Regular Paper
Journal of Computing Science and Engineering,
Vol. 5, No. 2, June 2011, pp. 121-130

Computing and Reducing Transient Error Propagation in Registers

Jun Yan
Mathworks, Boston, MA, USA **Jun.Yan@mathworks.com**

Wei Zhang*
Department of Electrical and Computer Engineering, Virginia Commonwealth University Richmond, VA, USA **wzhang4@vcu.edu**

Abstract
Recent research indicates that transient errors will increasingly become a critical concern in microprocessor design. As embedded processors are widely used in reliability-critical or noisy environments, it is necessary to develop cost-effective fault-tolerant techniques to protect processors against transient errors. The register file is one of the critical components that can significantly affect microprocessor system reliability, since registers are typically accessed very frequently, and transient errors in registers can be easily propagated to functional units or the memory system, leading to *silent data error* (SDC) or system crash. This paper focuses on investigating the impact of register file soft errors on system reliability and developing cost-effective techniques to improve the register file immunity to soft errors. This paper proposes the register vulnerability factor (RVF) concept to characterize the probability that register transient errors can escape the register file and thus potentially affect system reliability. We propose an approach to compute the RVF based on register access patterns. In this paper, we also propose two compiler-directed techniques and a hybrid approach to improve register file reliability cost-effectively by lowering the RVF value. Our experiments indicate that on average, 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, without sacrificing the register value integrity.

Category: Embedded computing

Keywords: Performance, Reliability, Transient errors, Compiler, Register files

I. INTRODUCTION

Recent research efforts indicate that microprocessors will become increasingly susceptible to transient errors (also called soft errors) due to shrinking feature size, lower supply voltage, higher frequency and higher density. Unlike hard errors that can be detected in the testing phase, transient errors occur at operation time, which can lead to *silent data corruption* (SDC) or system crash if left without protection. Consequently, microprocessors must be protected against soft errors to meet pre-defined reliability goals. A number of techniques (e.g., N Modular Redundancy), time redundancy or information redundancy (e.g., parity or error correction code [ECC]) fight transient errors, such as space redundancy. However, all these techniques incur various penalties in performance, area, energy consumption and cost. While some of these techniques are affordable for high-end products, it becomes increasingly necessary to develop cost-effective techniques to improve reliability against transient errors for embedded systems or processors with stringent cost constraints.

Soft error rate (SER) is typically described in failure in time (FIT). One FIT represents one error in a billion hours. Soft errors can be divided into two categories: undetected or detected. Undetected errors are also called SDC. The detected errors can be either recoverable or unrecoverable. The latter is referred to as *detected unrecoverable errors* (DUE), since the recoverable errors are generally not a concern. Accordingly, the soft error rate can be classified into SDC FIT and DUE FIT. Both the SDC and DUE errors can cause severe reliability problems. The soft

Open Access 10.5626/JCSE.2011.5.2.121 http://jcse.kiise.org

Received 25 January 2011, Accepted 4 March 2011
*Corresponding Author

error rate is currently often specified in terms of SDC and DUE numbers in industry [1].

While theoretically one would expect to kill all the soft errors, so the system is entirely error free, in practice, industry typically sets soft-error-rate budgets for their product based on target market requirements. For instance, IBM targets 114 SDC FIT, 4,566 system-kill DUE FIT and 11,415 processor-kill DUE FIT for Power4 processors [2]. Therefore, designers should develop or choose the most cost-effective mechanisms to meet the pre-defined reliability goal in terms of SDC FIT and DUE FIT to minimize reliability cost.

With the widespread use of load/store architecture, modern microprocessors often employ register files with a large number of registers and multiple ports that unfortunately are susceptible to soft errors. Moreover, since registers are accessed very frequently, soft errors occurring in the register file can easily propagate to the functional units or the memory hierarchy, leading to severe system reliability problems. Previous work has already shown that soft errors in register files can lead to a large number of system failures [3]. Some processors use error detection and correction schemes in the register files to enhance register file immunity to soft errors. For instance, IBM G5 utilizes an ECC-based scheme [4] to protect the registers. While the ECC scheme can detect double-bit errors and correct single-bit errors, it cannot correct double-bit errors. In addition, the ECC scheme is costly in terms of performance and energy consumption. Tremblay and Tamir [5] show that a simple ECC operation can incur three times the delay of a simple arithmetic logic unit (ALU) operation. Although ECC computation and verification can be performed in the background, the energy consumption cannot be hidden. Recent work indicates that the energy consumption of ECC is approximately an order of magnitude larger than that of a register access [6]. Therefore, ECC protection will be a very expensive mechanism for registers, especially for embedded processors with cost constraints. Compared to ECC, a less expensive technique to enhance register file immunity is parity check. However, reliability improvement by parity is limited, because the parity-based schemes cannot correct any errors or detect even-bit errors. Therefore, it is important to develop cost-effective techniques to enhance register file reliability without significantly affecting cost, performance and energy consumption, especially for embedded processors.

