FINGERPRINTING: BOUNDING SOFT-ERROR-DETECTION LATENCY AND BANDWIDTH

FINGERPRINTING SUMMARIZES THE HISTORY OF INTERNAL PROCESSOR STATE UPDATES INTO A CRYPTOGRAPHIC SIGNATURE. THE PROCESSORS IN A DUAL MODULAR REDUNDANT PAIR PERIODICALLY EXCHANGE AND COMPARE FINGERPRINTS TO CORROBORATE EACH OTHER'S CORRECTNESS. RELATIVE TO OTHER TECHNIQUES, FINGERPRINTING OFFERS SUPERIOR ERROR COVERAGE AND SIGNIFICANTLY REDUCES THE ERROR-DETECTION LATENCY AND BANDWIDTH.

Jared C. Smolens
Brian T. Gold
Jangwoo Kim
Babak Falsafi
James C. Hoe
Andreas G. Nowatzyk
Carnegie Mellon University

Recent studies suggest that the soft-error rate in microprocessor logic is likely to become a serious reliability concern by 2010. Detecting soft errors in the processor's core logic presents a new challenge beyond what error-detecting and correcting codes can handle. Currently, commercial microprocessor systems that require an assurance of reliability employ an error-detection scheme based on dual modular redundancy (DMR) in some form—from replicated pipelines within the same die[1] to mirroring of complete processors.[2] These solutions, however, typically require drastic modifications in hardware and/or software to detect and recover from errors. Moreover, current designs rely on tight hardware integration of the DMR processor pair and high interprocessor communication bandwidth, which preclude cost-effective and scalable server architectures.

The TRUSS (Total Reliability Using Scalable Servers) project, is building a cost-effective, reliable server architecture from a tightly coupled cluster of rack-mounted server blades. The TRUSS architecture adds support for reliability with minimal changes to the commodity hardware and no changes to the application software. By enforcing distributed redundancy at all processing and storage levels, the TRUSS system can survive any single component failure (memory device, processor, or system ASIC). In the TRUSS architecture, a DMR processor pair is split across different nodes in a system area network. This distributed redundancy provides greater reliability but requires implementing DMR error detection under the limited communication bandwidth and nonnegligible communication latency of the system area network.

To detect errors across a distributed DMR pair, we developed fingerprinting, a technique that summarizes a processor's execution history into a cryptographic signature, or "fingerprint." More specifically, a fingerprint is a hash value computed on the changes to a processor's architectural state resulting from a program's execution. The mirrored processors in a DMR pair exchange and compare a small fingerprint to

Published by the IEEE Computer Society

corroborate each other's correctness over the instruction sequence. Fingerprints tightly bound error-detection latency and greatly reduce the required interprocessor communication bandwidth. Relative to two popular DMR error-detection approaches, fingerprinting is the only mechanism that simultaneously allows high error coverage, low error detection bandwidth, and high I/O performance.

Backward-error recovery

We evaluated DMR error detection within the context of a backward-error recovery (BER) framework that restarts execution from a checkpoint when the error-detection mechanism detects an error. A checkpoint/recovery mechanism forms the basis of all BER schemes. Figure 1 shows the sequence of actions we assume for checkpointing and error detection. We assume that the mirrored DMR processors execute redundantly in lockstep and that the processor microarchitecture is fully deterministic. Thus, because the two processors behave identically in an error-free scenario, an error in one manifests as a behavioral difference between the two.

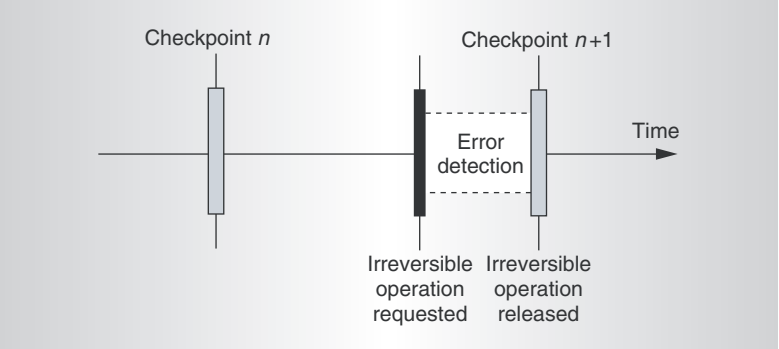

Figure 1: The error detection and checkpoint timeline for a single processor. Both processors in a DMR pair follow the same timeline.

