Prioritizing Verification via Value-based Correctness Criticality

Joonhyuk Yoo and Manoj Franklin
Department of Electrical and Computer Engineering
University of Maryland at College Park, MD 20742
{juneki, manoj}@umd.edu

Abstract

Microprocessors are becoming increasingly susceptible to soft errors due to the current trends of semiconductor technology scaling. Traditional redundant multi-threading architectures provide good fault tolerance by re-executing all the computations. However, such a full re-execution significantly increases the demand on the processor resources, resulting in severe performance degradation.

To address this problem, this paper introduces a correctness criticality based filter checker, which prioritizes the verification candidates so as to selectively do verification. Binary Correctness Criticality (BCC) and Likelihood of Correctness Criticality (LoCC) are metrics that quantify whether an instruction is important for reliability or how likely an instruction is correctness-critical, respectively. A likelihood of correctness criticality is computed by a value vulnerability factor, which is defined by the numerically significant bit-width used to compute a result. The proposed technique is accomplished by exploiting information redundancy of compressing computationally useful data bits. Based on the likelihood of correctness criticality test, the filter checker mitigates the verification workload by bypassing instructions that are unimportant for correct execution. Extensive measurements prove that the LoCC metric yields quite a wide distribution of values, indicating that it has the potential to differentiate diverse degrees of correctness criticality. Experimental results show that the proposed scheme accelerates a traditional fully-fault-tolerant processor by 1.7 times, while it reduces the soft error rate to 18% of that of a non-fault-tolerant processor.

1. Introduction

Microprocessors are becoming increasingly vulnerable to soft errors due to continued semiconductor technology scaling. Redundant Multi-Threading (RMT) [11, 15, 16, 22] and Dynamic Verification Implementation Architecture (DIVA) [2] are recently proposed approaches for concurrent error detection and recovery. RMT-like architectures use two separate threads and execute an instruction stream redundantly for perfect fault coverage, while DIVA-like architectures employ an in-order checker processor to re-execute computations of an out-of-order core processor. However, these fully redundant architectures significantly increase the demand on the processor resources and result in severe performance degradation [8, 10, 25].

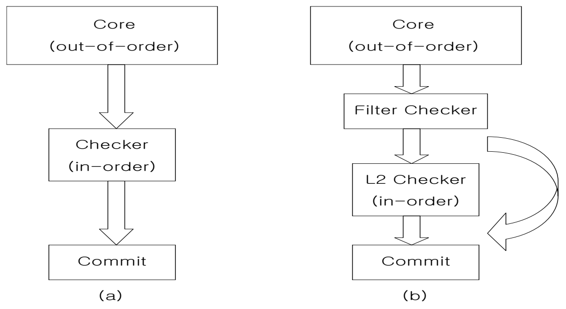


Figure 1. Comparison Between (a) Traditional DIVA and (b) Proposed Checker Hierarchy.

This paper addresses such a performance slowdown problem and presents a correctness criticality based filter checker, which is a pro-active verification management approach to mitigate the verification workload for high performance fault-tolerant microprocessors. The filter checker is motivated by cache hierarchies, which achieve both fast speed and large size by exploiting the principle of locality. Similarly, before the re-execution process starts at either the redundant thread or the checker processor, the filter checker quantifies a correctness criticality metric, estimating how likely an instruction is correctness-critical and prioritizes the verification candidates. Based on that likelihood, critical instructions may proceed to the second-level checker for further verification, while non-critical instructions may be directly forwarded to the commit stage without further

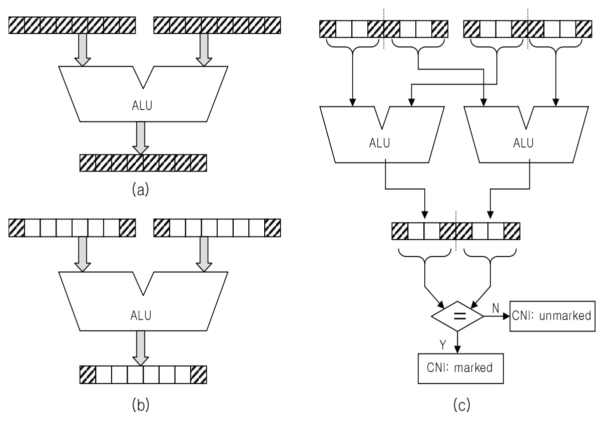


Figure 2. Value Compression for Fault Tolerance: (a) No value compression, (b) Value compression for identifying numerically significant bits, (c) Value compression for replicating numerically significant bits.

verification. Figure 1 describes a checker hierarchy in the DIVA-like architecture.

