UnSync-CMP: Multicore CMP Architecture for Energy-Efficient Soft-Error Reliability

Reiley Jeyapaul, *Student Member*, *IEEE*, Fei Hong, *Student Member*, *IEEE*, Abhishek Rhisheekesan, *Student Member*, *IEEE*, Aviral Shrivastava, *Senior Member*, *IEEE*, and Kyoungwoo Lee, *Member*, *IEEE*

Abstract—Reducing device dimensions, increasing transistor densities, and smaller timing windows, expose the vulnerability of processors to soft errors induced by charge carrying particles. Since these factors are only consequences of the inevitable advancement in processor technology, the industry has been forced to improve reliability on general purpose chip multiprocessors (CMPs). With the availability of increased hardware resources, redundancy-based techniques are the most promising methods to eradicate soft-error failures in CMP systems. In this work, we propose a novel customizable and redundant CMP architecture (UnSync) that utilizes hardware-based detection mechanisms (most of which are readily available in the processor), to reduce overheads during error-free executions. In the presence of errors (which are infrequent), the *always forward execution* enabled recovery mechanism provides for resilience in the system. The inherent nature of our architecture framework supports customization of the redundancy, and thereby provides means to achieve possible performance-reliability tradeoffs in many-core systems. We provide a redundancy-based soft-error resilient CMP architecture for both write-through and write-back cache configurations. We design a detailed RTL model of our UnSync architecture and perform hardware synthesis to compare the hardware (power/area) overheads incurred. We compare the same with those of the Reunion technique, a state-of-the-art redundant multicore architecture. We also perform cycle-accurate simulations over a wide range of SPEC2000, and MiBench benchmarks to evaluate the performance efficiency achieved over that of the Reunion architecture. Experimental results show that, our UnSync architecture reduces power consumption by 34.5 percent and improves performance by up to 20 percent with 13.3 percent less area overhead, when compared to the Reunion architecture for the same level of reliability achieved.

Index Terms—Multicore architecture, soft error, CMP, reliability, power efficiency

1 Introduction

The boons of technology scaling come with several consequences. We can build chips that have billions of transistors, but since we pack many of those transistors tightly together (for reduced die area and yield [1]), the increased power density, diminishing node capacitances, and reduced noise margins make these transistors unreliable. To counter the power density problem, chip designers have resorted to multicore architectures that provide a way to continue improving performance, without a commensurate increase in the power consumption. As a result, chip multiprocessors (CMPs) have become popular, for example, Intel core 2 duo, Intel core i7, AMD Opteron, IBM Cell processor, and so on. However, the reliability problem continues to grow. Transistors are becoming so small and fragile that a stray charge or high-energy particle can cause current pulses on a transistor and toggle the logic value of the gate. This phenomenon of radiation-induced transient error is referred to as "Soft Error." The problem is that while high-energy neutrons (100 keV-1 GeV from cosmic background) have caused soft errors for a long time, now low-energy neutron particles (10 meV-1 eV) also cause soft errors [2]. This effect is multiplied with the fact that there are many more low-energy particles than those at higher energies [3]. Soft errors have already been attributed to cause large fiscal damages, for example, Sun blamed soft errors for the crash of their million-dollar line SUN flagship servers in November 2000 [4]. At the current technology node, a soft error may occur in a high-end server once in every 170 hours, but it is expected to increase exponentially with technology scaling and reach alarming levels of once-per-day [5].

Chip multiprocessors are inherently good for reliability due to the availability of many-cores, on which redundant computations can be performed for error detection and/or correction. Many redundancy-based techniques, at various levels of design space abstraction, based on dual modular redundancy (DMR) [6], triple modular redundancy (TMR) [7], and checkpointing [8] have been proposed to enable error detection and correction in CMPs. Reunion [9] is one promising redundancy-based multicore architecture that achieves error resilience with low-performance overhead. In Reunion, a hash of a set of instructions called *fingerprint* is generated at regular intervals and is compared between redundant cores executing the same thread. The retired instructions are committed to their respective architectural register file (ARF) or memory *iff* the *fingerprints* are found to match, if not, the execution is resumed from the previous correct position. Though efficient in its design to detect and

- *R. Jeyapaul, F. Hong, A. Rhisheekesan, and A. Shrivastava are with the Compiler Microarchitecture Lab, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Byeng 407, 699 South Mill Avenue, Brickyard Building, Tempe, AZ 85281. E-mail: {reiley.jeyapaul, fei.hong, arhishee, aviral.shrivastava}@asu.edu.*
- *K. Lee is with the Dependable Computing Lab, Yonsei University, C518b, 50 Yonsei-ro, Seodaemun-gu, Seoul 120-749, Korea. E-mail: kyoungwoo.lee@yonsei.ac.kr.*

Manuscript received 9 Apr. 2012; revised 31 Dec. 2012; accepted 31 Dec. 2012; published online 11 Jan. 2013.
Recommended for acceptance by A. Nayak.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2012-04-0364.
Digital Object Identifier no. 10.1109/TPDS.2013.14.

recover from errors, the Reunion methodology suffers from issues during implementation:

1. *Significant changes to the core design.* Its implementation requires adding a new pipeline stage in the processor, resulting in high overheads of power and area.
2. *Performance overheads due to serializing instructions.* Blocking instructions like traps, memory barriers, and nonidempotent instructions require the redundant cores to synchronize causing performance degradation.
3. *Performance overheads due to architecture limitations.* Increased reorder buffer (ROB) occupancy during the synchronization process (between the redundant cores), as the executed instructions cannot be committed before being validated, leads to significant performance degradation.
4. *Ignorance of efficient hardware mechanisms for error detection.* The availability of efficient hardware-based error-detection schemes (e.g., parity bits, DMR, etc.) is overlooked, thus, underusing the hardware resources.

