancy," Proc. 31st Annual International Symposium on, Computer Architecture, pp.376–386, June 2004.
[5] D. Borodin and B. Juurlink, "Instruction precomputation with memoization for fault detection," Design, Automation Test in Europe Conference Exhibition (DATE), 2010, pp.1665–1668, March 2010.
[6] S.S. Mukherjee, J. Emer, and S.K. Reinhardt, "The soft error problem: An architectural perspective," Proc. 11th International Symposium on High-Performance Computer Architecture, pp.243–247, Washington, DC, USA, 2005.
[7] T. Karnik, B. Bloechel, K. Soumyanath, V. De, and S. Borkar, "Scaling trends of cosmic ray induced soft errors in static latches beyond 0.18 u," VLSI Circuits, 2001. Digest of Technical Papers. 2001 Symposium on, pp.61–62, 2001.
[8] S. Paul, C. Fang, Z. Xinmiao, and S. Bhunia, "Reliability-driven ecc allocation for multiple bit error resilience in processor cache," IEEE Trans., Comput., vol.60, no.1, pp.20–34, 2011.
[9] S. Rhee, C. Kim, J. Kim, and Y. Jee, "Concatenated reed-solomon code with hamming code for dram controller," 2010 Second International Conference on, Computer Engineering and Applications (ICCEA), pp.291–295, March 2010.
[10] R. Swamy, S. Bates, T.L. Brandon, B.F. Cockburn, D.G. Elliott, J.C. Koob, and C. Zhengang, "Design and test of a 175-mb/s, rate-1/2 (128,3,6) low-density parity-check convolutional code encoder and decoder," IEEE J., Solid-State Circuits, vol.42, no.10, pp.2245–2256, 2007.
[11] A. Timor, A. Mendelson, Y. Birk, and N. Suri, "Using underutilized cpu resources to enhance its reliability," IEEE Trans. Dependable and Secure Computing, vol.7, no.1, pp.94–109, Jan. 2010.
[12] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," 40th Annual IEEE/ACM International Symposium on, Microarchitecture, 2007. MICRO 2007. pp.197–209, Dec. 2007.
[13] J. Lee and K. Immink, "An efficient decoding strategy of 2D-ECC for optical recording systems," IEEE Trans. Consum. Electron., vol.55, no.3, pp.1360–1363, Aug. 2009.
[14] T. Austin, E. Larson, and D. Ernst, "Simplescalar: an infrastructure for computer system modeling," Computer, vol.35, no.2, pp.59–67, Feb. 2002.
[15] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," 2001 IEEE International Workshop on, Workload Characterization, 2001. WWC-4, pp.3–14, Dec. 2001.
[16] J. Ray, J. Hoe, and B. Falsafi, "Dual use of superscalar datapath for transient-fault detection and recovery," Proc. 34th ACM/IEEE International Symposium on, Microarchitecture, 2001. MICRO-34. pp.214–224, Dec. 2001.
[17] H. Dai and J. Zhang, "A framework for the correction of multi-bit errors in multi-core processors," 4th International Conference on, Embedded and Multimedia Computing, 2009. EM-Com 2009. pp.1–4, Dec. 2009.
[18] S. Jahinuzzaman, T. Shakir, S. Lubana, J.S. Shah, and M. Sachdev, "A multiword based high speed ecc scheme for low-voltage embedded srams," 34th European, Solid-State Circuits Conference, 2008. ESSCIRC 2008. pp.226–229, 2008.

Chao Yan was born in 1988. He has studied at the Department of Computer Science and Technology, Shandong University, China, since 2007. His research interests include fault-tolerant micro-architecture. He has published and presented 1 paper in TrustCom 2010, Hongkong.

Hongjun Dai was born in 1981. He received the B.S and Ph.D degree in the Department of Computer Science and Technology, Zhejiang University, China, in 2002 and 2007, respectively. He is currently a lecture in the Department of Computer Science and Technology, Shandong University, China. His research interests include micro-architecture, wireless sensor network and cyber-physical system. He has published more than 25 papers in journals and refereed international conference proceedings, and holds 3 national patents.

Tianzhou Chen received the B.S. degree in computer software in 1994, the PH.D degree in computer application in 1998, in the Department of Computer Science and Technology, Zhejiang University, China. He is currently a full professor of computer science at Zhejiang University, and the vice director of computer systems engineering research institute of Zhejiang University. He is a member of IEICE, IEEE, ACM, senior member of Chinese Institute of Electronics, and a council member of CCF (China Computer Federation). His current research interests include computer architecture, power-aware computing, hardware/software co-design and security. He has authored and coauthored 12 textbooks, published more than 200 research papers in major journals and international conferences, drawn up the 3 national standards, and holds 46 national patents.