Computing the likelihood of correctness criticality is accomplished by exploiting information redundancy of compressing computationally useful data bits. Given a 64-bit binary representation, computing '1 + 2' has the same architectural vulnerability to soft errors as executing '334487364720 + 7458523762573', even though the former operation seems to be much simpler. With the support of a value compression technique that explicitly specifies the computationally meaningful bits, the simpler operation could be encoded with a smaller number of bits, effectively reducing its vulnerability to soft errors. In other words, if the information about the number of computationally significant bits in representing the value could be available, the former operation would become less vulnerable to soft errors than the latter. Numerical significance hints let the filter checker prioritize the verification candidates, resulting in less fault coverage loss than skipping them blindly. Figure 2 illustrates this idea. Without value compression, all data bytes in Figure 2 (a) are numerically significant ones. Figure 2 (b) and (c) present two value compression techniques for identifying or replicating numerically significant bits, respectively. Once the value compression is done, the unshaded bytes become numerically insignificant because they are not required for executing a correct computation, while only the shaded bytes are necessary and important. Because the unshaded bytes are computationally meaningless by information redundancy of compressing numerically significant bytes, soft errors caused by any particle strikes to the unshaded bytes cannot affect the correct operation. Our proposed filter checker marks a Correctness Non-criticality Indicator (CNI) flag to indicate if its result is not critical from the overall correctness point of view and is not required for further verification by the second-level checker.

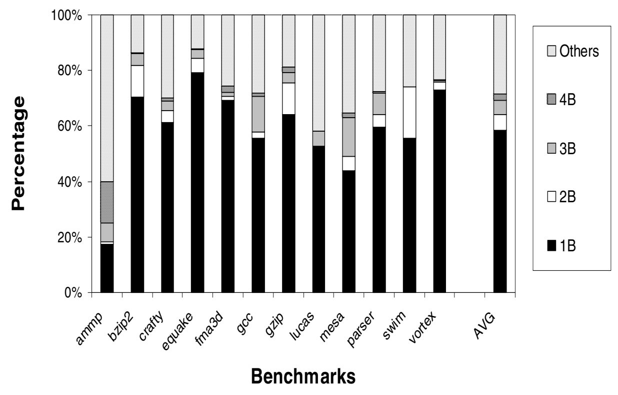


Figure 3. Percentage of Computationally Meaningful Bytes.

This paper presents one application of compressing numerically significant bits and of the correctness criticality based filter checker design. This paper is organized as follows. Section 2 investigates whether the number of computationally meaningful bytes is high and presents two value significance compression techniques, which are Value Identification (VI) and Value Replication (VR). Exploiting correctness criticality metrics for fault tolerance is proposed in Section 3. The section defines Binary Correctness Criticality (BCC) and Likelihood of Correctness Criticality (LoCC) metrics and presents an LoCC-based filter checker. The proposed LoCC-based filter checker is evaluated using the methodology described in Section 4. Finally, Section 5 shows that the proposed LoCC-based filter checker significantly improves both performance and reliability.

2. Significance Compression for Fault Tolerance

Narrow value compression is a special case of compressing numerically significant bits. More precisely, narrow values have a simple high order form—either all zeros or all ones. The information about all zeros or all ones can be encoded with a few bits by a simple compression technique. The distribution of narrow values has been found to be very non-uniform. The observation that a large percentage of the values produced and consumed in a processor are narrow [4], is utilized by several value compression techniques, such as designing very low power pipelines [5] and cache energy reduction [1, 7, 23]. While previous studies employ value significance compression for low power design, this paper exploits it for fault tolerance.

To investigate whether the number of computationally meaningful bytes is high, this paper measures the distribution of numerically significant bits over SPEC2000 benchmark suite. Figure 3 shows the percentage of computation-

