Compiler-guided Register Reliability Improvement Against Soft Errors

Jun Yan, Wei Zhang
Department of Electrical and Computer Engineering
Southern Illinois University Carbondale
Carbondale, IL 62901
jun,zhang@engr.siu.edu

ABSTRACT

With the scaling of technology, transient errors caused by external particle strikes have become a critical challenge for microprocessor design. As embedded processors are widely used in reliability-sensitive environments, it becomes increasingly important to develop cost-effective techniques to improve the processor reliability against soft errors. This paper focuses on studying the register file immunity against soft errors since modern processors typically employ a large number of registers, which are accessed very frequently. As a result, soft errors occurred in registers can easily propagate to functional units or the memory system, leading to *silent data error* (SDC) or system crash.

To develop cost-effective techniques to fight soft errors for embedded processors, the first step is to understand the register file susceptibility to soft errors and its impact on the system reliability accurately. Toward this goal, this paper proposes the concept of register vulnerability factor (RVF) to characterize the probability that register transient errors can escape the register file and thus potentially impact the system reliability. Built upon the RVF concept, we then propose two cost-effective compiler-guided techniques to improve the register file reliability by lowering the RVF value. Our experiments indicate that on average, the RVF can be reduced to 9.1% and 9.5% by the hyperblock-based instruction re-scheduling and the reliability-oriented register assignment respectively, which can potentially lower the reliability cost significantly while protecting register files against transient errors.

Categories and Subject Descriptors

B.8.1 [**Hardware**]: Performance and Reliability—*Reliability, Testing and Fault-Tolerance*

General Terms

Design, Reliability, Experimentation

Keywords

Register File, Soft Errors, Reliability, Register Lifetime

1. INTRODUCTION

Transient errors caused by external radiation events have become a critical challenge for microprocessor design. Such errors occur at the operation time, which can lead to *silent data corruption* (SDC) or system crash if left without protection. It is expected that microprocessors will become increasingly susceptible to transient errors (also called soft errors) due to the shrinking feature size, lower supply voltage, higher frequency and higher density. Consequently, microprocessors must be protected against soft errors to meet the pre-defined reliability goals. While there are a number of techniques to deal with transient errors, they come at various penalties in performance, area, energy consumption and cost. For embedded systems or processors with stringent cost constraints, it is crucial to develop cost-effective techniques for fighting soft errors to meet the reliability requirement.

To achieve high performance, modern microprocessors typically employ register files with a large number of registers and multiple ports, which unfortunately are susceptible to soft errors. Moreover, registers are accessed very frequently, and thus soft errors occurred in the register file can propagate to the functional units or the memory hierarchy, potentially leading to severe system reliability problems. Previous work has shown that soft errors in register files can result in a large number of system failures [1]. To enhance register file immunity to soft errors, some processors use error detection and correction schemes in the register files. For instance, IBM G5 utilizes an ECC-based scheme [10] to protect the registers. While the ECC scheme can detect double-bit errors and correct single-bit errors, it is very costly in terms of performance and energy consumption. Tremblay et al. [5] show that a simple ECC operation can take up to three times the delay of a simple ALU operation. Although ECC computation and verification can be performed in the background, its energy consumption cannot be hidden. Actually, recent work shows that the energy consumption of ECC is approximately an order of magnitude larger than that of a register access [6]. Compared to ECC, a less expensive technique to enhance register file immunity is parity check. However, the reliability improvement is limited, because the parity-based schemes cannot correct any errors. Furthermore, both the parity and ECC-based schemes have an area overhead of

12.5%, which is non-trivial. Therefore, it is important to develop cost-effective techniques to enhance register file reliability without significant impact on cost, performance and energy consumption, especially for embedded processors.