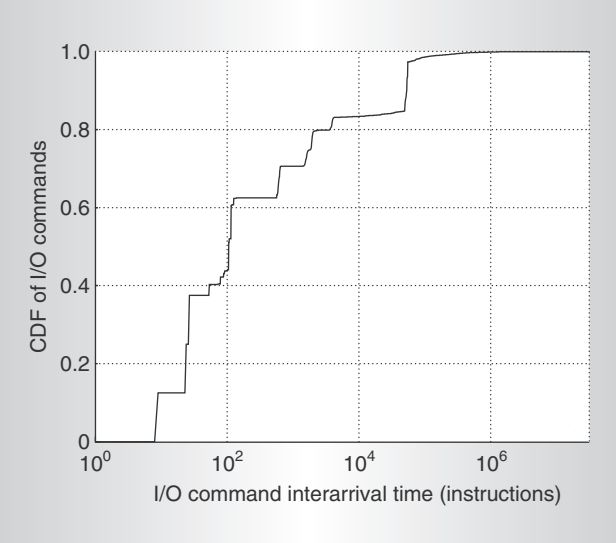

Figure 2: Cumulative distribution of SCSI command interarrival times during a TPC-C like workload.

Checkpointing

A checkpoint logically comprises a complete snapshot of architectural registers and memory values. Although the checkpoint can contain a copy of the entire architectural register file, the prohibitively large complete memory image means that the checkpoint records only changed memory values, using a copy-on-write mechanism (similar to SafetyNet).[3] Rollback consists of restoring register and memory values from the checkpoint. The time between checkpoints defines the checkpoint interval. For short intervals (hundreds to tens of thousands of instructions), checkpoints are small enough to fit in on-chip structures.[3,4]

Error recovery requires synchronized checkpoint and error-detection mechanisms. As Figure 1 shows, immediately before an operation with irreversible effects (such as an uncached load or store in I/O operations), the error-detection mechanism must observe any errors that have occurred since the last checkpoint. If the mechanism detects an error, program execution reverts to the last checkpoint; otherwise, if the mechanism does not detect an error, it releases the irreversible operation, discards the old checkpoint, and takes a new checkpoint to begin a new checkpoint interval. System state becomes unrecoverable if the error-detection mechanism fails to detect all errors before advancing into the next checkpoint interval. When the mechanism eventually detects a latent error in a subsequent checkpoint interval, it is not possible to recover the correct program state by reverting to an earlier checkpoint.

The interval between two I/O operations places a maximum bound on the distance between checkpoints. In commercial database workloads, I/O activity occurs frequently. Figure 2 shows the cumulative distribution of SCSI command interarrival times in a full-system TPC-C-like workload simulated for 100 billion instructions (50,000 database transactions). We

derived the times from a trace of reads and writes over the interval. Even though, on average, about 14,000 instructions separate each command, over half of all SCSI commands are within a few hundred instructions of each other. Therefore, a viable combination of checkpoint and error-detection mechanisms must be effective and efficient at checkpoint intervals as short as hundreds to thousands of instructions.

Current error-detection techniques

To evaluate fingerprinting's efficiency, we compared it with two current DMR error-detection techniques. The *chip-external* approach compares a DMR processor pair only at their chip-external interfaces. The *full state* approach compares all architectural state updates before each checkpoint creation.

Chip-external detection

The least intrusive DMR approach monitors and compares the two processors' external behaviors at the chip pins. We abstract the chip-external interface as address and data-bus traffic resulting from off-chip memory requests. The effect of an error originating in the execution core might not appear at the external pins for some time because of buffering internal to the processor, such as the registers and cache hierarchy. The exact error-detection latency depends on the program and the detailed microarchitecture. Detecting an error at the pins is sufficient in a fail-stop system, but the substantial and unbounded delay in error detection makes it difficult to recover and continue from an earlier known-good checkpoint.