In this paper, we propose a novel redundancy-based multicore architecture for CMPs: *UnSync*. In this, the already existing and readily available power efficient hardware error-detection mechanisms are used to provide effective protection against "infrequent" soft errors, instead of output/data comparison among redundant copies. UnSync consists of two identical cores executing the same thread; each core is modified with power/area efficient hardware-only error-detection mechanisms, for every sequential element in the processor core. Contrary to the popular redundancy-based techniques that involve loose/tight lock-stepping, UnSync disassociates the two redundant cores during error-free execution as much as possible, thereby allowing for maximum possible performance in the absence of errors. In case an error is detected in one of the cores, execution in both the cores is stalled. The architectural state from the error-free core is copied onto the erroneous core to resume correct execution on both the cores, ensuring *always forward execution*, i.e., it does not go back for re-execution. Our recovery mechanism has a higher overhead, compared to the popular redundancy-based techniques for CMP. However, by reducing the performance overheads during error-free execution, and given the fact that errors are infrequent, UnSync achieves better performance at lower power/area overheads and lesser design complexity. In UnSync, since every individual core is identical in its hardware architecture, the number and pairs of redundant cores in the multicore system can be configured by the user, based on reliability and performance requirements. The presence of hardware-based error-detection techniques introduces a hardware latency between detection and recovery, which may interfere with our region of coverage (ROC) premise that the data, that is, written into the memory is always soft-error free. This problem is easily solved by introducing a write-through L1 cache. Though this cache configuration is observed in many high-end systems, a more popular cache configuration is the write-back L1 cache. In this work, we propose an alternate copy-cache methodology to enable UnSync for a write-back cache configuration, thus, providing a more complete soft-error resilient CMP architecture. In Section 3 of the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.14, we present a detailed discussion of Reunion and the UnSync CMP microarchitecture details in the form of a comparative analysis with reference to the overheads in area, power, and performance realized.

To evaluate and compare the effectiveness of UnSync (compared to Reunion), we developed a multicore power, performance, and area estimation setup. We model the multicore architecture and estimate program cycle times using the M5 simulator [10]. To estimate the power and area overhead of the core and pipeline architecture, we implemented changes required for both the *UnSync* and *Reunion* in a five-stage pipelined RTL implementation of the MIPS [11] architecture. We synthesize and also place and route (PNR) the RTL models, using the Cadence encounter [12], to evaluate the on-chip hardware overheads. The L1 cache area and power are estimated using CACTI [13] models. Our experimental results show that the UnSync architecture achieves up to 20 percent improved performance with 14.6 percent reduced area and 34.5 percent lower power overhead as compared to the Reunion architecture.

The submitted paper is an extension of conference paper [14]. The list of contributions in this paper submission, over and above that in the conference paper, is provided in the "Differences-document" submitted herewith. Extended details of the proposed work is provided in the online supplemental material.

2 Related Work

Error-resilient redundant processor designs must solve two key problems: maintaining identical instruction streams and detecting divergent executions, on the redundant cores. Mainframes, which have provided fault tolerance for decades, solve these problems by tightly lock-stepping two executions [15]. Lock-step ensures that both processors observe identical load values, cache invalidations, and external interrupts. While conceptually simple, lock-step becomes an increasing burden as device scaling continues [16]. As technology scaling continues to concern the performance and power consumption overheads, multicore designs are being investigated to keep up with Moore's law [17]. The increase in the integration of a number of processor cores on a single chip makes the chip more dense in area and, hence, makes them more vulnerable to reliability threats such as soft errors. On the other hand, CMPs inherently provide replicated hardware resources that can be exploited for error detection and recovery. A number of proposals [9], [18], [19], [20] have attempted to take advantage of the inherent replication of cores in CMPs to provide fault tolerance by pairing cores and checking their execution results.

Redundant multithreading (RMT) [21] is a modified and efficient simultaneous multithreading (SMT) processor architecture, where only store addresses and values are checked to detect soft errors. It uses a load-value queue

(LVQ) to provide consistent replication, on redundant threads, of load values. Output comparison is performed by a store comparator (SC) that buffers completed but not yet committed stores. Once a store has been successfully verified, it is eligible to commit in each thread, and a single instance of the store is released to the memory hierarchy. The RMT technique was later extended to map redundant threads onto separate processor cores in a CMP, rather than separate hardware threads in an SMT. Chip-level redundant threading (CRT) [22] solved one source of resource contention while exacerbating another. Execution on separate cores eliminated contention by providing each thread its own private set of resources. However, CMP cores are not as tightly integrated as SMT threads, and the additional physical separation increases the round-trip store verification latency. This subsequently increases the average store execution time, which reduces any gains reaped from additional hardware resources, and results in a net slowdown. Fingerprinting [19] is a checkpointing scheme designed to minimize hardware changes to commodity hardware. Processor pairs identify errors by comparing cryptographic signatures that summarize architecture state updates. Mismatches trigger a rollback to a known good checkpoint; successful comparisons free prior checkpoints. Such techniques can be implemented cheaply; however, they rely on heavy-weight checkpointing mechanisms that capture all of system states (including memory) and increase error-detection latency. In this paper, we adapt the benefits achieved from core-level redundancy and propose a method to reduce the hardware overheads (memory storage, comparators, etc.), and also reduce intercore communication (load values, fingerprint, etc.) to ensure resilience in the system. To increase the resource usage and flexibility of CMPs, Gupta et al. [23] developed a redundancy-based technique at a finer granularity. In this, the pipeline stages are connected through routers and resource sharing is enabled across cores. However, the larger hardware overheads and design complexity involved limit their applicability when the number of cores in the system increases beyond hundred. In the work by Gomaa et al. [20], the comparison of load values is restricted to be based on the data-dependence chains in the executing threads, to reduce on performance overheads in comparison. In dynamic dual modular redundancy (DDMR) [24], the redundant processing on linked cores is performed dynamically. In these, dynamic linking achieves two main benefits: 1) reliability is unaffected by defective cores; and 2) cores with similar throughput may be paired for the purpose of running redundant threads at similar speeds. Although having the benefit of flexibility and better resource usage, these techniques suffer from increased design complexity and high-performance overhead.

Smolens et al. [9] in their work overcome the issues in design complexity and overheads in the load-value queue technique, and propose to relax strict input replication by allowing the redundant thread to issue loads directly to the memory system. However, to deal with input incoherence resulting from multiprocessor data races, it identifies cases where redundant loads receive updated values from other processors in the system. Such mismatches are essentially treated as transient errors—both threads reissue their respective load and recheck the load values. The mismatch in load values is handled by issuing the load for the third time via synchronizing memory requests that eliminate input incoherence for the requested cache line. Reunion [9] is one of the state-of-the-art CMP-based redundant techniques. To synchronize between redundant core pairs and reduce bandwidth of comparison, Reunion proposes a comparison of *fingerprints* between vocal and mute cores. However, our intensive analysis and experiments show that Reunion incurs high overheads in terms of area, performance, and power. In this paper, we highlight in detail the design issues and hardware overheads involved in the implementation of Reunion on a many-core processor setup (in Section 3 of the online supplemental material).