In this paper, we firstly study the register file susceptibility to soft errors by defining a new metric — Register Vulnerability Factor (RVF), which characterizes the probability that register transient errors can escape the register file and thus potentially impact the system reliability. The RVF can be used to understand the reliability requirement of register files accurately to avoid the over-protection or under-protection. We then propose two compiler-guided techniques to increase register reliability by performing instruction re-scheduling and reliability-oriented register assignment with a partially ECC-protected register file. Our experiments indicate that on average, the hyperblock-based instruction re-scheduling can reduce the RVF to 9.1% and the reliability-oriented register assignment with partial ECC protection can reduce the RVF less than 10%. Built upon these two techniques, we propose a hybrid approach to reduce the RVF further. Our experimental results show that the hybrid approach can reduce the average RVF to 6.1% with only 4 out of 64 registers covered by ECC, leading to substantial improvement of register reliability against soft errors without significant impact on cost or performance.

The rest of this paper is organized as follows. Section 2 introduces the concept of register vulnerability factor. Section 3 presents two compiler-guided techniques to improve register file reliability against transient errors by reducing the register vulnerability factor. Section 4 explains the evaluation methodology. The experimental results are given in section 5. Section 6 discusses the related work and section 7 concludes the paper.

2. REGISTER VULNERABILITY FACTOR

While it is critical to protect the register file against soft errors, not all soft errors occurred in the register file can lead to visible system faults. Over-estimation of the soft error problem can result in over-design of the protection mechanisms, which will increase the reliability cost eventually. On the other hand, insufficient protection of register files will make the system unreliable and thus is useless. As a result, designers must accurately measure the probability that register soft errors can impact other system components and thus lead to erroneous final output. Recently, Mukherjee et al. [2] proposed the concept of architectural vulnerability factor (AVF), which is defined as the probability that a fault in a processor structure will lead to a visible error in the final output of program. In general, the AVF provides designers an accurate estimate of the soft error rate for various hardware components for making cost/reliability trade-offs. While the concept of AVF can also be applied to the register file, it fails to exploit the fact that soft errors in the register file can be automatically overlapped by the new values written to the register file. If a value with soft errors is written before it is read, it will have no impact on the system output. Toward the goal to measure register file susceptibility to soft errors accurately and quantitatively, we define the register vulnerability factor (RVF) to be the probability that a soft error in registers can be propagated to other system components (i.e., functional units, memory). In contrast to the concept of AVF [2], which focuses on the effect of soft error propagation, the RVF concentrates on the probability of soft error propagation to other hardware elements. It should be noted that even if a soft error occurred in the register file is consumed by an instruction, it may still not affect the final output since this instruction may be mis-speculated. Indeed, such effects can be easily captured by the AVF [2], thus this paper focuses on examining the RVF. Obviously, the RVF and the AVF can be combined to select the most cost-effective techniques to increase the register file reliability against soft errors.

Since processors only employ a limited number of architecture registers while programs typically use a large number of values, multiple values can be stored in the same register as long as their lifetimes do not overlap. In general, a value is first written into a register, then it is read by one or more times and finally another value is written into the same register, which finishs the lifetime of the old value and begins the lifetime of the new value. As depicted in figure 1, we can divide the accesses to register files into four different patterns (or intervals), namely, the write-read (W-R), read-read (R-R), read-write (R-W) and write-write (W-W) patterns (note that the read/write mentioned in this paper refers to the corresponding operations on register values, including but not limited to the load/store instructions, which operate on the data from the memory hierarchy). Among these four patterns, the register file is only susceptible to soft errors during the W-R and R-R intervals. In contrast, the soft errors occurred during the R-W and W-W intervals can be overlapped by the latter write operations, and hence will not impact other system components. It is widely accepted that fault-inducing particle strikes are randomly and uniformly distributed [2], therefore, the probability that a soft error in registers can be propagated to other system components can be computed as the average ratio that the register values are exposed to the susceptible intervals (i.e., W-R and R-R), as described in Equation (1). In this Equation, RV_i represents any register value, the $SusceptibleTime(RV_i)$ represents the time intervals that RV_i is exposed to the susceptible intervals (i.e., W-R and R-R intervals for RV_i), and the $Lifetime(RV_i)$ represents the lifetime of RV_i, which is time interval between the time that a register is allocated for RV_i and the the time it is overlapped by another value. Since both the $SusceptibleTime(RV_i)$ and $Lifetime(RV_i)$ can be easily obtained from a performance simulator, it would be straightforward to compute the RVF.