Full state comparison

In a full state comparison between the mirrored DMR processors, the error-detection mechanism compares the entire register-file contents and all cache lines modified since the last checkpoint. This approach guarantees that the new checkpoint is error free (assuming the previous checkpoint is also error free). Unfortunately, full state comparison also requires substantially higher bandwidth than chip-external comparison.

Fingerprinting

Fingerprinting provides a complete and concise view of a processor's architectural state so as to detect differences in execution between two DMR processors. The result is high error coverage, little comparison bandwidth, and the ability to accommodate a wide range of checkpoint intervals.

How it works

Fingerprinting summarizes the history of architectural state updates as a cryptographic signature. Designers can add simple, non-intrusive fingerprinting hardware to a state-of-the-art processor core by passively monitoring the in-order architectural state updates in the pipeline's commit stage. An error-detection mechanism that uses fingerprinting detects errors by having the DMR processor pair exchange and compare fingerprints at the end of each checkpoint interval. A matching fingerprint indicates that the mirrored executions by two DMR processors remained in agreement during the last checkpoint interval. At this point, it is safe to replace the old checkpoint with a new one to begin a new checkpoint interval.

The fingerprinting hash function should be a well-constructed, linear block code that is compact and easy to generate and that has a low probability of undetected errors. A commonly used coding scheme is the cyclic redundancy code (CRC). Encoding the state updates in an interval in a p-bit CRC signature bounds the probability of an undetected error by fingerprint comparison to at most 2^{-p}.[5] For a soft error rate of 10^4 failures per billion hours, which future high-performance processors may exhibit,[6] a 16-bit CRC yields a mean-time-to-failure (MTTF) of more than 300,000 years.

Implementation

In a speculative, superscalar, out-of-order processor pipeline, the fingerprint computation monitors the committing instruction results, in program order, from the reorder buffer (ROB) and the load-store queue. If the microarchitecture has a physical register file, the fingerprint can also be computed by hashing the instruction results written into the register file in completion order. In this case, the fingerprint would include state updates by both committed and speculative instructions, possibly from the wrong path of a branch misprediction. Such a fingerprint is, nonetheless, effective in ensuring agreement between two DMR processors because the fingerprint

necessarily includes state updates from all committed instructions, and the processors also agree on the results of wrong-path instructions.

Evaluation results

To evaluate chip-external error detection, full state comparison, and fingerprinting, we simulated the execution of all 26 SPEC CPU 2000 benchmarks using SimpleScalar sim-cache and two commercial workloads using Virtutech Simics. The simulated processor executes one instruction per cycle at a clock frequency of 1 GHz. The only microarchitecture parameter relevant to our evaluation is the level-two (L2) cache configuration. The simulated processor has an inclusive 1-Mbyte four-way-set associative cache with 64-byte lines.

In SimpleScalar, we simulated the first reference input set for each SPEC CPU 2000 benchmark. Using the prescribed procedure from SimPoint,[7] we simulated up to eight predetermined 100-million instruction regions from each benchmark's complete execution trace. In Simics, we ran two commercial workloads on Solaris 8: a TPC-C-like online transaction processing (OLTP) workload with IBM DB2 and SPECWeb. The 100-client OLTP workload consists of a 40-warehouse database striped across five raw disks and one dedicated log disk. (We have calibrated this scaled-down database configuration to represent a full-scale system.) The SPECWeb workload services 100 connections with Apache 2.0. We warmed both commercial workloads until the CPU utilization reached 100 percent and the transaction rate reached steady state. Once warmed, the commercial workloads executed for 500 million instructions.