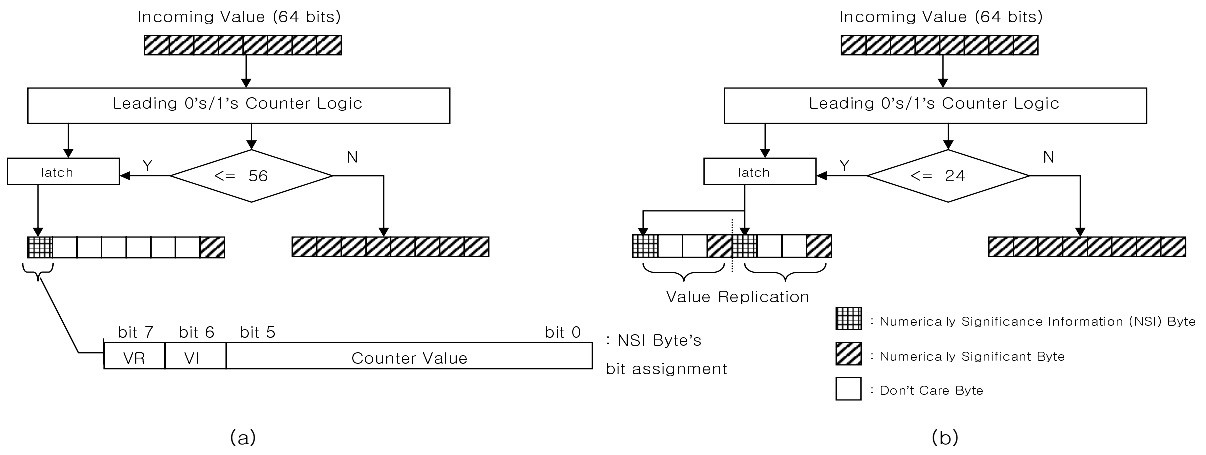

Incoming Value (64 bits)
Leading 0's/1's Counter Logic
latch
Y
<= 56
N
bit 7
bit 6
bit 5
bit 0
VR
VI
Counter Value
: NSI Byte's bit assignment
(a)
Incoming Value (64 bits)
Leading 0's/1's Counter Logic
latch
Y
<= 24
N
Value Replication
: Numerically Significance Information (NSI) Byte
: Numerically Significant Byte
: Don't Care Byte
(b)

Figure 4. Significance Compression Encoder: (a) Value Identification (VI), (b) Value Replication (VR).

ally meaningful bytes in each program. The data in the figure demonstrate that a wide range of application programs have a large number of narrow values and that the distribution of numerically significant bits is strongly biased. The '1B' label in the legend indicates the percentage of operands that can be represented by one byte. Similarly, '2B' denotes the percentage of operands represented by two bytes. We can see that roughly 60% of all operands are less than 8 bits wide and 36% have just one computationlly meaningful bit! This means that many of the upper order bits in the operand values are computationally meaningless. Hence, a significance compression technique can reduce the vulnerability to soft errors by storing the information about computationally meaningful bit widths in the conventional value-holding structures such as register file and data cache.

In order to make use of the observed value significance distribution, we propose a significance compression technique for narrow values. To effectively capture instances in which leading zeros or ones are not in multiple of 8 bits, we use a bit granularity instead of a byte granularity. Given a 64-bit architecture, a leading zeros or ones counter logic counts the leading bits and encodes its counting value through a 64-to-6 line priority encoder with inputs exclusive-OR-ed for adjacent bits of the value to determine a bit position where the value changes from 0 to 1 or 1 from 0. The number of leading zeros or ones can be obtained by negating the output of the encoder. Instead of using additional bits to keep this number, this paper proposes a scheme that reuses the Most Significant Byte (MSB), which is unnecessary if the counter value is less than 56.

Figure 4 illustrates two encoding techniques for narrow values. Figure 4 (a) shows a Value Identification (VI) encoding scheme in which a narrow value is defined by a value of width less than 56 bits. The upper two bits of the MSB are narrow value indicator bits to indicate whether the value is encoded or not and which encoding methods are used for significance compression. The other six bits in the MSB represent the number of leading zeros or ones counted in the previous counter logic. Even though the Value Identification encoding scheme reduces the architectural vulnerability of value holding structures, it introduces an one-byte overhead. Soft errors can still happen in the unprotected parts of the compressed value. Therefore, we present another Value Replication (VR) encoding scheme to provide redundancy for it by replicating the value (depicted in Figure 4 (b)). In this case, a narrow value is defined by a value of width less than 24 bits with the Value Replication encoding scheme.

3. Value-based Correctness Criticality

This section presents a simple mechanism that effectively reduces the vulnerability to soft errors in a microprocessor, which is motivated by the fact that many of the produced and consumed values in the processors are narrow and their upper order bits are meaningless. Therefore, soft errors caused by any particle strikes to those higher order bits can be avoided by simply compressing these narrow values. Our approach exploits a Value-based Correctness Criticality (VCC) property and manages unnecessary redundancy pro-actively.

The correctness criticality of an instruction is defined by the fraction of correctness-critical bits in the instruction. Correctness-Critical (CC) bits are those that are required for architecturally correct execution, and Correctness-Non-Critical (CNC) bits are those that do not affect correct execution. For example, consider an NOP instruction. Clearly, the only correctness-critical bits in the NOP are the opcode bits that distinguish it from non-NOP instructions; the remaining bits are correctness-non-critical. Another exam-