The first step is to understand the impact of register soft errors on system reliability to protect the register file against transient errors cost-effectively. Estimation based on raw register SER is too conservative, since not all register soft errors can affect system reliability. Overestimating the register reliability problem can lead to over-protection that will unnecessarily increase reliability cost. Similarly, underestimating the register reliability problem may result in under-protection, which will make the processors unreliable. In this paper, we study the register file susceptibility to soft errors by defining a new metric — register vulnerability factor (RVF). RVF characterizes the probability that register transient errors can escape the register file and thus potentially affect system reliability.

Based on the register access patterns and the assumption that soft errors distribute uniformly, we develop an approach to compute the RVF quantitatively, which can be used to estimate the reliability requirement of register files accurately to avoid over-protection or under-protection. We 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 built upon the RVF concept. Our experiments indicate that on average, 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%. Moreover, we propose a hybrid approach by integrating these two techniques to reduce the RVF further. Our experimental results show the hybrid approach can reduce the average RVF to 6.1% with only four 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 remainder of this paper is organized as follows. Section II introduces the concept of the register vulnerability factor. Section III presents two compiler-guided techniques to improve register file reliability against transient errors by reducing the register vulnerability factor. Section IV explains the evaluation methodology. The experimental results are given in section V.

Section VI discusses related work. Section VII concludes the paper.

II. REGISTER VULNERABILITY FACTOR

Register files are more resilient to transient errors than are conventional memory cells. However, as technology scales, the charge retaining capabilities of CMOS devices decrease, and more clock edges can occur during a given period. Thus, the window of vulnerability for a flip-flop being around its clock edges makes it more susceptible to soft errors at increased frequencies [7]. While it is important to protect the register file against soft errors early in the design cycle, one should be cautious not to overestimate this problem, which can lead to expensive and excessive protection. Design based on the raw SER of latches will over-estimate the register reliability problem, since not all soft errors occurring in the register file can lead to visible system faults. For instance, soft errors between two register write operations to the same register will be automatically corrected by the latter write operation. Therefore, designers must accurately measure the probability that register soft errors can affect other system components and thus lead to erroneous final output. Mukherjee et al. [1] proposed the concept of architectural vulnerability factor (AVF). AVR is defined as the probability that a fault in a processor structure will lead to a visible error in the final program output. In general, the AVF provides designers an accurate estimate of the soft error rate for various hardware components to make 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. We define the RVF to be the probability that a soft error in registers can be propagated to other system components (i.e., functional units, memory) toward the goal to measure register file susceptibility to soft errors accurately and quantitatively. RVF concentrates on the probability of soft error propagation to other hardware

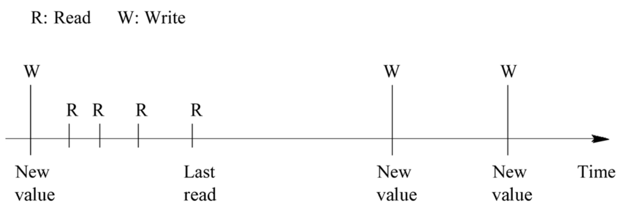


Fig. 1. Register access patterns.

elements, in contrast to the AVF concept [1] that focuses on the effect of soft error propagation. 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 miss-speculated. Indeed, such effects can be easily captured by the AVF [1]. 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.

Multiple values can be stored in the same register, as long as their lifetimes do not overlap, since processors only employ a limited number of architecture registers, while programs typically use a large number of values. In general, a value is first written to a register, then it is read once or more and finally another value is written to the same register, which finishes the lifetime of the old value and begins the lifetime of the new value. As depicted in Fig. 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 occurring during the R-W and W-W intervals can be overlapped by the latter write operations, and hence will not affect other system components. It is widely accepted that fault-inducing particle strikes are randomly and uniformly distributed [1]. Therefore, the probability that a soft error in registers can be propagated to other system components can be computed, as the average ratio to which 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 the time interval between the time that a register is allocated for RV_i and the time it is overlapped by another value. It would be straightforward to compute the RVF, since both the *Susceptible Time*(RV_i) and *Lifetime*(RV_i) can be easily obtained from a performance simulator.

$$\text{RVF} = \frac{\text{SusceptibleTime}(\text{RV}_i)}{\sum \text{Lifetime}(\text{RV}_i)} \tag{1}$$

The RVF indicates the probability that register soft errors can spread to other hardware elements and thus affect 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 more accurately to avoid both over-protection or under-protection, *it also opens up the avenues for software (e.g., compiler) to enhance register file reliability by reordering and optimizing the read/write operations to minimize the RVF*. In contrast, traditional soft- ware optimizations focus on performance. Therefore, RVF allows the compiler to consider both performance and reliability to optimize register access patterns. Such a software-based approach has no hardware overhead, which fundamentally differs from traditional space redundancy or information redundancy techniques.