State-comparison bandwidth

We first evaluated the three error-detection mechanisms according to their requirements for state-comparison bandwidth between mirrored processors. For chip-external detection, we calculate this bandwidth requirement as the sum of the address and data-bus traffic from off-chip memory requests. The average chip-external bandwidths generated by the three application classes—SPEC CPU 2000 integer (SPEC CInt), SPEC CPU 2000 floating-point (SPEC CFP), and commercial—are

- SPEC CInt: 3.8 Mbytes per second (MB/s)
- SPEC CFP: 45.6 MB/s
- Commercial: 121.0 MB/s

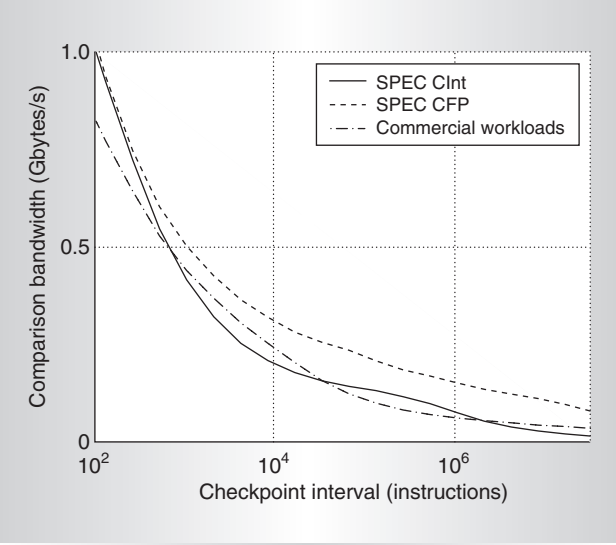

Figure 3. Required bandwidth for full state comparison as a function of the checkpoint interval.

Traditional DMR systems can easily handle these required bandwidths, since the mirrored DMR processors are on the same motherboard. However, except for SPEC CInt applications, whose working sets fit almost entirely within the L2 cache, the required bandwidth places a considerable burden on the system-area network when the mirrored DMR processors are on different nodes.

Full state comparison examines the entire register file contents and all cache lines modified since the last checkpoint. As the checkpoint interval increases, the spatial locality of memory references dictates that the number of updated cache lines grows at a slower rate. Assuming the full-state-comparison bandwidth amortizes over the entire checkpoint interval, the required bandwidth decreases as the checkpoint interval increases. In Figure 3, the average bandwidth requirement decreases sharply as the checkpoint interval increases. However, for the range of intervals compatible with I/O interarrival times of commercial workloads (hundred to thousands of instructions), the required bandwidth remains above several hundred megabytes per second.

Fingerprinting provides a compressed view of architectural state changes. Instead of comparing every instruction result, fingerprinting

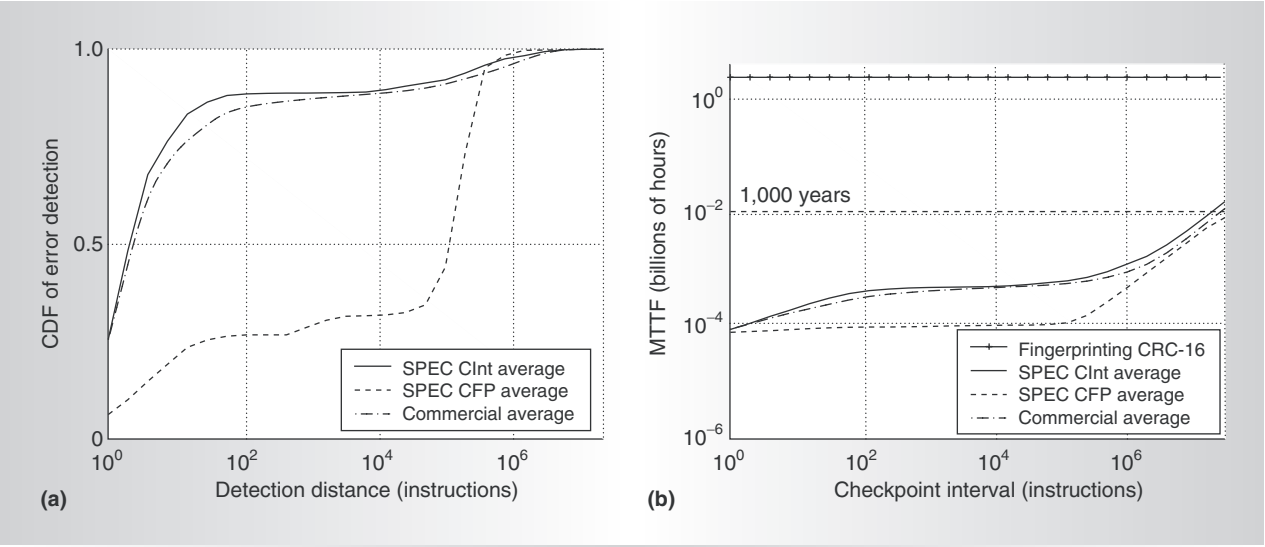

Figure 4. The cumulative distribution of error detection distances (a) and the mean time to failure, as a function of checkpoint interval (b).