$$RVF = \frac{\sum SusceptibleTime(RV_i)}{\sum Lifetime(RV_i)} \tag{1}$$

The RVF indicates the probability that register soft errors can spread to other hardware elements and thus impact the system output. The higher the RVF, the lower the register file reliability, and hence more expensive techniques are needed to fight soft errors. Measuring the RVF is not only useful to understand the reliability requirement of register files accurately for avoiding both the over-protection or under-protection, *it also opens up avenues for software (e.g., compiler) to enhance register file reliability by reordering the read/write operations to reduce the RVF.* In contrast, traditional software optimizations mainly focus on performance. Therefore, the RVF allows compilers to consider both performance and reliability to optimize the register access patterns. Such a software based approach has no hardware overhead, which is fundamentally different from traditional space redundancy or information redundancy techniques.

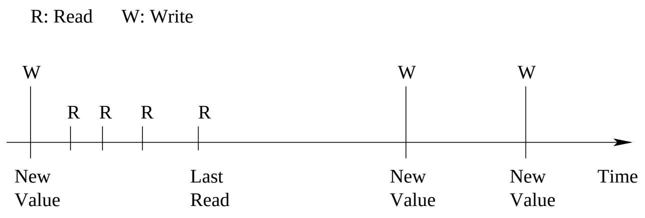

Figure 1: Register access patterns.

3. TECHNIQUES TO REDUCE REGISTER VULNERABILITY FACTOR

There are a number of research efforts on improving reliability of various system components of processors in the literature, such as the techniques to address soft errors for main memories [3, 12], caches [7, 8], and the datapath [9, 10, 11]. However, little work has been done to explore the impact of soft errors on register files, which is accessed very frequently and thus can significantly impact the system reliability if not protected. Recently, Memik et al. [4] proposed a scheme to replicate the register values into the physical registers at runtime to increase the register file reliability. Such a technique can exploit the additional physical registers efficiently to benefit reliability, however, it cannot be applied to processors without dynamic renaming support, such as VLIW architectures, which are increasingly used in embedded systems. By comparison, this paper proposes two compiler-guided techniques to improve the register file immunity to soft errors, which can be applied to embedded processors with or without physical Based on the concept of RVF, the first technique aims at enhancing register file reliability by re-scheduling the register read/write operations to reduce the RVF value without impacting the performance. The second technique assumes that a fraction of the register file employ the ECC code and thus we modify the register allocator to protect the registers that are most susceptible to soft errors based on the RVF profiling results. Built upon these two techniques, we also propose a hybrid scheme that can reduce the RVF further to improve the register file immunity to transient errors.

3.1 Re-schedule Instructions to Reduce RVF

Since registers are only susceptible to transient errors during the W-R and R-R intervals, the RVF can be reduced by delaying the write operations as late as possible and scheduling the read operations as early as possible. By doing this, the W-R and R-R intervals are shortened, while the R-W interval is lengthened, both of which can lead to a smaller RVF value and hence higher register file reliability. In order to not impact performance, we propose to re-schedule the read/write operations by exploiting the scheduling slacks [13]. Specifically, after the instruction scheduling phase, the compiler can analyze the dependence graph to identify the slacks for each scheduled code region. For instructions with slacks, the compiler then re-schedules the write operations as late as possible and the read operations as early as possible without increasing the critical path delay. Consequently, the RVF is reduced without compromising on performance. The advantage of this approach is that it is purely a software-based approach, which can increase the register file reliability without any hardware cost. However, the effectiveness of the approach depends on the flexibility to move the read/write operations in the scheduled code regions, which is constrained by data dependences and the critical path latency. To enhance the compiler's capability to reorder instructions, we also make use of the superblock scheduling [14] and hyperblock scheduling [15] algorithms to form larger blocks, in which the compiler has more flexibility to optimize the register access patterns for minimizing the RVF value.