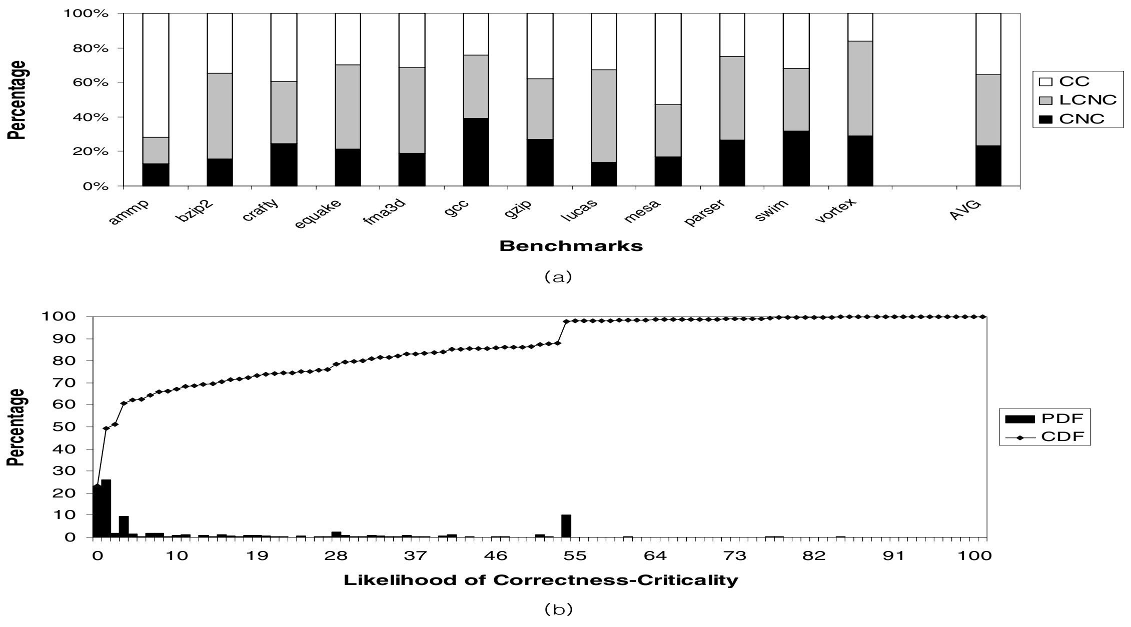


Figure 5. Likelihood of Correctness Criticality: (a) Percentage of LCNC instructions, (b) Density function of LoCC values.

ple of a correctness-non-critical instruction is a dynamically dead instruction, in which the result is not used. Such a Binary Correctness Criticality (BCC) metric is associated with the verification candidates by the filter checker for bypassing instructions that are unimportant for reliability. In the reorder buffer, a new Correctness Non-criticality Indicator (CNI) bit is added to indicate if its result is not critical from the overall correctness point of view. Marked instructions are directly passed to the commit stage, while un-marked instructions proceed to the second-level checker for further verification. Unfortunately, the number of correctness-non-critical instructions is not very high and only about 20% of the dynamic instructions can be marked according to our measurement. Instead, this paper uses a Likelihood of Correctness Criticality (LoCC) metric, which indicates how likely an instruction is to be correctness-critical. The filter checker prioritizes the verification candidates based on that likelihood.

The Likelihood of Correctness Criticality (LoCC) is quantified by computing a Value Vulnerability Factor (VVF), which is defined by the numerically significant bit-width used to compute a result. The VVF metric of a value is defined by the probability that a bit flip caused by a particle strike on any bit in the value will result in an erroneous behavior in the executed computation. Therefore, it can be estimated as a ratio of the number of leading zeros or ones to the total number of bits. The LoCC metric for an instruction is given by a scaled sum of the VVFs of all the source operands of that instruction. With the support of the value compression technique introduced in Section 2, the information about the computationally significant bits is already propagated along the data path to compute an execution result. Therefore, we can compute the LoCC metric for each dynamic instruction. The filter checker can utilize this value to speculate how likely the instruction is correctness-critical. Given two instructions I_a and I_b, if $LoCC(I_a) > LoCC(I_b)$, then instruction I_a is more likely to be correctness-critical than I_b. For simplicity, an LoCC of 50% is assigned to an instruction if 50% of the value widths averaged over all source operands of that instruction is computationally meaningful.

For the verification priority determined by a filter checker, the computed LoCC values are compared with a threshold LoCC value, which is set to 5 in this paper, to mark the Likely Correctness Non-Critical (LCNC) instructions. In other words, a type of hypothesis testing with a likelihood function is done to speculate how likely an instruction is to be correctness-non-critical. Figure 5 (b) shows the distribution of the LoCC metric over the SPEC2000 benchmark suite. Its density function shows that the LoCC metric yields a wide distribution of values, which indicates the potential to distinguish not just 0-1 correctness criticality but various degrees of correctness criticality. Figure 5 (a) shows the percentage of LCNC instructions determined by a filter checker. Being able to distinguish between CC and LCNC instructions by exploiting the LoCC values (which are equally CC instructions in the case of 0-1 correctness criticality), the filter checker can afford to mark more instructions without sacrificing the fault coverage. To understand the benefit of being able to distinguish