We define a new metric called register file reliability factor (RFRF), based on the concept of RVF, which is the product of 1) the RVF, 2) the raw SER per latch, and 3) the number of latches per register file. As shown in Equation 2, N denotes the number of latches per register file and $\text{SER}_{\text{latch}}$ represents the raw soft error rate per latch that varied with different technology. Therefore, we can estimate the reliability of register file against transient errors more accurately by incorporating the RVF.

$$\text{RFRF} = \text{RVF} * \text{SER}_{\text{latch}} * \text{N} \tag{2}$$

III. 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. These include techniques to address soft errors for main memory [8, 9], cache [10, 11], and the datapath [4, 12, 13]. However, very little work has been done to explore the impact of soft errors on register files. Memik et al. [14] proposed a scheme to replicate register values into the physical registers to increase the register file reliability. However, such a technique cannot be applied to processors without physical registers, such as very long instruction word (VLIW) architectures, which are increasingly used in embedded systems. This paper, in comparison, proposes two compiler-guided techniques to improve the register file immunity to soft errors that can be applied to a wide variety of embedded processors. Based on the RVF concept, the first technique aims to enhance register file reliability by rescheduling the register read/write operations to reduce the RVF value without impacting performance. The second technique assumes that a fraction of the register file employs 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 propose a hybrid scheme that can reduce the RVF further to improve the register file immunity to transient errors.

A. Re-schedule Instructions to Reduce RVF

RVF can be reduced by delaying the write operations as late as possible and scheduling the read operations as early as possible, since registers are only susceptible to transient errors during the W-R and R-R intervals. Thus, 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

```
INPUT: A sequence of operations ("region") scheduled using a performance-oriented scheduler;
OUTPUT: A re-scheduled set of operations where slacks have been exploited to minimize RVF
Algorithm(region)
  begin
    compute_slack(region);
    list = build_slack_list(region);
    curr_max = -1;
    next_op = NULL;
    for each operation op in list do
      if (heuristic_rvf(op) > curr_max) then
        next_op = op;
        curr_max = heuristic_rvf(op);
      endif
    endfor
    if ((next_op ! = NULL && (curr_max > 0)) then
      stime = stime_old + slack(op);
      update_region(region,next_op,stime);
      Algorithm(region);
    endif
  end

Int heuristic_rvf(op)
  begin
    int gain,stime;
    stime = stime_old + slack(op);
    gain = compute_rvf(dest(op),stime) + compute_rvf(src1(op),stime) + compute_rvf(src2(op),stime);
    return gain;
  end
```

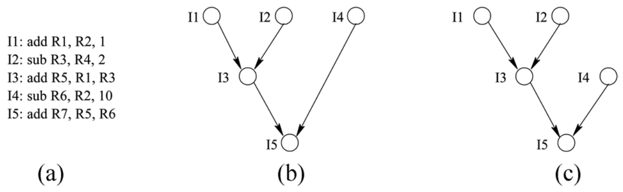
Fig. 2. Instruction rescheduling algorithm to reduce register vulnerability factor (RVF). We assume an ASAP scheduling algorithm, where each operation is scheduled as early as possible.

reliability. The movement of the register read or write operations, however, is subject to the data dependence between different operations. We propose to re-schedule the read/write operations by exploiting the scheduling slacks to not impact performance. Fig. 2 sketches the algorithm of the instruction re-scheduling.

This algorithm takes a region of code to schedule (*region*). *compute slack*() computes the slack for each operation in the region and *build slack list*() builds a list of operations with slacks. In the for-loop, we employ *a selection heuristic* to determine the most beneficial operation candidate for slack exploitation to minimize the RVF. Specifically, our selection heuristic evaluates each operation with a slack and calculates the potential gain if the associated slack is exploited.

The potential gain is the difference in RVF between the original schedule and the schedule after exploiting the slack, which is performed in the function *compute rvf*(). The *heuristic RVF*() function will calculate the potential gain of rescheduling by considering all the operands of the instruction, including the destination operands and two source operands. The added potential gain is returned to the main algorithm, which will select the operation with the largest potential gain (*curr_max* keeps the maximum RVF reduction so far). After selecting an operation with positive gain in RVF, the scheduler updates the code region, and calls itself with the updated region to exploit the remaining slack.

The complexity of this algorithm is O(n^2), where n is the number of instructions in the region. This complexity has the same order of magnitude, as some other widely used optimization phases of compilation, such as the instruction scheduling.



```
I1: add R1, R2, 1
I2: sub R3, R4, 2
I3: add R5, R1, R3
I4: sub R6, R2, 10
I5: add R7, R5, R6
```

Fig. 3. (a) Sample code (b) Original schedule (c) Re-schedule to reduce register vulnerability factor (RVF) by exploiting the slack of I4.

Therefore, the latency of the instruction re-scheduling can be tolerated by the compiler to generate better code by considering both register reliability and performance.

Fig. 3 shows an example of instruction re-scheduling, where instruction I3 is dependent on I1 and I2, and I5 is dependent on I3 and I4. Given sufficient resources, I1, I2 and I4 can be scheduled at the first cycle, I3 can be scheduled at the second cycle and I5 is scheduled at the third cycle, as shown in Fig. 3b.