3.2 Reliability-oriented Register Assignment with Partial ECC Protection

In contrast to the first technique that is pure software-based, the second scheme assumes that a certain number of registers have employed the ECC code to detect/correct transient errors. Since most soft errors are one-bit errors, the ECC code is sufficient to protect the register file against soft errors in most cases. However, since ECC is costly, we assume that only a small fraction of register file is covered by ECC. In order to minimize the RVF of a partially ECC-protected register file, we propose to modify the conventional register allocation algorithm by distinguishing the registers with ECC and the normal registers without ECC. We develop a profiling-based approach to direct the register allocation. Specifically, based on the RVF profiling for each register, the compiler selects the registers with the highest RVF values. If these registers are not protected by ECC, the compiler then re-assigns the registers so that the registers with ECC always have the highest RVF values. Since the most susceptible register values are now covered by ECC, the overall reliability of the register file can be improved substantially.

3.3 Hybrid Scheme

Based on the above two techniques, we propose a hybrid scheme that combines both the instruction re-scheduling and the reliability-oriented register assignment to further improve the register file reliability without significant cost increase. In the hybrid scheme, the compiler first performs the instruction re-scheduling to minimize the RVF based on hyperblocks and then re-allocates registers based on the profiling information and the number of registers covered by ECC. Compared with the pure software-based approach, such a hybrid scheme can improve the reliability further by exploiting the small number of registers that are protected by ECC. Likewise, compared with the pure register assignment based approach, the cost of the partially ECC-protected register file can be reduced by first applying the software-based instruction rescheduling to lower the RVF value as much as possible.

4. EVALUATION METHODOLOGY

We evaluate the register file reliability in a VLIW processor since VLIW architecture is increasingly used in embedded computing. We implement the method to compute RVF and the proposed techniques in the trimaran framework [16], which consists of both an advanced compiler and a VLIW simulator. The VLIW configuration used in our experiments contains four IALUs (integer ALUs), two FPALUs (floating-point ALUs), one LD/ST (load/store) unit and one branch unit. The register file consists of 64 general-purpose registers. The basic block scheduling algorithm is used as the default algorithm. We select ten benchmarks from the Mediabench [17] for the evaluation.

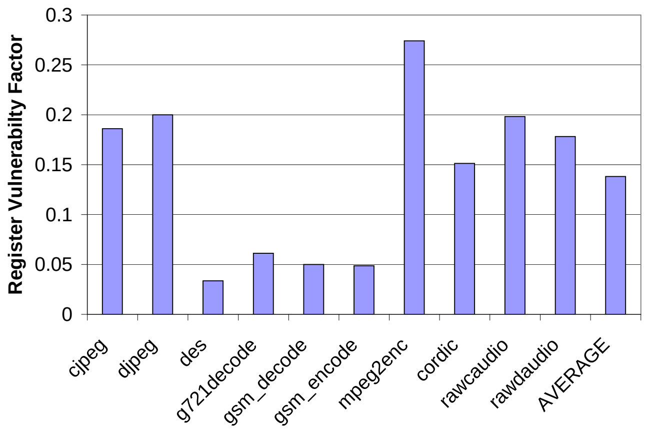

Figure 2: Register Vulnerability Factor for different benchmarks.

5. EXPERIMENTS

5.1 Register Vulnerability Factor Results

Figure 2 shows the RVF for different benchmarks. As can be seen, except for mpeg2enc, the RVF values of all other benchmarks are less than 20% and some RVF values are even less than 5%. Such low RVF values indicate that the majority of soft errors occurred in the register file can be automatically overlapped by the write operations, and hence have no impact on other system components or the system output. As a result, the reliability cost can be potentially reduced by choosing less expensive (and often less powerful) techniques to protect the register file while meeting the predefined reliability goal. These results also show that the register vulnerability factor is dependent on the application behaviors. Different applications access the register file in different patterns, leading to varied RVF values. Therefore, for embedded processors, which typically run a set of fixed applications, one can evaluate the register access patterns in the early design cycle to derive the RVF value, based on which the most cost-effective technique can be selected to protect the register file against soft errors.