compares only two bytes per checkpoint interval. The bandwidth overhead for fingerprint comparison is orders of magnitude less than that for full state comparison. Assuming a 1,000-instruction checkpoint interval on a 1,000-MIPS processor, fingerprinting consumes just 2 MB/s.

Error coverage

In a BER scheme, if a checkpoint contains an undetected error, the system can no longer recover when the error-detection mechanism later finds that error. Therefore, we define error coverage as the probability that an error occurring in a given checkpoint interval is detected by the end of that interval. Under this definition, full state comparison has perfect error coverage, since it compares all state updates explicitly before taking a new checkpoint. Fingerprinting's error coverage is not quite perfect because two different update sequences could result in the same CRC signature. (For a 16-bit CRC, the likelihood for signature collision is at most 2^{-16}.) With nonzero detection latency, chip-external detection is likely to miss errors near the end of a checkpoint interval, especially in the last L instructions where L is the expected chip-external detection latency. The achievable coverage with chip-external detection varies directly with the relative magnitude of the checkpoint interval and detection latency.

Detection distance. To evaluate chip-external detection's error coverage, we used a dataflow-analysis tool to measure the minimum distance (in instructions) from each instruction to its first possible error-detection opportunity at the chip-external interface (outside the cache hierarchy). To simplify the analysis, we disregarded the effect of fault masking (conservatively in favor of chip-external detection coverage). We also conservatively assumed that the chip-external detection mechanism can immediately observe erroneous branch targets and load/store effective addresses because these errors are likely to cause cache misses.

Figure 4a presents the resulting cumulative distribution function, which shows the probability of detecting an error within a given distance from an instruction's execution. All the benchmarks keep some values buffered in the cache for extended periods—a significant fraction of instruction results remain unobserved for millions of instructions. In particular, the periodic replacement patterns of the floating-point benchmarks result in poor coverage up to the L2 cache replacement period.

Mean time to failure. From the cumulative distribution of detection distance, we determined processor reliability in a DMR system. For an anticipated raw fault rate ($\lambda = 10^4$ failures per billion hours) in the logic circuits of high-

Work in error detection and checkpointing

Hewlett Packard's NonStop Himalaya is an example of a dual modular redundant (DMR) system with chip-external error detection.[1] When the detection mechanism observes a difference within the DMR processor pair, it shuts down both processors to prevent memory or disk corruption. Because of chip-external detection's unbounded detection latency, fine-grained, application-transparent recovery is not possible. Recovery requires coarse-grained software checkpointing and operating system intervention or a failover to a hot spare processor pair.

The IBM z900's G5 processor is a DMR design that duplicates instruction pipelines in the same core.[2] The detection mechanism corroborates the replicated pipelines' outputs before committing each instruction. The IBM G5's full-state detection mechanism allows fine-grained, automatic recovery after error detection, but the required comparison bandwidth is sustainable only by two pipelines on the same die.

For fault analysis, microprocessor designs incorporate scanout logic that can shift out the internal state of select flip-flops serially through an external pin, without disturbing the microprocessor's normal operation.[3] Signature-generation logic continuously scans flip-flops and, in every cycle, incorporates each flip-flop's new value into a scanout stream using XORs. Akin to fingerprinting, the signature stream at the scanout pin is a compressed summary of the history of all scanned flip-flops. Thus, the scanout signature can serve the same function as the fingerprint for DMR error detection.[4] Unlike fingerprinting, however, scanout signatures introduce a detection latency that could affect error coverage, and signature comparison cannot tie an error to a specific instruction interval.