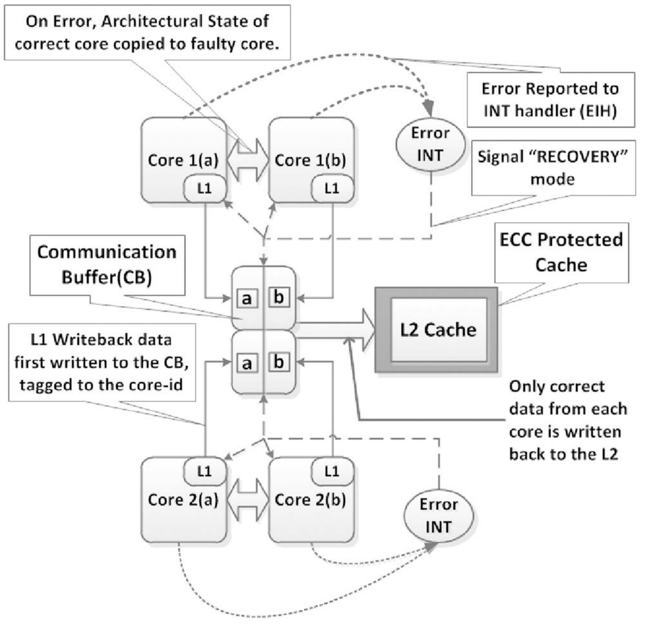

On Error, Architectural State of correct core copied to faulty core.
Error Reported to INT handler (EIH)
Error INT
Signal "RECOVERY" mode
Core 1(a)
Core 1(b)
L1
L1
Communication Buffer(CB)
ECC Protected Cache
a
b
L2 Cache
L1 Writeback data first written to the CB, tagged to the core-id
a
b
Only correct data from each core is written back to the L2
L1
L1
Core 2(a)
Core 2(b)
Error INT

Fig. 1. UnSync Architecture: The L1 of each redundant core writes into a CB, which acts as a secondary write buffer. Only one of the redundant copies, from the CB-pair (a, b), is written into the L2 cache. An error detected in any of the cores signals "RECOVERY" through the EIH, to the corresponding core-pair and the CB.

Error-free execution, minimal error-detection overhead (hardware or software), and possible customization of redundancy are the three ideal expectations of a reliable many-core system. Our proposal, UnSync, satisfies these three ideal expectations: 1) UnSync reduces error-detection overheads in terms of area, performance, and power by exploiting readily available hardware-based error-detection mechanisms, especially during error-free executions. 2) UnSync further has simple and easy provisions to implement customizable redundancy and realize performance-reliability tradeoffs in scalable many-core systems.

3 The UnSync CMP Architecture

3.1 Overview

Fig. 1 describes an overview of the UnSync architecture. It shows two core-pairs of our two-way redundant UnSync architecture with their intercore and intracore communication links. Each core in this architecture is configured with an on-core write-through L1 cache and off-core shared ECC protected L2 cache. As part of the hardware-based error-detection machinery, the L1 cache contains a parity-bit on each cache line to detect 1-bit errors. Similarly, the core architecture blocks are fitted with error-detection circuitry, details of which will be discussed in detail later in this section. The hardware error-detection blocks are connected to an *error interrupt handler* (EIH) for each core-pair, to signal recovery in case an error is detected. Data committed into the L1 cache, from each core of a core-pair executing identical threads of the program, is first written into a *communication buffer* (CB). From here, one copy of the data is passed on, to be written back in the protected L2 cache. Please refer to Section 1 of the online supplemental material for a detailed description of the customizabile UnSync-CMP architecture construction.

3.2 Working

The working of the UnSync architecture can be best studied by observing its phases of operation. Please refer to Section 2 of the online supplemental material for a detailed explanation of the working (with examples) of the UnSync-CMP architecture in each of the working modes described here.

3.2.1 Error-Free Mode

The two identical cores in a core-pad of the application, where each core performs memory accesses on the shared L2 cache as independent cores. Data written on the L1-cache of a core, as it leaves the core (as in a write-through cache), is written on a noncoalescing CB, one for each core in the core-pair, as described in Fig. 1. In the CB, each updated entry is tagged with its corresponding instruction address. As and when the L1-L2 data bus is free (available for data transfer), the latest entry that has completed execution on both the CB is selected, and one copy of all the CB entries, earlier to this, is written into the L2 cache. This process ensures that, when processed data leaves the cores to be updated into the lower-level memory, both the cores have completed a particular state in the execution, and since no error was detected during this time, the two copies are correct.

3.2.2 Error Detection

Error detection in each core of the UnSync architecture is enforced by the use of hardware-only error-detection blocks. The L1 cache, register file, and the queuing structures are enabled with 1-bit parity-based detection owing to the fact that data write (parity generation) and read (parity verification) have a minimum of one-cycle time difference. On the other hand, for the architecture blocks like the program counter and pipeline registers, where data is read/written on every cycle, parity-based detection cannot be employed; therefore, dual mode redundancy (DMR)-based error detection is enabled. If any of these detection blocks determine an error in the data, on either core, an interrupt is transmitted to the EIH for that core-pair, which then performs error recovery. The interconnect between the core and the EIH is described by the dotted arrows in Fig. 1.

3.2.3 Recovery Mode

Once the EIH receives an error interrupt, it signals "RECOVERY" to both the cores and CB of the corresponding core-pair (the mechanism and details of the recovery process are described with examples in Section 2 of the supplemental material). In this mode, the following procedure implements our "*always forward execution*" recovery mechanism:

1. Program execution on both the cores of the core-pair is stopped.
2. The pipeline of the erroneous core is flushed, so as to reset the pipeline registers.
3. The architectural state (register file, PC, etc.) and the content of the L1 cache of the error-free core are copied onto that of the erroneous core (the core in which error was detected). This operation is performed by specific subroutines using the shared L2 cache.
4. Data transfer from the CB to the L2 cache is stopped. Only the transfers currently in flight are completed.
5. The content of the CB, corresponding to the erroneous core, is overwritten by data from the error-free core.
6. Once the architectural state, program counter, L1 cache contents, and the CB content of both the erroneous core have been overwritten, both the cores resume execution of the program from the same program counter state as that was copied from the error-free core.

3.3 Features

Here we discuss some of the important features of UnSync that ensure its novel efficient error resilience.

3.3.1 Power-Efficient Error Detection in Hardware

The power and chip-area benefits of the UnSync architecture hinge on the use of hardware-based error-detection mechanisms in each of the cores, in a redundant multicore setup. Single error correction and double error detection (SECDED) is a cache detection and correction mechanism which incurs an overhead of 22 percent in cache area, owing to tree of XOR gates, the depth of which increases super-linear to the number of bits covered by the ECC [25]. In addition, the ECC generation and verification logic requires more than one cycle to complete, which also has an impact on the cycle time of the processor. On the other hand, 1-bit parity-based error-detection mechanism requires only a negligible ($<1\%$) power and area overhead [25]. In addition, the series of XOR gates do not impose a significant delay in its processing, and therefore, can complete its operation in one cycle. It should be noted that, though the probability of an energy particle strike is uniform throughout the processor core, sequential elements which store data (even if it is for one cycle) are the most vulnerable architectural blocks [26]. The TMR-based detection and correction techniques for sequential elements incur an overhead of 200 percent in power, while the DMR-based detection-only technique requires only around 6 percent power overheads [24], [27].