As can be seen, I4 has one cycle slack, since it can be scheduled in the second cycle without increasing the critical path delay. Since I4 writes to register R6 and I5 reads register R6, we can re-schedule I4 to be executed in the second clock cycle, as shown in Fig. 3c. While R6 is susceptible to soft errors during two clock cycles in schedule (b), its susceptible interval is reduced to one clock cycle in schedule (c). Consequently, the RVF of R6 is reduced. By exploiting the scheduling slacks to move register write operations as late as possible and register read operations as early as possible, the RVF can be potentially lowered without compromising performance. The advantage of this approach is that it is purely a software-based approach, which can increase the register file reliability with no additional 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. We also make use of the superblock scheduling [15] and hyperblock scheduling [16] algorithms to form larger blocks, in which the compiler will have more flexibility to re-arrange and optimize the register access patterns to minimize the RVF value to enhance the compiler's capability to reorder instructions.

B. Reliability-oriented Register Assignment with Partial ECC Protection

In contrast to the first technique, which is purely software-based, the second scheme assumes that a certain number of registers have employed the ECC code, which can detect double-bit errors and correct single-bit errors. The ECC code is sufficient to protect the register file against soft errors in most cases, since most soft errors are one-bit errors. Therefore, we assume a single-bit soft error model in this paper. We assume that only a small fraction of register file is covered by ECC, because ECC is costly, especially for embedded processors. We propose to modify the conventional register allocation algorithm by distinguishing the registers with ECC and the normal registers without ECC to minimize the RVF of a partially ECC-protected register file. 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 (i.e., the registers with ECC will not be susceptible to soft errors during any access intervals), the overall reliability of the register file can be improved substantially.

C. Hybrid Scheme

We propose a hybrid scheme that combines both the re-scheduling and the reliability-oriented register assignment, based on these two techniques. In the hybrid scheme, the compiler firstly performs the instruction re-scheduling to minimize the RVF based on hyperblocks and then re-allocate 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, 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.

IV. EVALUATION METHODOLOGY

We evaluate the register file reliability in a VLIW processor, since VLIW architecture is increasingly used in embedded computing. We implement the proposed RVF-based techniques in the trimaran framework [17] that consists of both an advanced compiler and a VLIW simulator. A program flows through the frontend compiler IMPACT, the backend compiler Elcor, and the cycle-level VLIW processor simulator. IMPACT applies optimization level 4 (O4), which includes machine-independent classical optimizations and transformations to the source program; whereas Elcor is responsible for machine-dependent optimizations, including instruction scheduling and register allocation. The VLIW configuration used in our experiments has 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. Table 1 shows the default cache parameters. We assume each instruction word contains eight operations in the simulated VLIW processor. The basic block-scheduling algorithm is used as the default algorithm. We select ten benchmarks from Mediabench [18] for the evaluation.

Table 1. Default parameters used in our simulations

Parameters	Value
L1 instruction cache	32 KB direct-mapped
L1 instruction cache latency	1 cycle
L1 instruction cache block size	32 B
L1 data cache	32 KB 2-way set associative
L1 data cache latency	1 cycle
L1 data cache block size	32 B
Unified L2 cache	512 KB 4-way set associative
L2 cache latency	10 cycles
L2 cache block size	64 B
Memory latency	100 cycles

V. EXPERIMENTS

A. Register Vulnerability Factor Results

Fig. 4 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 occurring 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. Thus, the reliability cost can be potentially reduced by choosing less expensive (and often less powerful) techniques to protect the register file, while meeting the pre-defined reliability goal. These results also show that the register vulnerability factor is dependent on the application behaviour. 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

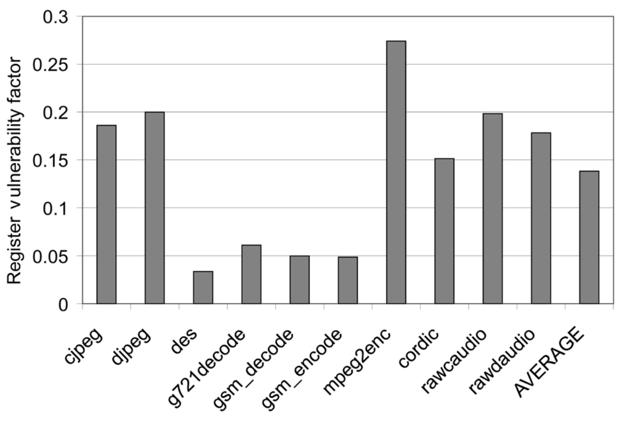


Fig. 4. Register vulnerability factor for different benchmarks.