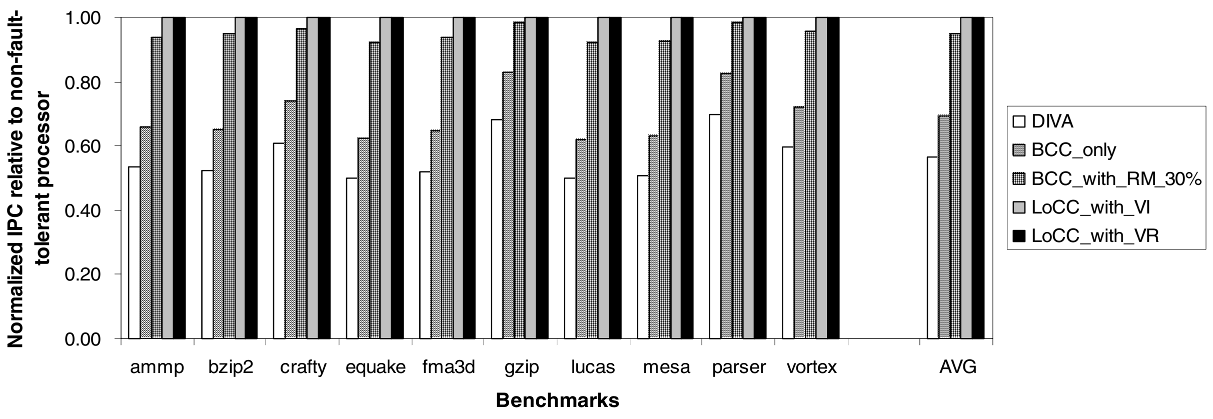


Figure 6. Normalized IPC (Instructions Per Cycle) for Five Different Dynamic Verification Schemes.

between CC and LCNC instructions based on the LoCC values, it is useful to view an LoCC value as a measure of expected penalty of skipping verification when any particle strike happens in the marked instructions. For example, if a 10% fault coverage loss results when a defective instruction with an LoCC value of 50% is skipped for verification, a wrong decision on correctness criticality speculated by a filter checker for an instruction with an LoCC value of 20% would incur a 4% fault coverage loss. By preferring the latter instruction as a marking candidate, the filter checker can save about 6% fault coverage loss.

4. Experimental Methodology

To evaluate the proposed correctness criticality based filter checker, we simulated a dual-core Chip Multi-Processor (CMP) with one complex out-of-order core processor and one simple in-order checker processor. Although the experiment is done only for a DIVA-like architecture, the same methodology could be applied to other architectures and the results would be similar. The simulator used in this paper was derived from the M5 Simulator System release 1.1 [14], which we modified to model the proposed correctness criticality based filter checker. We collected results for twelve of the SPEC2000 benchmarks—six integer and six floating-point applications. We selected simulation regions by using SimPoint [18] to analyze the first 16 billion instructions of each benchmark, and picked the earliest representative region reported by SimPoint. Then, we fast-forwarded each benchmark to its representative region, and finally turned on our simulations for 10 million committed instructions to be simulated.

Table 1 presents the hardware parameters for the baseline core processor. The second level checker processor has a 4-issue checker pipeline, which is an in-order single CPI machine with its own register file, functional units, and L1 cache of the same type as the core processor.

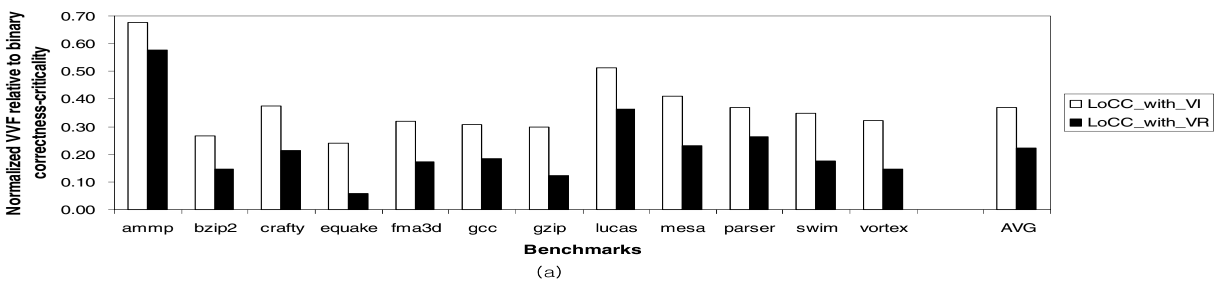
Parameter	Value
Fetch Queue Size	32 instructions
Fetch/Decode/Commit Width	4 instructions
Branch Predictor	4K entry BTB and hybrid predictor
Return Address Stack Size	16 entries
Physical Register File	128 INT, 128 FP
Issue Width	4 INT, 4 FP
Issue Queue Size	32 INT, 32 FP
Load-Store Queue / Reorder Buffer	64 / 256 entries
INT Functional Units	6 ALU, 2 MUL/DIV
FP Functional Units	4 ALU, 2 MUL/DIV
L1 I-Cache	64K 2-way set-associative, 64B line
L1 D-Cache	64K 2-way set-associative, 64B line
L2 Cache (Shared)	2M 32K set-associative, 64B line

Table 1. Baseline Core Processor Hardware Parameters.