Having identified that error detection on sequential elements and storage elements can ensure error resilience in processors, an efficient choice of the right detection mechanism has to be made for each architecture block. In UnSync, we choose either parity-bit or DMR-based error-detection methods. Owing to the time latency requirements of the parity-bit generation and verification, data storage elements like load store queue (LSQ), translation lookaside buffer (TLB), register file, and the L1 cache data are enabled with parity-bit error detection. All the other sequential elements (e.g., pipeline register, PC) which have accesses on every cycle of processor execution; the one-cycle latency of parity-bit technique is unacceptable, and therefore DMR-based error detection is employed. The use of DMR is reduced as much as possible, because of the hardware area and power overheads involved in every access.

Since only error-detection methods are employed within the core, to ensure error resilience, the redundancy of CMPs is used. By introducing a novel hardware-only error reporting scheme, through the use of the EIH network in the UnSync architecture, area/power-efficient error resilience is achieved.

3.3.2 Need for Synchronization among the Cores Is Eliminated

In popular redundancy-based techniques for error resilience, the execution on the cores (or within a core) is synchronized either by lock-stepping [15] or through memory accesses [21], because: 1) error detection is implemented by comparing the execution outputs or by comparing the memory accessed among the redundant threads (on the same or different cores); and 2) when an error is detected, both the cores can be directed to re-execute a set of instructions from a previously identified error-free position (e.g., checkpoint [19]). However, as suggested by the name of our architecture, *UnSync*, the need to synchronize execution among the redundant cores is eliminated. This is made possible by 1) hardware-based error-detection mechanisms that eliminate the need to compare redundant executions, and 2) the "*always forward execution*"-based recovery mechanism ensures that the cores resume from the last executed position of the correct core and no re-execution is required in either cores.

When an error is detected on a core, execution on both the cores is stopped and the architectural state from the error-free core is copied onto the erroneous core. While resuming the processor, the execution sequence is altered on only the erroneous core (since PC is copied from the error-free core). The error-free core resumes from where it was stopped. Another aspect of this technique is that the amount of instructions retraced (if any) by the erroneous core depends on the difference in the execution speeds between the two cores. In the case when the erroneous core was executing at a slower speed, during recovery, execution on this core is forwarded. The absence of re-executions in the recovery mechanism provides some compensation to the overhead involved in the architectural state and L1 cache content copy from one core to the other.

3.3.3 Customizable Resilience and Redundancy

As the number of cores on a many-core architecture is scaled, the availability of cores and the requirement for soft-error tolerance grow simultaneously. The Cisco Metro chip with 188 cores and 4 redundant cores per die poses as an

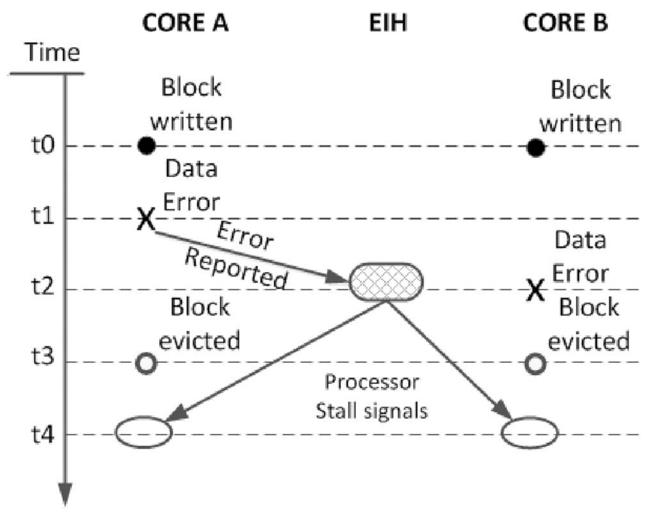


Fig. 2. Once an error is detected in a core, non-zero cycles are incurred in the communication to the EIH and subsequent RECOVERY signaling to stall the processor and perform cache copy. In write-back cache configuration, corrupt data (if any) in core B could be written into the memory or copied over to core A during recovery.

ideal example of defect tolerance in a many-core architecture with core-level redundancy. The overall system throughput is thus given by the maximum number of nonredundant tasks that can be executed (36 in the case of the Cisco Metro chip) on the processor. However, if the level of redundant cores executing a thread on the processor can be customized dynamically, it is possible to increase or decrease the system throughput in accordance with the reliability requirements and the current soft-error rate (SER). With the rapid increase in the number of cores on even general purpose processors, the need for user-configurable means to realize performance-reliability trade-offs becomes evident. In UnSync, no synchronization or data comparison exists between the two redundant cores; only an instruction completion check is performed at the CB, to write a single copy of data from the CB to the L2, when the data bus is available. Since there does not exist any hardwired synchronization between the two cores, the CB and EIH are the only architecture blocks creating a virtual paring of the two adjacent cores. This architecture, thus, provides an easy means to configure the CB and EIH to decouple the two adjacent cores, thereby, allowing the two cores to execute independent threads without error resilience on each. As a result, UnSync can be easily customized for non-redundant execution (statically or dynamically), and therefore, a tradeoff between reliability and performance can be achieved.

3.4 UnSync-CMP: Implementation Details

3.4.1 Need for Write-through L1 Cache Configuration

Most architectures used in high-reliability applications use the L1 cache in write-through mode. This is because in the write-through configuration, a copy of the updated cache data always exists in the lower-level memory. Therefore, if an error is detected during read operation on a cache line, the cache line can be invalidated, and the correct updated cache line can be loaded from the memory. Here, we discuss the need for the write-through cache configuration and its

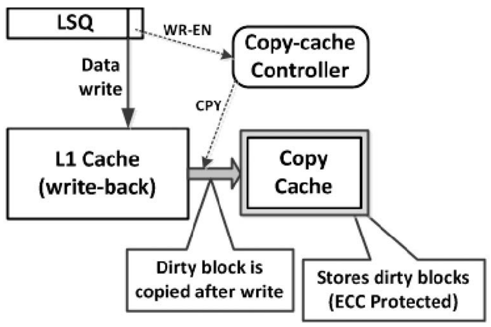


Fig. 3. Copy-cache methodology for the write-back L1 cache (on UnSync), to overcome the problem of soft errors on dirty cache-blocks. Write operation on the L1 cache is explained through an example.