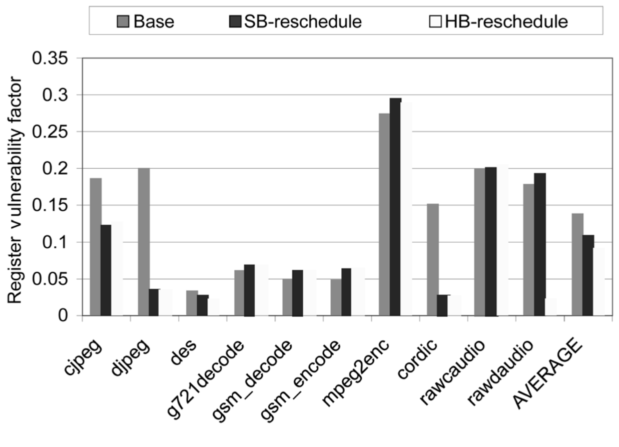


Fig. 5. Register vulnerability factor (RVF) of instruction re-scheduling based on superblock and hyperblock scheduling.

cost-effective technique can be selected to protect the register file against soft errors.

B. Effect of Instruction Re-scheduling

Table 2 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, since a small RVF value implies high reliability. The second column in Table 2 gives the RVF values of the original schedule that uses the list-scheduling algorithm [19]. The RVF values after instruction re-scheduling decrease for all benchmarks compared to the base scheme. 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 insignificant, since the instruction reordering is limited within small basic blocks.

Fig. 5 shows the RVF values of instruction re-scheduling based on superblocks [15] and hyperblocks [16]. The compiler has more flexibility to move instructions without increasing the critical path delay, since the superblocks and hyperblocks are much larger than the basic blocks. Thus, 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 an 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 become larger, because superblock scheduling and hyperblock scheduling also change the total execution cycles, compared to basic block scheduling.

Table 2. Register vulnerability factor values of instruction re-scheduling compared to the base scheme.

Benchmarks	Base	Re-schedule
cjpeg	0.186	0.185
djpeg	0.200	0.195
des	0.034	0.033
g721decode	0.061	0.060
gsm decode	0.050	0.050
gsm encode	0.049	0.049
mpeg2enc	0.274	0.273
cordic	0.151	0.145
rawcaudio	0.198	0.198
rawdaudio	0.178	0.177
Average	0.138	0.137

C. Effect of Reliability-oriented Register Assignment

Commercial microprocessors, such as IBM G5, [4] 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 applications and not all registers are accessed uniformly. Table 3 lists the RVF values of the reliability-oriented register assignment by varying the number of registers protected by ECC. The profiling-based register assignment is effective in reducing RVF values. On average, RVF is reduced to 9.5%, with only four out of 64 registers protected by ECC. The RVF value can be further lowered with more registers covered by ECC. For instance, with eight and sixteen registers protected by ECC, the average RVF value is reduced to 6.5% and 3.2%, respectively. Obviously, cost will also increase, with more registers covered by ECC. Consequently, the designers need to trade-off cost and reliability to meet design goals.

We also experiment reducing the total number of general-purpose registers, so that each register is likely to be accessed more frequently, to evaluate the effectiveness of the proposed reliability-oriented register assignment scheme. Tables 4 and 5 give the RVF values with 0, 2, 4, 8 and 16 registers covered by ECC for register files with 32 and 16 registers. The base RVF value is increased, since each register will be accessed more frequently with fewer registers. The RVF values can still be reduced effectively, by allocating the reliable registers with ECC to cover the most susceptible intervals. For instance, with four out of 32 registers protected by ECC, the average RVF value is as low as 10%. However, protecting four registers with ECC for a register file with 16 registers can only reduce the RVF to 13.7% on average. Nevertheless, the average RVF for a small register file is reduced more significantly by protecting more registers with ECC. For instance, with eight our of 64 registers covered by ECC, the average RVF is 6.5%, while with eight out of 32 or 16 registers protected by ECC, the average RVF is 5.8% and 5.9% respectively. That is, a larger portion of register will be covered by ECC, because for a smaller register file, leading to higher reliability. Obviously, if all the 16 registers are covered by ECC, the RVF value becomes zero under the single-bit error model, indicating high register reliability. Therefore, the reliability-oriented register assignment is quite effective in improving register file immunity to soft errors for varying numbers of registers.

Table 3. Register vulnerability factor values of register assignment with 0, 2, 4, 8, and 16 registers protected by ECC. There are 64 registers.

Benchmarks	0 ECC	2 ECCs	4 ECCs	8 ECCs	16 ECCs
cordic	0.151	0.120	0.089	0.047	0.024
cjpeg	0.186	0.161	0.136	0.098	0.062
g721decode	0.061	0.039	0.023	0.009	0.000
des	0.034	0.022	0.020	0.016	0.011
gsm decode	0.050	0.029	0.014	0.004	0.000
gsm encode	0.049	0.028	0.012	0.003	0.000
djpeg	0.200	0.187	0.175	0.154	0.115
mpeg2enc	0.274	0.253	0.233	0.192	0.111
rawcaudio	0.198	0.167	0.136	0.073	0.000
rawdaudio	0.178	0.147	0.116	0.053	0.000
Average	0.138	0.115	0.095	0.065	0.032

Table 4. The register vulnerability factor values of register assignment with 0, 2, 4, 8, and 16 registers protected by ECC. There are 32 registers.