5.2 Effect of Instruction Re-scheduling

Since a small RVF value implies high reliability, table 1 lists the RVF values by re-scheduling the register write operations as late as possible and the register read operations as early as possible based on the scheduling slacks. The second column in table 1 gives the RVF values of the original schedule, which uses the list scheduling algorithm [18]. Compared with the base scheme, the RVF values after the instruction re-scheduling decrease for all benchmarks. These results clearly indicate that the compiler can optimize the register access patterns to improve the register file immunity to soft errors. Nevertheless, we also find that the amount of RVF reduction is not significant, since the instruction reordering is limited within small basic blocks.

Figure 3 shows the RVF values of instruction re-scheduling

Benchmarks	Base	Re-schedule
cjpeg	0.186010	0.185490
djpeg	0.200022	0.194768
des	0.033647	0.033398
g721decode	0.061247	0.060142
gsm_decode	0.049989	0.049620
gsm_encode	0.048718	0.048518
mpeg2enc	0.274092	0.273101
cordic	0.151289	0.145255
rawcaudio	0.198224	0.197671
rawdaudio	0.178190	0.177265
Average	0.138143	0.136523

Table 1: RVF values of instruction re-scheduling compared with the base scheme.

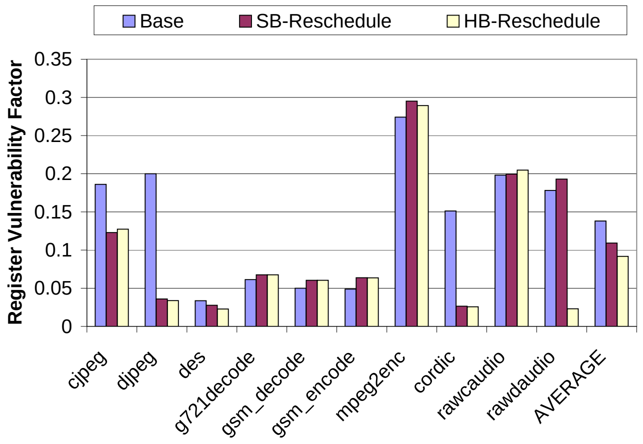

Figure 3: RVF of instruction re-scheduling based on superblock and hyperblock scheduling.

based on superblocks [14] and hyperblocks [15]. Since the superblocks and hyperblocks are much larger than the basic blocks, the compiler has more flexibility to move instructions without increasing the critical path delay. As a result, we observe that the RVF values of some benchmarks are reduced substantially. For instance, the RVF of djpeg decreases from 20% to 3.5% and 3.3% respectively for the superblock-based and hyperblock-based instruction re-scheduling approaches. On average, the superblock-based and hyperblock-based instruction re-scheduling can achieve the averaged RVF value as low as 10.9% and 9.1% respectively, which can be translated to the register file reliability improvement and the reliability cost reduction. We also find that for some benchmarks, the RVF values actually become larger, this is because the superblock scheduling and hyperblock scheduling also change the total execution cycles, compared with the basic block scheduling.

5.3 Effect of Reliability-oriented Register Assignment

Commercial microprocessors such as IBM G5 [10] have employed ECC to protect the register file against soft error. Although it is too costly to add ECC to each register for embedded processors, it is attractive to employ ECC to protect a limited number of registers that store the most critical data, since reliability is also critical to many embedded application and not all registers are accessed uniformly. Table 2 lists the RVF values of the reliability-oriented register assignment by varying the number of registers protected by ECC. As can be seen, the profiling-based register assignment is effective at reducing the RVF values. On average, with only 4 out of 64 registers protected by ECC, the RVF is reduced to 9.5%. With more registers covered by ECC, the RVF value can be further lowered. For instance, with 8 and 16 registers protected by ECC, the average RVF value is reduced to 6.5% and 3.2% respectively. Obviously, with more number of registers covered by ECC, the cost will also increase. Consequently, the designers need to make a tradeoff between the cost and the reliability to meet the design goals.