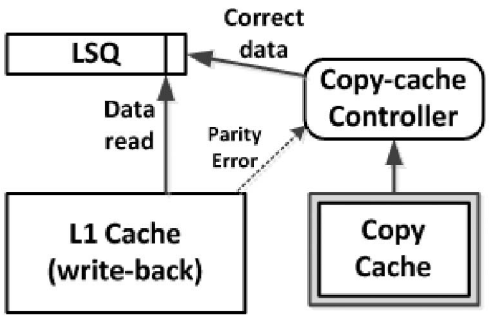


Fig. 4. Copy-cache methodology for the write-back L1 cache (on UnSync), to overcome the problem of soft errors on dirty cache-blocks. Read operation on the L1 cache is explained.

importance in the UnSync architecture. If the UnSync core was configured using a write-back cache, an error on a cache line may leave us in an irrecoverable state. Fig. 2 describes such a scenario, and thereby, demonstrates the importance of the write-through cache in UnSync. Fig. 2, shows two redundant cores with write-back caches, running an identical thread, and the events in the caches. At time $t0$, a data cache block is written (on both cores), and therefore, the cache line is deemed dirty. At time $t1$, an error is detected in core A and then the error is reported to EIH. The EIH, on receipt of the error signal, transmits processor stall signals (RECOVERY) to both the cores, which requires a non-zero number of cycles for both the processor pipelines to stall. Based on information received by the EIH, core A is known to be erroneous and has to be replaced with contents from core B. Meanwhile, an error may strike on a dirty cache block in core B, which would be detected only when read during the recovery process. Since the cache is a write-back cache, and the erroneous cache line in core B is dirty, it does not have an updated copy anywhere else in the system. This scenario, thus, makes it impossible to recover from errors on the cores. On the other hand, if the cache configuration was a write-through cache, as and when the block was updated, a copy is written into the L2 cache (or memory). In a similar situation as in Fig. 2, it is possible to simply invalidate both the cache lines and treat the same as cache-misses, as the correct updated copy of the cache blocks are available in the ECC protected L2 cache.

3.4.2 Copy-Cache for the Write-Back L1 Cache

The write-through cache configuration will enable the power-efficient working of our UnSync CMP architecture. However, the use of a write-through cache has an impact on the overall power consumption of the system owing to the increased number of memory writes and cache-memory bus occupancy. Therefore, the write-back cache is a more popular configuration for the L1 data cache in high-end systems. To accommodate this configuration, we propose the use of a "copy-cache" to overcome the detection-to-recovery latency in UnSync. The copy-cache architecture consists of another L1 cache, which is designated to contain a copy of the dirty cache blocks in the main L1 cache, copied into it whenever a data block is written. When read from the cache, a parity error on a dirty cache block (soft error in the L1 cache) is signaled to the copy-cache controller, which then reads the saved copy of the dirty block and redirects the same to the load request.

Our proposed copy-cache architecture for the L1 cache configuration is described through figures. The working of the copy-cache methodology in the two modes of cache operation can be described as follows:

WRITE mode (see Fig. 3). When a data block is written into the L1 cache. On every write, a cache-block is updated in the cache, which renders it `dirty`. In parallel to the data-write operation, a copy of this cache-block is written into the ECC protected copy-cache. The processor is abstracted from the presence of the copy-cache during error-free execution, and therefore, does not incur any additional performance overheads due to its existence. ECC protection on the copy-cache ensures that the data stored here is always correct and safe from soft errors.

READ mode (see Fig. 4). When a dirty block of data is being read from the L1 cache. During error-free execution, the data from the L1 cache is read with the same read-latency, and thus, does not impact the performance of the system. The parity error detection is triggered on a soft error in any of the L1 cache blocks. If the cache-block is `clean`, the error recovery is simple; the correct data is loaded from the memory assuming a cache-miss for the block in the cache. If the cache-block is `dirty`, the copy-cache controller is triggered. This block loads the correct data from the copy-cache and redirects the same to serve the corresponding load request from the LSQ. This scheme incurs a performance penalty of two read accesses, which is comparable to the cache-miss latency of a write-through cache, which indicates no significant performance difference between the UnSync implementations on a write-through or a write-back cache configuration.

4 Experimental Setup

To evaluate and compare the effectiveness of UnSync, in comparison with Reunion, we develop a multicore setup to estimate power, performance, and area. The cycle-accurate M5 multicore simulator [10] is modified to model the UnSync and Reunion implementation on a 4 core processor. The specifications of each core is tabulated in Table 1. The simulator is instrumented to obtain application statistics of cycle-time and the cycle-delays of each architecture block. We model accurately 1) the faster core's delay due to stalls during the execution of serializing instructions and increased ROB occupancy in the Reunion architecture, and 2) the stalls

TABLE 1
Simulated Baseline CMP Parameters

Parameter	Configuration
Processor Cores	8 logical cores, Alpha 21264 2GHz, 5-stage pipeline; out-of-order 4-wide fetch/issue/commit
Issue Queue	64
L1 Cache	32KB split I/D, 2-way, 10 MSHRs 2 cycle access latency, 64-byte/line
Shared L2 Cache	4MB, 8-way, 64-byte/line 20-cycle access latency, 20 MSHRs
I-TLB	48 entries, 2-way
D-TLB	64 entries, 2-way
Memory	3GB, 64-bit wide, 400 cycles access latency

caused when the CB is full and the bus is busy. We experiment over benchmarks from SPEC2000 and MiBench.

To estimate the power and area overheads incurred, we perform hardware synthesis (using the Cadence encounter [12]) on an RTL implementation of the MIPS [11] processor. We implement both the UnSync and Reunion architecture on the baseline MIPS core. The hardware implementation details are as in Table 2. In our analysis, since each core is identical and executes redundantly, we compare and contrast the area and power of only a single core in three configurations. The hardware components added to the MIPS core for the Reunion implementation are *fingerprint* $\text{size} = 16$ bits, $fingerprint\ interval = 10$ instructions, and the CHECK $\text{Stage Buffer} = 17$ entries each of 66 bits. In the case of UnSync, the L1 cache is in write-through configuration with 10 entries in the Communication Buffer, for each core. We synthesize three core models for the same frequency of 300 MHz at 65-nm technology. To accurately evaluate the impact of data-paths and interconnects in the processor implementations, we perform place-and-route (PNR) at the nominal density of 0.49. For an accurate analysis of cache area and power, we estimate the parameters using the CACTI cache simulator [13]. We scale cache line size (and total cache-size accordingly) to derive power and area values for the cache including parity-bit and SECDED [28] error protection mechanisms. For our experiments we implement the L1 cache in both the write-through mode and the write-back mode of operations, where the write-back mode is accompanied with the copy-cache hardware architectures. Our simulations indicate no performance impact due to an equally big L1 cache for the copy cache in the write-back mode. Therefore, our simulation results are targeted to compare the performance of the system notwithstanding the cache configuration.