Benchmarks	0 ECC	2 ECCs	4 ECCs	8 ECCs	16 ECCs
cordic	0.092	0.068	0.049	0.034	0.014
cjpeg	0.189	0.155	0.127	0.084	0.033
g721decode	0.124	0.078	0.049	0.019	0.000
des	0.040	0.016	0.014	0.009	0.003
gsm decode	0.100	0.057	0.029	0.008	0.000
gsm encode	0.097	0.057	0.024	0.006	0.000
djpeg	0.133	0.108	0.092	0.069	0.038
mpeg2enc	0.221	0.180	0.141	0.096	0.025
rawcaudio	0.396	0.334	0.271	0.147	0.000
rawdaudio	0.356	0.294	0.231	0.107	0.000
Average	0.175	0.135	0.103	0.058	0.011

Table 5. The register vulnerability factor values of register assignment with 0, 2, 4, 8, and 16 registers protected by ECC. There are 16 registers.

Benchmarks	ECC=0	ECC=2	ECC=4	ECC=8	ECC=16
cordic	0.094	0.047	0.035	0.020	0.000
cjpeg	0.185	0.132	0.090	0.045	0.000
g721decode	0.226	0.134	0.079	0.024	0.000
des	0.061	0.013	0.010	0.004	0.000
gsm decode	0.184	0.102	0.047	0.008	0.000
gsm encode	0.193	0.102	0.042	0.007	0.000
djpeg	0.118	0.065	0.047	0.024	0.000
mpeg2enc	0.293	0.215	0.170	0.083	0.000
rawcaudio	0.605	0.480	0.355	0.134	0.000
rawdaudio	0.743	0.618	0.493	0.243	0.000
Average	0.270	0.191	0.137	0.059	0.000

D. Effect of the Hybrid Scheme

Tables 6-8 list the RVF values of the hybrid scheme for a

Table 6. The register vulnerability factor values of the hybrid scheme with 0, 2, 4, 8 and 16 registers protected by ECC. There are 64 registers.

Benchmarks	0 ECC	2 ECCs	4 ECCs	8 ECCs	16 ECCs
cordic	0.026	0.004	0.002	0.002	0.001
cjpeg	0.127	0.114	0.106	0.091	0.067
g721decode	0.068	0.045	0.030	0.011	0.000
des	0.023	0.011	0.010	0.008	0.006
gsm decode	0.060	0.040	0.025	0.006	0.000
gsm encode	0.064	0.044	0.026	0.006	0.000
djpeg	0.034	0.021	0.019	0.016	0.012
mpeg2enc	0.289	0.269	0.251	0.215	0.144
rawcaudio	0.205	0.174	0.142	0.086	0.028
rawdaudio	0.023	0.011	0.008	0.005	0.002
Average	0.092	0.073	0.062	0.045	0.026

Table 7. The register vulnerability factor values of the hybrid scheme with 0, 2, 4, 8, and 16 registers protected by ECC. There are 32registers.

Benchmarks	0 ECC	2 ECCs	4 ECCs	8 ECCs	16 ECCs
cordic	0.024	0.003	0.002	0.001	0.000
cjpeg	0.086	0.060	0.051	0.037	0.014
g721decode	0.137	0.093	0.063	0.024	0.000
des	0.034	0.010	0.008	0.006	0.002
gsm decode	0.121	0.079	0.049	0.013	0.000
gsm encode	0.127	0.088	0.052	0.012	0.000
djpeg	0.043	0.018	0.016	0.012	0.006
mpeg2enc	0.289	0.247	0.211	0.153	0.078
rawcaudio	0.097	0.066	0.047	0.019	0.003
rawdaudio	0.028	0.004	0.002	0.001	0.001
Average	0.099	0.067	0.050	0.028	0.010

Table 8. The register vulnerability factor values of the hybrid scheme with 0, 2, 4, 8, and 16 registers protected by ECC. There are 16 registers.

Benchmarks	0 ECC	2 ECCs	4 ECCs	8 ECCs	16 ECCs
cordic	0.042	0.003	0.002	0.001	0.000
cjpeg	0.096	0.045	0.034	0.016	0.000
g721decode	0.249	0.157	0.100	0.030	0.000
des	0.059	0.009	0.007	0.002	0.000
gsm decode	0.243	0.164	0.106	0.021	0.000
gsm encode	0.235	0.147	0.092	0.020	0.000
djpeg	0.066	0.015	0.011	0.006	0.000
mpeg2enc	0.211	0.160	0.121	0.063	0.000
rawcaudio	0.057	0.008	0.004	0.001	0.000
rawdaudio	0.048	0.002	0.001	0.000	0.000
Average	0.131	0.071	0.048	0.016	0.000

register file with 64, 32, or 16 registers, respectively. The hybrid scheme is more effective at reducing the RVF for different benchmarks than either the re-scheduling and register re-assignment approaches alone. The RVF value is as low as 6.1%, on average, with only four out of 64 registers protected by ECC. Protecting four registers with ECC will reduce the RVF to 5.0% and 4.7% for a register file with 32 or 16 registers, respectively, indicating great improvement in register file immunity against soft errors.