To evaluate the effectiveness of the proposed reliability-oriented register assignment scheme, we also make experiments by reducing the total number of general-purpose registers, so that each register is likely to be accessed more frequently. Table 3 gives the RVF values with 0, 2, 4, 8 and 16 registers covered by ECC and the total number of registers is 32. As we can see, the base RVF value is increased, since the register are generally read more frequently. By allocating the reliable registers with ECC to cover the most susceptible intervals, the RVF values can still be reduced effectively. For instance, with 4 register protected by ECC, the average RVF value is as low as 10%. Therefore, the reliability-oriented register assignment is effective at improving register file immunity to soft errors.

5.4 Effect of the Hybrid Scheme

The RVF values of the hybrid scheme are listed in table 4. As can be seen, the hybrid scheme can reduce the RVF values of different benchmarks significantly. With only 4 out of 64 registers protected by ECC, the RVF value is as low as 6.1% on average, implying substantial improvement of register file reliability.

6. RELATED WORK

Transient errors caused by external particle strikes have traditionally been a concern for systems that operate in highly noisy environments, but with the scaling of technology, they have become increasingly a challenge for microprocessor design. To understand the microprocessor vulnerability to soft errors, Kim and Somani [19] conducted fault injection experiments on picoJava-II in its RTL model and found large variations for different hardware blocks. Wang and Patel [20] studied the soft error sensitivity of a modern microprocessor similar to the Alpha 21264 through fault injection on a RTL model and reported that fewer than 15% of single bit errors in processor state result in software visible errors. Recently, Mukherjee et al. [2] proposed an approach to measure Architectural Vulnerability Factor (AVF) based on a performance model and reported that the AVFs of the instruction queue and execution units are 28% and 9% respectively for an Itanium2-like IA64 processor. In contrast to previous work, this paper focuses on studying the register file vulnerability to soft errors by exploiting the fact that register soft errors can be overlapped by the write operations, which cannot be captured by previous models. In addition, the RVF proposed in this paper can be combined with the AVF to understand the register reliability requirement even more accurately. In this paper, we also proposed several novel techniques to improve register file reliability without significant hardware cost.

To protect hardware components against transient errors, there are a number of techniques in the literature, but most of them focus on protecting the main memories [3, 12], caches [7, 8], and the datapath [9, 10, 11]. The most widely-used mechanisms to protect the storage units are parity and ECC, but come at the cost of area, energy and design time. Also, if the ECC computation is on the critical path, it may also impact the performance. Recently, Memik [4] proposed a scheme to replicate register values into the physical registers to increase the register file reliability. While this approach can utilize the available physical registers to enhance reliability, it can only be used for superscalar processors, where additional physical registers are employed to support the dynamic register renaming. In contrast, VLIW processors rely on compiler to manage the registers and they typically do not have additional physical registers, therefore, the approach proposed in [4] cannot be applied to VLIW-like processors, which do not have physical registers. By comparison, this paper proposes two compiler-guided techniques to improve the register file immunity to soft errors, which can be widely applied to a variety of processors.

7. CONCLUSION

With the shrinking feature size, lower supply voltage, higher density and frequency, soft errors are becoming an increasing challenge for microprocessor design. To protect processors against soft errors, the first step is to understand the vulnerability of different hardware components to soft errors. Based on the accurate estimate of the reliability requirement, the most cost-effective technique can be selected to meet the pre-defined reliability goal, which is of particular importance for embedded systems with cost constraints. While existing work mainly focuses on examining the impact of soft errors on main memory [3, 12], caches [7, 8] or datapath [9, 10, 11], this paper explores the register file reliability against soft errors, since registers are susceptible to transient errors and are accessed very frequently. In this paper, we propose the concept of Register Vulnerability Factor (RVF) to characterize the probability that register soft errors can be propagated to other system components and thus impact the final output. We also propose an approach to calculate the RVF based on the register access patterns.