SafetyNet incorporates a coherence-level invariant checker to detect errors in message generation and reception in shared-memory multiprocessors.[5] The SafetyNet checker computes signatures of coherence activity at each processor and memory controller. The checker verifies that for each cache-coherence upgrade message, there are corresponding downgrade messages at the sharer nodes. In contrast, fingerprinting implements error detection at the computation level and is orthogonal to schemes that target error detection in the memory system and/or interconnect.

With design goals ranging from increasing performance to improving reliability, many recent studies use architectural checkpointing. The rewind window on current out-of-order microarchitectures is only a few hundreds of instructions, but Akkary et al. have proposed using checkpointing in conjunction with an out-of-order microarchitecture scalable to windows of a few thousand instructions.[6] Gniady and Falsafi have proposed SC++, which aims to relax memory order speculatively and store the speculative state in the processor's register file and on-chip cache hierarchy, while maintaining a history of the prior architectural values in a custom queue.[7]

Others have proposed hardware checkpointing techniques with a granularity of hundreds of thousands to millions of instructions. SafetyNet augments on-chip caches with checkpoint buffers to hold old cache line values on the first write to a cache line in each checkpoint interval.[8] ReVive takes global checkpoints in main memory by flushing all caches and enforcing a copy-on-write policy for changed cache lines during the checkpoint interval.[9]

References

1. D. McEvoy, "The Architecture of Tandem's Nonstop System," *Proc. ACM '81 Conf.*, ACM Press, 1981, p. 245.
2. T.J. Slegal et al., "IBM's S/390 G5 Microprocessor Design," *IEEE Micro*, vol. 19, no. 2, March–April 1999, pp. 12-23.
3. Y.E. Hong et al., "An Overview of Advanced Failure Analysis Techniques for Pentium and Pentium Pro Microprocessors," *Intel Technology J.*, 1998, pp. 2-10.
4. E. Sogomonyan et al., "Early Error Detection in Systems-on-Chip for Fault-Tolerance and At-Speed Debugging," *Proc. VLSI Test Symp.*, IEEE CS Press, May 2001, pp. 184-189.
5. D.J. Sorin, M.D. Hill, and D.A. Wood, "Dynamic Verification of End-to-End Multiprocessor Invariants," *Proc. Int'l Conf. Dependable Systems and Networks*, IEEE CS Press, 2003, pp. 281-290.
6. H. Akkary, R. Rajwar, and S.T. Srinivasan, "Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors," *Proc. Int'l Symp. Microarchitecture (Micro 36)*, IEEE CS Press, 2003, pp. 423-434.
7. C. Gniady, B. Falsafi, and T.N. Vijaykumar, "Is SC + ILP = RC?," *Proc. Int'l Symp. Computer Architecture*, IEEE CS Press, 1999, pp. 162-171.
8. D.J. Sorin et al., "SafetyNet: Improving the Availability of Shared Memory Multiprocessors with Global Checkpoint/Recovery," *Proc. Int'l Symp. Computer Architecture*, IEEE CS Press, 2002, pp. 123-134.
9. M. Prvulovic, Z. Zhang, and J. Torrellas, "ReVive: Cost-Effective Architectural Support for Rollback Recovery in Shared-Memory Multiprocessors," *Proc. Int'l Symp. Computer Architecture*, IEEE CS Press, 2002, pp. 111-122.

performance processors early in the next decade,[6] the MTTF is a function of error coverage, C:

$$MTTF = \frac{1}{2\lambda(1-C)}$$

For chip-external detection, we determine error coverage for a checkpoint interval, t, as

$$C = \frac{\sum_{i=1}^{t} CDF(i)}{t}$$

Figure 4b is a plot of the MTTF for different workloads using chip-external detection. The same figure also reports the MTTF of fingerprinting, which is based on a 16-bit CRC. The MTTF target of 1,000 years for the IBM

Power4 system[8] serves as a reference. As the figure shows, in all the benchmarks, chip-external detection requires checkpoint intervals of at least 10 to 25 million instructions to achieve the target reliability. This requirement is incompatible with the practical upper bound on the checkpoint interval that I/O-intensive workloads demand. In contrast, fingerprinting maintains high error coverage, regardless of the checkpoint interval.