VI. RELATED WORK

Transient errors caused by external particle strikes have traditionally been a concern for systems that operate in highly noisy environments. They have increasingly become a challenge for microprocessors ranging from high-end servers to embedded processors used for reliability-critical applications with the scaling of technology. Kim and Somani [20] conducted fault injection experiments on picoJava-II in its RTL model to understand the micro- processor vulnerability to soft errors. They found large variations for different hardware blocks. Wang et al. [21] studied the soft error sensitivity of a modern microprocessor, similar to the Alpha 21264, through fault injection on a RTL model. They reported that less than 15% of single bit errors in the processor state result in software visible errors. Mukherjee et al. [1] proposed an approach to measure AVF based on a performance model. They reported the AVFs of the instruction queue and execution units are 28% and 9%, respectively, for an Itanium2-like IA64 processor.

Biswas et al. [22] extended the lifetime analysis technique to examine the architectural vulnerability factors for address-based structures. All these prior research efforts have motivated us to study the sensitivity of register files to transient errors more accurately. In contrast to previous work, this paper focuses on studying the reliability of register files, which are not address-based but can significantly affect overall system reliability if unprotected. We develop a method to compute the probability of register soft error propagation accurately by exploiting the fact that register soft errors can be overlapped by the register write operations, which are not captured by previous models. In this paper, we also proposed several novel techniques to improve register file reliability without significant hardware cost or performance degradation.

A number of techniques in the literature improve hardware reliability against transient errors. However, most of the research effort focuses on protecting main memory [8, 9], cache [8, 9, 23], datapath [4, 12, 13, 24], or multicore [25]. Currently, parity and ECC are the most widely-used mechanisms to protect the storage units, but come at the cost of area, energy and design time. Particularly, if the ECC computation is on the critical path, it may affect performance. Rajaram et al. analyzed the soft error rate for a variety of flip-flops with regard to register reliability against transient errors [7].

Memik et al. [14] proposed a scheme to replicate register values in the physical registers to increase 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 dynamic register renaming. In contrast, VLIW processors rely on compilers to manage the registers. They typically do not have additional physical registers. Therefore, the approach proposed in [14] cannot be applied to VLIW-like processors that do not have physical registers. This paper, in comparison, proposes two compiler-guided techniques to improve the register file immunity to soft errors that can be widely applied to a variety of processors. Lee and Shrivastava [26] proposed a compiler-microarchitecture hybrid approach to enhance the energy efficiency of soft error protection for register files. In contrast, this paper focuses on improving the reliability of register files.

VII. CONCLUSION

As technology scales, lower supply voltage, higher density and higher frequency will make microprocessors more vulnerable to transient errors. The first step is to understand the vulnerability of different hardware components to soft errors accurately to enhance processor reliability cost-effectively. This 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 [8, 9], cache [10, 11] or datapath [4, 12, 13], 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 RVF to characterize the probability that register soft errors can be propagated to other system components and thus affect the final output. We also propose an approach to compute RVF based on register access patterns. Therefore, the reliability of register files can be estimated using both the RVF and raw soft error rate of latches.

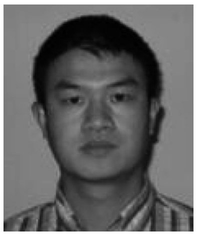
We develop two compiler-guided techniques built upon the concept of RVF to improve register file reliability by decreasing the RVF value, because a smaller RVF value indicates that the register file is less susceptible to soft errors and thus is more reliable. The first technique is a pure software-based approach that exploits the scheduling slack to move the register write operations, as late as possible, and the register read operations, as early as possible, without increasing critical path latency.

Our experiments demonstrate that the instruction re-scheduling based on hyperblocks [16] 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 improves 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 techniques to further reduce RVF. Experiments show the hybrid scheme can reduce average RVF to 6.1% with only four out of 64 registers covered by ECC. Thus, register file reliability is improved substantially without significantly influencing cost. Moreover, all the techniques proposed in this paper can enhance register file immunity to soft errors without compromising performance.