Traditionally, Mean Time To Failure (MTTF) has been used as a reliability metric for processors. It is defined as the reciprocal of the Soft Error Rate (SER). The SER of a hardware structure can be estimated as the product of the raw error rate and the Architectural Vulnerability Factor (AVF) of the structure [12]. We used the methodology described in [3, 12] to compute the AVF of a processor. Because raw error rate is generally likely to be proportional to the area of each hardware component, the processor's AVF is obtained by a weighted sum of each component's AVF. From the published floor plan of the Alpha 21364 processor [19], we used the given area model to compute the weight.

5. Experimental Results

Impacts of our proposed Active Verification Management (AVM) on the processor performance and reliability are evaluated in this section. One advantage of using the proposed scheme is that it reduces the performance degradation due to checker processors. We explore how much the proposed scheme (for AVM with significance compression capability) is effective in preventing the checker from



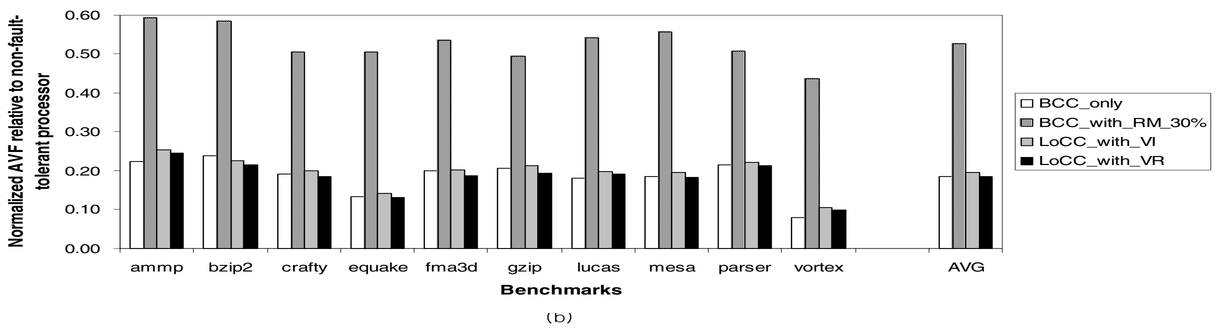


Figure 7. Soft Error Rate for Different Dynamic Verification Schemes: (a) Value Vulnerability Factor, (b) Architectural Vulnerability Factor.

becoming a performance bottleneck. Figure 6 shows the normalized Instructions Per Cycle (IPC) of several dynamic verification processors (with different correctness criticality metric) relative to a non-fault-tolerant processor. For each benchmark, 5 bars are shown, corresponding to the following 5 dynamic verification schemes: DIVA, BCC only, BCC with Random Marking (RM) of skipping blindly 30% of the instructions, LoCC with VI, and LoCC with VR.

Let us analyze the results presented in Figure 6. Because DIVA has no marking scheme, its performance is the worst. With no AVM, the checker's congestion badly affects the DIVA processor's performance, reducing it to 57% of that of a non-fault-tolerant processor. The AVM scheme based on 0-1 correctness criticality shows a limited performance improvement over DIVA. The 'BCC-with-RM-30%' in the legend indicates that AVM employs additional random marking of 30% with 0-1 correctness criticality based marking. Through bypassing instructions blindly from verification process, this scheme shows additional performance gain, but the fault coverage decreases compared with the former one. The results for both 'LoCC-with-VI' and 'LoCC-with-VR' schemes demonstrate that the proposed AVM with value compression capability effectively manages the verification congestion problem. VCC-based AVM always maintained the same performance as that of a system without a checker processor, and about 1.7 times the performance of a conventional DIVA processor.

According to the definition of the VVF metric, it indicates how much a value is vulnerable to soft errors. After summing each VVF for a value, the average VVF is computed by dividing it by the total number of values. With 0-1 correctness criticality, its VVF is defined by unity because it has no significance compression capability. Figure 7 (a) compares the normalized VVF of two significance compression encoding methods relative to 0-1 correctness criticality. Value Replication (VR) shows less VVF than Value Identification (VI). This comes from the fact that the VI scheme introduces an one-byte overhead, which is unprotected from soft errors, while the VR scheme is hardened by replicating the compressed value.

Figure 7 (b) shows the normalized processor AVF of AVM architectures with different correctness criticality metrics relative to a non-fault-tolerant processor. It indicates that AVM provides better fault tolerance than a non-fault-tolerant processor. As expected, 'BCC-with-RM-30%' shows severe decrease in fault coverage, compared with the other three AVM schemes, through bypassing instructions blindly from the verification process. It also shows that the proposed AVM scheme based on VCC has little impact on the soft error rate and provides the same level of reliability as the AVM based on 0-1 correctness criticality. Figure 7 (b) shows that the proposed VCC-based AVM can reduce the soft error rate by 18% of that of a non-fault-tolerant processor.