TABLE 2
Hardware Implementation Details

Reunion	
Fingerprint size	16 bit
Fingerprint interval	10 cycles
Check stage buffer	17 entries (1 entry = 66 bit)
UnSync	
L1 Cache	WT/WB modes
Write buffer	64 entries
Communication buffer	20 entries

TABLE 3
Hardware Overhead Comparison

Parameter	Basic MIPS	Reunion	UnSync
Chip-Area Overhead			
Core (μm^2)	98558	144005	115945
L1 Cache (mm^2)	0.1934	0.2086	0.1939
CB (mm^2)	N/A	N/A	0.00387
Total Area (μm^2)	291958	352605	313715
Overhead (%)	N/A	20.77	7.45
Power Overhead			
Core (W)	1.153	2.038	1.635
L1 Cache (mW)	38.35	42.15	38.45
CB (mW)	N/A	N/A	0.77258
Total Power (W)	1.19	2.08	1.67
Overhead (%)	N/A	74.79	40.34

5 Experimental Evaluation

In this section, we discuss the comparative analysis between UnSync and Reunion through performance simulation experiments. Specific experiments that highlight the overheads in Reunion not seen in UnSync are discussed in Section 4 of the online supplemental material. The customizable functionality of the UnSync CMP architecture is discussed in Section 5 of the online supplemental material.

5.1 Lower Area and Power Overheads in UnSync

5.1.1 *Error Detection Mechanisms Incur Lower Overheads*

Here, we summarize the area and power overheads of the two redundant processor architecture implementations (UnSync and Reunion), in comparison with the baseline MIPS core. The hardware parameters discussed here are that of a single core after PNR, and therefore, demonstrate with greater accuracy the actual power and area specifications of the chip, in each configuration, after fabrication.

In Table 3, we observe that the UnSync architecture core has a hardware area overhead of only 7.45 percent (compared to the MIPS core) and occupies 13.32 percent lesser chip-area when compared to the Reunion implementation. Two significant architectural blocks constitute the area overhead of the Reunion: 1) *the additional CHECK stage* (46 percent in core area), which is composed of *Fingerprint Generator*, *CHECK Stage Buffer*, and the data-path in the register forwarding logic, and *the SECDED protected L1 cache* (7.85 percent in cache area), which is composed of additional storage bits (8 *check* bits for every 64 bit data chunk), and ECC generation and verification circuitry, when compared to the MIPS baseline processor. On the other hand, the UnSync implementation is composed of only 17.6 percent increased core-area and 0.2 percent increased cache area (1 parity bit for a 256 bit cache line). In UnSync, the hardware detection blocks added to all the sequential elements in the core are mostly composed of combinational logic, which can be synthesized optimally for tighter chip-area by the default configurations of the design compiler. On the other hand, the hardware components that contribute to the Reunion area overhead are composed of storage elements, which are mostly regular array structures, that demonstrate lesser flexibility in circuit optimization, for area and power. It should be noted here that the only additional storage block added to the UnSync

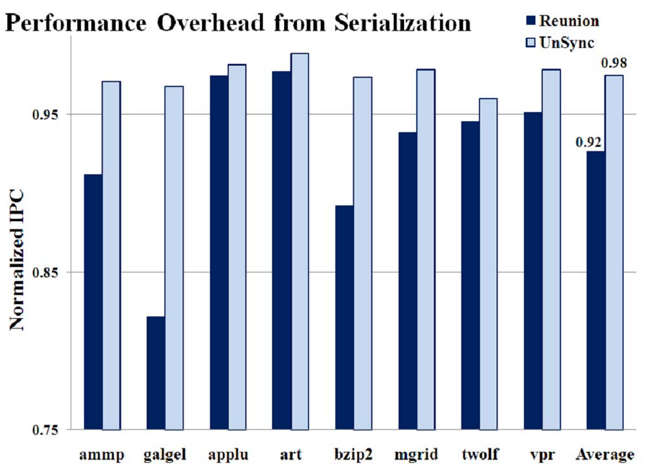


Fig. 5. Reunion is affected by serializing instructions, while UnSync is not. The set of smaller bars on the left demonstrate performance overheads incurred.

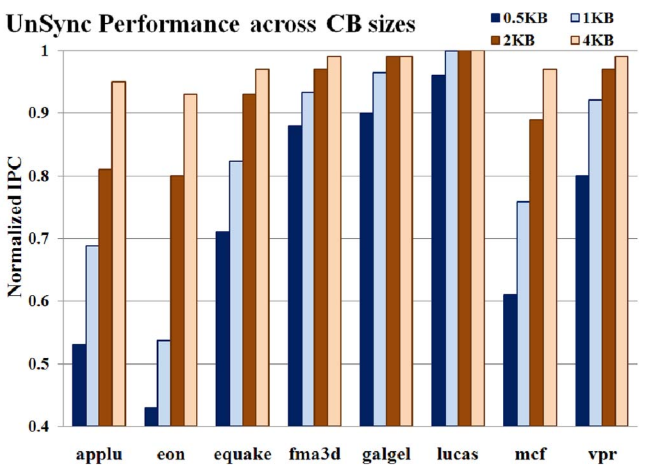


Fig. 6. Larger CB size eliminates the resource occupancy bottleneck and demonstrates better performance.

architecture is the CB, which has a negligible area overhead. In our experiments, we assume an equally big L1 cache with ECC protection, designated as the copy-cache, when the UnSync architecture is configured with a write-back cache. Our Synthesis results indicate corresponding power and area overheads, and negligible performance impact due to its inclusion.

On similar lines, we observe in Table 3 that the power consumption of the Reunion implementation is a staggering 75 percent more than the baseline MIPS core. This is mostly due to accesses to the power-consuming CHECK stage components (hashing logic and buffer array structure), which consumes 76.8 percent more core power compared to the MIPS core. In addition, the SECDED generation and verification on every cache access constitutes around 10 percent more cache power consumption than the baseline MIPS cache. On the other hand, the hardware detection blocks added to the processor core only cause around 42 percent increased power consumption, while the parity bit protection technique employed does incur an insignificant power overhead (0.2 percent). It should be noted here that the only additional storage structure in UnSync (CB) adds negligible power consumption overheads. We thus demonstrate here through accurate hardware synthesis experiments that the UnSync implementation is of significantly reduced chip-area and low-power consumption. Currently the above discussion is based on worst-case analysis with basic design optimizations at the design compiler and power analysis tools. Any circuit level optimization on the detection techniques, or circuit implementations, will only further reduce the overheads incurred. A discussion of the projected overheads when scaled for many-core systems is found in Section 4.1 of the online Supplemental File.