Built upon the concept of RVF, we develop two compiler-guided techniques to improve register file reliability by decreasing the RVF value because a smaller RVF value indicates that the register file is less susceptible to soft errors. The first technique is a pure software-based approach, which exploits the scheduling slacks to move the register write operations as late as possible and the register read operations as early as possible. Our experiments demonstrate that the instruction re-scheduling based on hyperblocks [15] can reduce the RVF to 9.1% on average. The second technique targets register files that are partially protected by ECC. The proposed reliability-oriented register assignment aims at improving register file immunity to soft errors by protecting the most susceptible intervals based on the profiling information. We also propose a hybrid scheme built upon both these two techniques to further reduce the RVF. The experimental results show that the hybrid scheme can reduce the average RVF to 6.1% with only 4 out of 64 registers covered

Benchmarks	0 ECC	2 ECCs	4 ECCs	8 ECCs	16 ECCs
cordic	0.151288891	0.120040484	0.088792109	0.046742438	0.02429325
cjpeg	0.186009609	0.160762828	0.136325031	0.098380625	0.062427047
g721decode	0.061247047	0.038915094	0.0234375	0.008844359	0
des	0.033646875	0.02176425	0.019728578	0.016340734	0.010626719
gsm_decode	0.049989453	0.028505078	0.014410906	0.003800688	0.000105578
gsm_encode	0.048717953	0.028485578	0.011818906	0.003205125	0.000080125
djpeg	0.20002175	0.187089328	0.175444406	0.15350875	0.114980703
mpeg2enc	0.274091891	0.252926703	0.232575047	0.191884563	0.110947031
rawcaudio	0.198224297	0.166974391	0.135728609	0.073279031	0.000000375
rawdaudio	0.178189859	0.146939984	0.115695078	0.053284719	0
Average	0.138142763	0.115240372	0.095395617	0.064927103	0.032346083

Table 2: The RVF values of register assignment with 0, 2, 4, 8 and 16 registers protected by ECC. The total number of register is 64.

Benchmarks	0 ECC	2 ECCs	4 ECCs	8 ECCs	16 ECCs
cordic	0.091754031	0.068341563	0.049365656	0.033553969	0.013885313
cjpeg	0.189492656	0.155292063	0.12686875	0.084054844	0.033249313
g721decode	0.123968125	0.07768275	0.049233469	0.019457531	0
des	0.039867094	0.015828313	0.013607125	0.009330375	0.002692844
gsm_decode	0.099978906	0.057010156	0.028821813	0.007601375	0.000211156
gsm_encode	0.097435906	0.056971156	0.023637813	0.00641025	0.00016025
djpeg	0.132947156	0.107630688	0.091748375	0.069231281	0.038132406
mpeg2enc	0.221359594	0.180454875	0.1413295	0.095834719	0.024723313
rawcaudio	0.396448625	0.333948813	0.27145725	0.146558094	0.00000075
rawdaudio	0.356379719	0.293879938	0.231390156	0.106569438	0
Average	0.174963181	0.134704031	0.102745991	0.057860188	0.011305534

Table 3: The RVF values of register assignment with 0, 2, 4, 8 and 16 registers protected by ECC. The total number of register is 32.