To validate the utility of the proposed VCC-based AVM, we need to capture both performance and reliability. Weaver *et al* [24] provide the metric *Mean Instructions To Failure (MITF)* to reason about the trade-off between

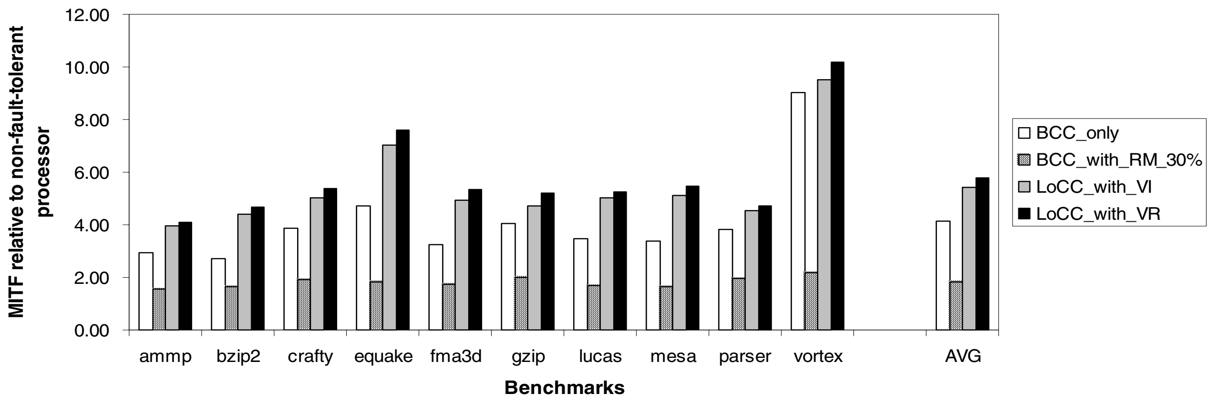


Figure 8. Mean-Instruction-To-Failure.

performance and reliability. At a fixed frequency and raw error rate of a processor, MITF is proportional to the ratio of IPC to AVF. A higher MITF means a larger amount of work done by the processor between errors. However, in the case that AVF is equal to zero, MITF goes to infinity and does not capture the performance improvement. Assuming that increasing MITF is desirable within certain bounds, MITF can be used to evaluate the trade-off. Figure 8 shows the normalized MITF of several AVM schemes relative to a non-fault-tolerant processor. VCC-based AVM provides better MITF than other AVM schemes based on 0-1 correctness criticality. It shows that the proposed VCC-based AVM can improve both performance and reliability 5.8 times better than that of a non-fault-tolerant processor.

6. Related Work

Computer architects have repeatedly shown that narrow values have a strongly biased distribution, say a few narrow values account for the majority of values used. They have observed that a large percentage of the values processed in the pipeline are narrow and can be encoded with less bits than full data width [1, 5, 7, 23] or just identified [4, 6, 9]. Several value compression techniques are taken advantage of by several micro-architectural techniques, such as designing very low power pipelines [5] and cache energy reduction [1, 7, 23]. While previous studies employ value compression techniques mainly for reducing energy consumption such as register file or data cache, this paper exploits them for improving both performance and reliability.

A host of concurrent system-level fault tolerance techniques has been proposed exploiting additional redundant execution processors [2, 11, 13, 15, 16, 21, 22]. Such redundant execution techniques employ Simultaneous Multi-Threading (SMT) [11, 15, 16, 22] or Chip Multi-Processing (CMP) [2, 21] to provide coarse-grain instruction replication with a modest amount of additional hardware support. Because the main thread shares the limited resources with the redundant thread, the performance degradation problem caused by redundant execution is common to all fully-fault-tolerant architectures. Most of previous studies have focused mainly on the achievement of reliability with little attention on the performance overhead incurred by re-execution. The proposed work considers performance of fault-tolerant processors to explore the trade-off between performance and reliability.

Running multiple threads in thread-level redundancy solutions places a significant pressure on the processor resources, resulting in a considerable performance loss. Recently, efficient resource management techniques [8, 10, 20] between the main leading thread and the redundant trailing thread have been proposed in an SMT implementation. Exploring a partial redundancy for soft error detection is similar to opportunistic transient fault detection [8], in which the redundancy is opportunistically adjusted according to the amount of Instruction Level Parallelism (ILP). Our approach exploits a value-based correctness criticality property and manages the redundancy pro-actively based on the biased distribution of Likelihood of Correctness Criticality (LoCC).

The concept of Binary Correctness Criticality (BCC) borrows from computing Architectural Vulnerability Factor (AVF) proposed by Mukherjee *et al* [12]. While AVF is used to statically compute the vulnerability factor of a hardware structure, our correctness criticality metric is applied to dynamic instruction stream for characterizing likely correctness critical instructions. Similar likelihood function has been applied for criticality analysis of program execution latency for clustering in superscalar processors [17]. Our LoCC metric is different in that criticality in terms of correctness viewpoint is dynamically exploited to prioritize verification candidates using the filter checker.