5.2 Negligible Performance Overhead in UnSync

5.2.1 No Performance Penalty from Serialization

To show the impact of serializing instructions, we analyze the performance variation on benchmarks which have serializing instructions. The FI in the Reunion implementation determines the granularity or frequency of synchronization, and therefore, we consider the baseline value for the interval as 10 instructions (smaller the better for Reunion). As we demonstrate here, the UnSync implementation is not affected by serializing instructions; we consider the baseline architecture as described in Section 4. We observe from the results in Fig. 5 that the Reunion implementation incurs an average of 8 percent performance overhead due to serializing instructions. Applications: *bzip2*, *ammp*, and *galgel* suffer from more than 10 percent overhead because they have more serializing instructions, which are 2 percent, 1.7 percent, and 1 percent of total instructions, respectively. However, *galgel* also suffers from increased ROB occupancy, and therefore, has the maximum overhead. On the other hand, UnSync demonstrates a consistently negligible variation (around 2 percent) in performance.

5.2.2 Larger CB Size Eliminates Performance Bottleneck

In UnSync, a core with a full CB has to stall and wait for the latest instruction to complete execution on the other CB for comparison and thereby eviction to L2. Therefore, if an application has a large number of write operations, it will affect the performance of system by stalling the cores when the CB is full. Fig. 6 shows the performance of UnSync across different CB sizes. We can see that when the CB size is small, the performance decreases, whereas larger CB sizes (2 kB and 4 kB) completely eliminate the resource occupancy bottleneck, and UnSync has almost identical performance with that of the baseline CMP architecture.

6 Conclusion

Growing technology scaling exposes modern and future processors to soft-error failures caused due to charge-carrying particles. In this paper, we propose UnSync—a customizable, error resilient, and power-efficient multicore architecture based on core-level redundancy. Our *always forward execution* enabled recovery mechanism coupled with an efficient choice of hardware-only detection techniques, reduce performance overheads in redundant designs, while also ensuring error resilience. We compare our architecture implementation on a multicore environment with that of Reunion (a state-of-the-art redundant error resilient technique). Experimental results show that the UnSync achieves up to 20 percent improvement in performance, with

13.32 percent reduced area and 34.5 percent less power overhead when compared to that of the Reunion architecture. In addition, we observe that the UnSync architecture achieves the same level of reliability, with a larger ROEC, and at lesser hardware overheads of power and chip-area.

7 Future Work

With the increasing parallelism in application software, and the drastic increase in the number of cores available in the CMP, our architecture opens doors to varied level of customization and programmability both at the compiler and application level to facilitate development of soft-error resilient applications. Since our architecture framework is independent of the underlying architecture within the core, more efficient hardware detection techniques (multibit correction for cache blocks, hardened pipeline registers, efficient register file protection, etc.) can be implemented. Our architecture and its working are unaffected by such modifications. In this work, we propose the copy-cache architecture to provide for a working implementation of the UnSync-CMP architecture when the L1 cache is in write-back mode of operation. Our experiments and analysis demonstrate the impact of this hardware as a proof of concept. Our future work involves extended design space exploration of such an architecture and its impact on the performance and power of the UnSync-CMP with a write-back L1 cache.

Acknowledgments

This work was partially supported by funding from the US National Science Foundation (NSF) grants CCF-0916652, CCF-1055094 (CAREER), NSF I/UCRC for Embedded Systems (IIP-0856090), Raytheon, Intel, SFAz, and Stardust Foundation, by the Basic Science Research Program through NRF funded by MEST (2012-015421), and by MKE under the Global Collaborative R&D Program supervised by KIAT (M002300089). Kyoungwoo Lee is the corresponding author.

References

[1] S. Kosonocky, V. Stojanovic, K.V. Berkel, M. Chao, T. Knoll, and J. Friedrich, "Power/Performance Optimization of Many-Core Processor SoCs," *Proc. IEEE Int'l Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, pp. 508-509, Feb. 2012.

[2] C. Slayman, "Alpha Particle or Neutron SER-What Will Dominate in Future IC Technology?" ewh.ieee.org/soc/cpmt/presentations/cpmt0910e.pdf, 2010.

[3] E. Ibe, H. Taniguchi, Y. Yahagi, K.-i. Shimbo, and T. Toba, "Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule," *IEEE Trans. Electron Devices*, vol. 57, no. 7, pp. 1527-1538, July 2010.

[4] D. Lyons, "Sun Screen: Soft Error Issue in Sun Enterprise Servers," www.members.forbes.com/global/2000/1113/0323026a.html, Nov. 2000.

[5] S. Kayali, "Reliability Considerations for Advanced Microelectronics," *Proc. IEEE Pacific Rim Int'l Symp. Dependable Computing (PRDC)*, p. 99, 2000.

[6] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier, "Multicore Soft Error Rate Stabilization Using Adaptive Dual Modular Redundancy," *Proc. Conf. Design, Automation and Test in Europe (DATE)*, 2010.

[7] D.D. Thaker, F. Impens, I.L. Chuang, R. Amirtharajah, and F.T. Chong, "On Using Recursive TMR as a Soft Error Mitigation Technique," citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.131.523, 2008.

[8] C.C. Corporation, "Data Integrity for Compaq Non-Stop Himalaya Servers," nonstop.compaq.com, 1999.

[9] J.C. Smolens, B.T. Gold, B. Falsafi, and J.C. Hoe, "Reunion: Complexity-Effective Multicore Redundancy," *Proc. IEEE/ACM 39th Ann. Int'l Symp. Microarchitecture (MICRO),* 2006.

[10] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro,* vol. 26, no. 4, pp. 52-60, July/Aug. 2006.

[11] J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett, and J. Gill, "MIPS: A Microprocessor Architecture," *Proc. 15th Ann. Workshop Microprogramming (MICRO),* 1982.

[12] C. Inc., "User Manuals for Cadence Encounter Tool Set Version 09.10-p104," cadence.com/products/ld/rtl_compiler, 2009.

[13] N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi, "CACTI 6.0: A Tool to Model Large Caches," www.hpl.hp.com/techreports/2009/HPL-2009-85.html, 2009.

[14] R. Jeyapaul, F. Hong, A. Rhisheekesan, A. Shrivastava, and K. Lee, "UnSync: A Soft Error Resilient Redundant Multicore Architecture," *Proc. Int'l Conf. Parallel Processing (ICPP '11),* pp. 632-641, http://dx.doi.org/10.1109/ICPP.2011.76, 2011.