Benchmarks	0 ECC	2 ECCs	4 ECCs	8 ECCs	16 ECCs
cordic	0.025700359	0.004397672	0.002273719	0.001621578	0.000885469
cjpeg	0.127385906	0.114456375	0.106415047	0.091380641	0.066774406
g721decode	0.067519422	0.044931984	0.030035609	0.010767469	0
des	0.022856078	0.010695469	0.009877625	0.008380328	0.005581578
gsm_decode	0.060325078	0.039777906	0.024668016	0.00641025	0.000171703
gsm_encode	0.063652719	0.044228188	0.026212469	0.006190266	0.000128078
djpeg	0.03372375	0.021106703	0.018647578	0.016207969	0.012148578
mpeg2enc	0.289249875	0.269204828	0.250841047	0.215085438	0.144025141
rawcaudio	0.204760625	0.173510797	0.142265484	0.086420625	0.027582234
rawdaudio	0.023031125	0.011191563	0.008446703	0.004637344	0.001563359
Average	0.091820494	0.073350148	0.06196833	0.044710191	0.025886055

Table 4: The RVF values of the hybrid scheme with 0, 2, 4, 8 and 16 registers protected by ECC. The total number of register is 64.

by ECC. As a result, the register file reliability is improved adequately without significant impact on cost. Moreover, all the techniques proposed in this paper can enhance register file immunity to soft errors without compromising on performance.

8. REFERENCES

[1] M. Rebaudengo, M. S. Reorda and M. Violantc. An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor. In Proc. of the Design, Automation and Testing Europe (DATE), 2003.

[2] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt and T. Austin. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. MICRO 2003.

[3] C. L. Chen and M.Y Hsiao. Error-correcting codes for semiconductor memory applications: a state of the art review. In Reliable Computer Systems - Design and Evaluation, pages 771-786, Digital Press, 2nd edition, 1992.

[4] G. Memik, M. Kandemir, O. OZturk. Increasing register file immunity to transient errors. In Proc of DATE 2005.

[5] M. Tremblay and Y. Tamir. Support for fault tolerance in VLSI processors. ISCS, 1989.

[6] R. Phelan. Addressing soft errors in ARM core-based SoC. ARM White Paper, Dec. 2003.

[7] S. Kim and A. K. Somani. Area efficient architectures for information integrity in cache memories. In Proc. of the International Symposium on Computer Architecture, 1999.

[8] C. Chen and A. K. Somani. Fault containment in cache memories for TMR redundant processor systems. IEEE Transactions on Computers, March 1999.

[9] T. Austin. DIVA: a reliable substrate for deep submicron microarchitecture design. MICRO, 1999.

[10] S.K. Reinhardt and S.S. Mukherjee. Transient fault detection via simultaneous multithreading. In Proc. of ISCA, 2000.

[11] J. Ray et al. Dual use of superscalar datapath for transient-fault detection and recovery. MICRO, 2001.

[12] T. J. Dell. A white paper on the benefits of chipkill-correct ECC for PC serve main memory. IBM, Nov 1997.

[13] W. Zhang, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, D. Duarte and Y. Tsai. Exploiting VLIW schedule slacks for dynamic and leakage energy reduction. In Proc. of MICRO 2001.

[14] W. W. Hwu, S. A. Mahlke, W. Y. Chen, P. P. Chang, N. J. Warter, R. A. Bringmann, R. G. Ouellette, R. E. Hank, T. Kiyohara, G. E. Haab, J. G. Holm, and D. M. Lavery. The superblock: an effective technique for VLIW and superscalar compilation. The Journal of Supercomputing, pp. 229–248.

[15] S. A. Mahilke, D. C. Lin, W. Y. Chen, R. E. Hank and R. A. Bringmann. Effective compiler support for predicated execution using hyperblock. In Proc. of the 25th International Symposium on Microarchitecture, pp.45-54, Dec. 1992.

[16] http://www.trimaran.org.

[17] C. Lee and M. Potkonjak, and W. H. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In Proc. the International Symposium on Microarchitecture, pp. 330–335, 1997.

[18] S. S. Muchnick. Advanced compiler design implementation. Morgan Kaufmann Publishers, 1997.

[19] S. Kim and A.K. Somaini. Soft error sensitivity characterization for microprocessor dependability enhancement strategy. In Proc. of the International Conference on Dependable Systems and Networks (DSN), 2002.

[20] N. J. Wang, J. Quek, T.M. Rafacz, S.J. Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In Proc of the International Conference on Dependable Systems and Networks (DSN), 2004.