7. Conclusions

This paper presents a correctness criticality based filter checker, a pro-active verification management technique to mitigate the verification workload of fully-fault-tolerant processors. This is accomplished by exploiting information redundancy of compressing numerically insignificant bits. Exploiting value-based correctness criticality for the filter checker keeps its performance the same as that of high performance non-fault-tolerant processor, while significantly increasing reliability. Experimental results show that the proposed scheme accelerates a traditional fully-fault-tolerant processor by 1.7 times, while it reduces the soft error rate to 18% of that of a non-fault-tolerant processor.

References

[1] C. Aliagas, C. Molina, M. Garcia, A. Gonzalez, and J. Tubella. Value compression to reduce power in data caches. *Lecture Notes in Computer Science*, 2790, 2004.

[2] T. M. Austin. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proceedings of the 32nd International Symposium on Microarchitecture*, pages 196–207, Nov 1999.

[3] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan. Computing architectural vulnerability factors for address-based structures. *Computing Architectural Vulnerability Factors for Address-Based Structures*, 33, May 2005.

[4] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proceedings of the 5th International Symposium on High Performance Computer Architecture*, Jan 1999.

[5] R. Canal, A. Gonzalez, and J. E. Smith. Very low power pipelines using significance compression. In *Proceedings of the 33rd International Symposium on Microarchitecture*, Dec 2000.

[6] O. Ergin, O. Unsal, X. Vera, and A. Gonzalez. Exploiting narrow values for soft error tolerance. *IEEE Computer Architecture Letters*, 5, Jul-Dec 2006.

[7] M. Ghosh, W. Shi, and H.-H. S. Lee. Coolpression: A hybrid significance compression technique for reducing energy in caches. In *Proceedings of IEEE Systems-on-Chip Conference*, Apr 2004.

[8] M. A. Gomaa and T. N. Vijaykumar. Opportunistic transient-fault detection. In *Proceedings of the 32nd International Symposium on Computer Architecture*, Jun 2005.

[9] J. Hu, S. Wang, and S. G. Ziavras. In-register duplication: Exploiting narrow-width value for improving register file reliability. In *Proceedings of the 2006 International Conference on Dependible Systems and Networks*, Jun 2006.

[10] S. Kumar and A. Aggarwal. Reducing resource redundancy for concurrent error detection techniques in high performance microprocessors. In *Proceedings of the 12th International Symposium on High-Performance Computer Architecture*, Feb 2006.

[11] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. Detailed design and evaluation of redundant multithreading alternatives. In *Proceedings of the 29th International Symposium on Computer Architecture*, Jun 2002.

[12] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. M. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th International Symposium on Microarchitecture*, Dec 2003.

[13] J. Ray, J. C. Hoe, and B. Falsafi. Dual use of superscalar datapath for transient-fault detection and recovery. In *Proceedings of the 34th International Symposium on Microarchitecture*, Dec 2001.

[14] S. K. Reinhardt. Using the M5 Simulator. ISCA tutorials and workshops, University of Michigan, Jun 2005.

[15] S. K. Reinhardt and S. S. Mukherjee. Transient fault detection via simultaneous multithreading. In *Proceedings of the 27th International Symposium on Computer Architecture*, Jun 2000.

[16] E. Rotenberg. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In *Proceedings of the 29th International Symposium on Fault-Tolerant Computing*, Jun 1999.

[17] P. Salverda and C. Zilles. A criticality analysis of clustering in superscalar processors. In *Proceedings of the 38th International Symposium on Microarchitecture*, Nov 2005.

[18] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the 10th International Conference on Parallel Architectures and Compilation Techniques*, Sep 2001.

[19] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temparature-aware microarchitecture. In *Proceedings of the 30th International Symposium on Computer Architecture*, Jun 2003.

[20] J. C. Smolens, J. Kim, J. C. Hoe, and B. Falsafi. Efficient resource sharing in concurrent error detecting superscalar microarchitectures. In *Proceedings of the 37th International Symposium on Microarchitecture*, Dec 2004.

[21] K. Sundaramoorthy, Z. Purser, and E. Rotenberg. Slipstream processors: Improving both performance and fault tolerance. In *Proceedings of the 33rd International Symposium on Microarchitecture*, Dec 2000.

[22] T. N. Vijaykumar, I. Pomeranz, and K. Cheng. Transient-fault recovery using simultaneous multithreading. In *Proceedings of the 29th International Symposium on Computer Architecture*, Jun 2002.

[23] L. Villa, M. Zhang, and K. Asanovic. Dynamic zero compression for cache energy reduction. In *Proceedings of the 33rd International Symposium on Microarchitecture*, Dec 2000.

[24] C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt. Techniques to reduce the soft error rate of a high-performance microprocessor. In *Proceedings of the 31st International Symposium on Computer Architecture*, Jun 2004.

[25] J. Yoo and M. Franklin. The filter checker: An active verification management approach. In *Proceedings of the 21st International Symposium on Defect and Fault Tolerance in VLSI Systems*, Oct 2006.