REFERENCES

1. S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 29-40.
2. S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: an architectural perspective," *The 11th International Symposium on High-Performance Computer Architecture*, San Francisco, CA, 2005, pp. 243-247.
3. M. Rebaudengo, M. S. Reorda, and M. Violante, "An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor," *Design, Automation and Test in Europe Conference and Exhibition*, Munich, Germany, 2003, pp. 602-607.
4. S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," *Proceedings of the 27th International Symposium on Computer Architecture*, Vancouver, BC, 2000, pp. 25-36.
5. M. Tremblay and Y. Tamir, "Support for fault tolerance in VLSI processors," *IEEE International Symposium on Circuits and Systems*, Portland, OR, 1989, pp. 388-392.
6. R. Phelan, Addressing Soft Errors in ARM Core-Based SoC, Cambridge, UK: ARM Ltd., 2003.
7. R. Ramanarayanan, V. Degalahal, N. Vijaykrishnan, M. J. Irwin, and D. Duarte, "Analysis of soft error rate in flip-flops and scannable latches," *Proceedings of the IEEE International Systems-on-Chip (SOC) Conference*, Portland, OR, 2003, pp. 231-234.
8. C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state of the art review," *Reliable Computer Systems: Design and Evaluation*, 2nd ed., Burlington, MA: Digital Press, 1992, pp. 771-786.
9. T. J. Dell, A White Paper on the Benefits of Chipkill-Correct ECC for PC Serve Main Memory. Watson, NY: IBM Microelectronics Division, 1997.
10. S. Kim and A. K. Somani, "Area efficient architectures for information integrity in cache memories," *Proceedings of the 26th Annual International Symposium on Computer Architecture*, Atlanta, GA, 1999, pp. 246-255.
11. C. H. Chen and A. K. Somani, "Fault-containment in cache memories for TMR redundant processor systems," *IEEE Transactions on Computers*, vol. 48, no. 4, pp. 386-397, Apr. 1999.
12. T. M. Austin, "DIVA: a reliable substrate for deep submicron microarchitecture design," *Proceedings of the 32nd Annual International Symposium on Microarchitecture*, Haifa , Israel, 1999, pp. 196-207.
13. J. Ray, J. C. Hoe, and B. Falsafi, "Dual use of superscalar datapath for transient-fault detection and recovery," *Proceedings of the 34th ACM/IEEE International Symposium on Microarchitecture*, Austin, TX, 2001, pp. 214-224.
14. G. Memik, M. T. Kandemir, and O. Ozturk, "Increasing register file immunity to transient errors," *Proceedings of the Design, Automation and Test in Europe*, Munich, Germany, 2005, pp. 586-591.
15. W. M. 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," *Journal of Supercomputing*, vol. 7, no. 1-2, pp. 229-248, May. 1993.
16. S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, and R. A. Bringmann, "Effective compiler support for predicated execution using the hyperblock," *Proceedings of the 25th Annual International Symposium on Microarchitecture*, Portland, OR, 1992, pp. 45-54.
17. "Trimaran: an infrastructure for research in backend compilation and architecture exploration," http://www.trimaran.org.
18. C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture*, Research Triangle Park, NC, 1997, pp. 330-335.
19. S. S. Muchnick, Advanced Compiler Design and Implementation. San Francisco, CA: Morgan Kaufmann Publishers, 1997.
20. S. W. Kim and A. K. Somani, "Soft error sensitivity characterization for microprocessor dependability enhancement strategy," *Proceedings. International Conference on Dependable Systems and Networks*, Washington, DC, 2002, pp. 416-425.
21. N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," *International Conference on Dependable Systems and Networks*, Florence, Italy, 2004, pp. 61-70.
22. A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," *32nd Interntional Symposium on Computer Architecture*, Madison, WI, 2005, pp. 532-543.
23. A. Biswas, C. Recchia, S. S. Mukherjee, V. Ambrose, L. Chan, A. Jaleel, A. E. Papathanasiou, M. Plaster, and N. Seifert, "Explaining cache SER anomaly using relative DUE AVF measurement," *The 16th IEEE International Symposium on High-Performance Computer Architecture*, Bangalore, India, 2010, pp. 1-12.
24. N. K. Soundararajan, A. Parashar, and A. Sivasubramaniam, "Mechanisms for bounding vulnerabilities of processor structures," *The 34th Annual International Symposium on Computer Architecture*, San Diego, CA, 2007, pp. 506-515.
25. N. Soundararajan, A. Sivasubramaniam, and V. Narayanan, "Characterizing the soft error vulnerability of multicores running multithreaded applications," *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, New York, NY, 2010, pp. 379-380.
26. J. Lee and A. Shrivastava, "A compiler-microarchitecture hybrid approach to soft error reduction for register files," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 7, pp. 1018-1027, Jul. 2010.



Jun Yan

Jun Yan is currently a research scientist at Mathworks. He received his Ph.D. degree in Electrical and Computer Engineering from Southern Illinois University Carbondale (SIUC) in 2009. He worked in R&D at Lucent Technologies from 2004 to 2005 and at Huawei Technologies from 2002 to 2004, before he came to SIUC. He received his MS from Tianjin University, China, in 2002, and BS from Shenyang Architecture and Civil Engineering Institute, China, in 1998.

Wei Zhang

Wei Zhang is an associate professor in Electrical and Computer Engineering at Virginia Commonwealth University. He received his Ph.D. degree in computer science and engineering from the Pennsylvania State University in 2003. Since then, he has worked as an assistant and an associate professor at Southern Illinois University Carbondale until August 2010. His research interests are in embedded and real-time computing systems, computer architecture and compiler. Dr. Zhang has received the 2009 SIUC Excellence through Commitment Outstanding Scholar Award for the College of Engineering, and 2007 IBM Real-time Innovation Award. His research has been supported by NSF, IBM and Altera. He is a senior member of IEEE. He has served as a member of the technical program committees for several IEEE/ACM conferences and workshops.