[15] T.J. Slegel et al., "IBM's S/390 G5 Microprocessor Design," *IEEE Micro,* vol. 19, no. 2, pp. 12-23, Mar. 1999.

[16] P. Meaney, S. Swaney, P. Sanda, and L. Spainhower, "IBM z990 Soft Error Detection and Recovery," *IEEE Trans. Device and Materials Reliability,* vol. 5, no. 3, pp. 419-427, Sept. 2005.

[17] K. Meng, F. Huebbers, R. Joseph, and Y. Ismail, "Modeling and Characterizing Power Variability in Multicore Architectures," *Proc. Int'l Symp. Performance Analysis of Systems and Software (ISPASS),* 2007.

[18] N. Aggarwal, P. Ranganathan, N.P. Jouppi, and J.E. Smith, "Configurable Isolation: Building High Availability Systems with Commodity Multi-Core Processors," *Proc. 34th Ann. Int'l Symp. Computer Architecture (ISCA),* 2007.

[19] J.C. Smolens, B.T. Gold, J. Kim, B. Falsafi, J.C. Hoe, and A.G. Nowatzyk, "Fingerprinting: Bounding Soft-Error Detection Latency and Bandwidth," *Proc. ASPLOS-XI,* 2004.

[20] M. Gomaa, C. Scarbrough, T. Vijaykumar, and I. Pomeranz, "Transient-Fault Recovery for Chip Multiprocessors," *Proc. 30th Ann. Int'l Symp. Computer Architecture (ISCA),* pp. 98-109, June 2003.

[21] S.K. Reinhardt and S.S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA),* pp. 25-36, 2000.

[22] S.S. Mukherjee, M. Kontz, and S.K. Reinhardt, "Detailed Design and Evaluation of Redundant Multithreading Alternatives," *Proc. 29th Ann. Int'l Symp. Computer Architecture (ISCA),* 2002.

[23] S. Gupta, S. Feng, A. Ansari, B. Jason, and S. Mahlke, "StageNetSlice: A Reconfigurable Microarchitecture Building Block for Resilient CMP Systems," *Proc. Int'l Conf. Compilers, Architectures, and Synthesis for Embedded Systems (CASES),* pp. 1-10, 2008.

[24] A. Golander, S. Weiss, and R. Ronen, "DDMR: Dynamic and Scalable Dual Modular Redundancy with Short Validation Intervals," *Computer Architecture Letters,* vol. 7, no. 2, pp. 65-68, July 2008.

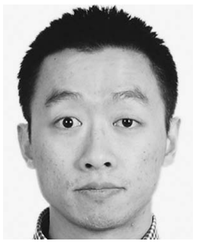
[25] R. Phelan, "Addressing Soft Errors in ARM Core-based Designs," technical report, ARM, 2003.

[26] A.A. Nair, L.K. John, and L. Eeckhout, "AVF Stressmark: Towards an Automated Methodology for Bounding the Worst-Case Vulnerability to Soft Errors," *Proc. IEEE/ACM Int'l Symp. Microarchitecture (Micro),* pp. 125-136, 2010.

[27] A. Nieuwland, S. Jasarevic, and G. Jerin, "Combinational Logic Soft Error Analysis and Protection," *Proc. IEEE 12th Int'l Symp. On-Line Testing (IOLTS),* pp. 99-104, 2006.

[28] L.S. Corp, "ECC Module Reference Design," www.latticesemi.com/products/intellectualproperty/referencedesigns/eccmodule.cfm, 2005.

Reiley Jeyapaul received the bachelor's degree in electronics and communication engineering from SRM Engineering College, University of Madras, Chennai, Tamil Nadu, India, and the master's degree in electrical engineering and the PhD degree in computer science both from Arizona State University, Phoenix. He is a postdoctoral researcher in the Compiler Microarchitecture Lab, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University. His research work is at the Compiler Microarchitecture interface for embedded and multicore systems, with the goal of improving its reliability, robustness and energy consumption. His research work in developing compiler based techniques for embedded and multicore systems was highlighted in an article by ASU News, and ACM Techn News. He is a student member of the IEEE.



Fei Hong received the BS degree in computer science and the MS degree in computer engineering both from Central South University, P.R. China, in 2004 and 2007, respectively, and the MS degree in computer science from Arizona State University, Phoenix, in 2011. He is currently working as a software engineer in Allied Telesis. His research interests include soft-error resilient CMP Architectures and high-performance memory subsystems. He is a student member of the IEEE.

Abhishek Rhisheekesan received the BE degree in electronics from Sardar Vallabhbhai National Institute of Technology, Surat, Gujarat, India, in 2003 and is currently working toward the MS degree in computer science from the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University (ASU), Phoenix. He is associated with the Compiler Microarchitecture Lab, ASU, under the guidance of Prof. Aviral Shrivastava. He has more than 7 years of industry experience, working at Intel Technologies India Pvt. Ltd. and Wipro Technologies, Bengaluru, India. His research interests include software and hardware solutions to provide protection against soft errors. He has recently published a conference paper in the International Conference on Parallel Processing (ICPP), 2011. He is a student member of the IEEE.

Aviral Shrivastava received the bachelor's degree in computer science and engineering from the Indian Institute of Technology, Delhi, the master's and PhD degrees in information and computer science from the University of California, Irvine. He is an associate professor in the School of Computing, Informatics, and Decision Systems Engineering at the Arizona State University (ASU), Phoenix, where he has established and heads the Compiler and Microarchitecture Lab (http://aviral.lab.asu.edu). His research interests include the intersection of compilers and architectures of embedded and multicore systems, with the goal of improving power, performance, temperature, energy, reliability, and robustness. His research is funded by US National Science Foundation (NSF) and several industries including Microsoft, Raytheon Missile Systems, Intel, Nvidia, and so on. He serves in organizing and program committees of several premier embedded system conferences, including ISLPED, CODES+ISSS, CASES, and LCTES, and regularly serves in NSF and US Department of Energy review panels. Right now, he is a visiting faculty in the Electrical Engineering and Computer Science Department, University of California, Berkeley. He is a 2011 NSF CAREER Award Recipient, and recipient of 2012 Outstanding Junior Researcher in CSE at ASU. He is a senior member of the IEEE.

Kyoungwoo Lee received the BS and MS degrees in computer science from Yonsei University, Seoul, Korea, in 1995 and 1997, respectively, and the PhD degree in information and computer science at the University of California, Irvine, in 2008. He is an assistant professor in the Department of Computer Science and Engineering, Yonsei University, where he has established and heads the Dependable Computing Lab (http://dclab.yonsei.ac.kr). His research interests include embedded systems, with a specific focus on cross-layer design and optimization for error-aware, and energy-efficient embedded systems. He is a member of the IEEE.