Traditional DMR error-detection schemes are not well-suited for the requirements of both detecting and recovering from soft errors. When used in conjunction with a checkpoint-recovery framework, the traditional chip-external error-detection approach requires checkpoint intervals greater than tens of millions of instructions to maintain the reference MTTF of 1,000 years.[8] This operating mode is incompatible with I/O-intensive workloads such as commercial OLTP applications, which demand checkpoint intervals in the thousands of instructions. On the other hand, brute-force solutions such as full state comparison require unacceptably high interprocessor bandwidth. Fingerprinting overcomes both obstacles. MICRO

Acknowledgments

We thank the anonymous ASPLOS 2004 reviewers for their valuable feedback on early drafts of this article. We also thank the SimFlex team at Carnegie Mellon for their simulation infrastructure.

This research was supported in part by NSF awards ACI-0325802 and CCF-0347560 and Intel Corp. An Intel equipment grant provided the computers we used in our research. Brian Gold's work was supported by graduate fellowships from NSF, Northrop Grumman, and the US DoD (NDSEG/HPCMO).

References

1. T.J. Slegal et al., "IBM's S/390 G5 Microprocessor Design," *IEEE Micro*, vol. 19, no. 2, March–April 1999, pp. 12-23.
2. D. McEvoy, "The Architecture of Tandem's Nonstop System," *Proc. ACM '81 Conf.*, ACM Press, 1981, p. 245.
3. D.J. Sorin et al., "SafetyNet: Improving the Availability of Shared Memory Multiprocessors with Global Checkpoint/Recovery," *Proc. Int'l Symp. Computer Architecture*, IEEE CS Press, 2002, pp. 123-134.
4. C. Gniady, B. Falsafi, and T.N. Vijaykumar, "Is SC + ILP = RC?" *Proc. Int'l Symp. Computer Architecture*, IEEE CS Press, 1999, pp. 162-171.
5. V. Pless, *Introduction to the Theory of Error-Correcting Codes*, 2nd ed., John Wiley & Sons, 1989.
6. P. Shivakumar et al., "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic," *Proc. Int'l Conf. Dependable Systems and Networks*, IEEE CS Press, 2002, pp. 389-398.
7. T. Sherwood et al., "Automatically Characterizing Large-Scale Program Behavior," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, IEEE CS Press, 2002, pp. 45-57.
8. D. Bossen, "CMOS Soft Errors and Server Design," *IEEE 2002 Reliability Physics Symp. Tutorial Notes, Reliability Fundamentals*, IEEE Press, 2002, pp. 121-07.1–121-07.6.

Jared C. Smolens is a PhD student in electrical and computer engineering at Carnegie Mellon University, where his research interests include reliable computer systems and multiprocessor system design. He received an MS in electrical and computer engineering from Carnegie Mellon University and is a student member of the IEEE.

Brian T. Gold is a PhD student in electrical and computer engineering at Carnegie Mellon University, where his research interests are reliable computer systems and multiprocessor system design. He received an MS in computer engineering from Virginia Tech and is a student member of the IEEE.

Jangwoo Kim is a PhD student in electrical and computer engineering at Carnegie Mellon University. His research interests include reliable computer architecture, multiprocessor architecture, and full computer system simulation. Kim received an MEng in computer science from Cornell University and is a student member of the IEEE.

Babak Falsafi is an associate professor of electrical and computer engineering at Carnegie

Mellon University. His research interests are computer architecture with emphasis on high-performance memory systems, nanoscale CMOS architecture, and tools to evaluate computer system performance. He received a PhD in computer science from the University of Wisconsin and is a member of the IEEE and ACM.

James C. Hoe is an assistant professor of electrical and computer engineering at Carnegie Mellon University. His research interests are in computer architecture and high-level hardware description and synthesis. He received a PhD in electrical engineering and computer science from MIT and is a member of the IEEE and ACM.

Andreas G. Nowatzyk is an associate professor of robotics at Carnegie Mellon University. His research interests are in shared-memory multiprocessor architectures and interconnect structures. He received a PhD in computer science from Carnegie Mellon University and is a member of the IEEE and ACM.

Direct questions and comments about this article to Babak Falsafi, Electrical and Computer Engineering Dept., Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213; babak@ece.cmu.edu.