Measuring Architectural Vulnerability Factors

PROCESSOR DESIGNERS NEED ACCURATE ESTIMATES OF SOFT-ERROR RATES EARLY IN THE DESIGN CYCLE TO MAKE APPROPRIATE COST-RELIABILITY TRADEOFFS. HERE, THE AUTHORS PRESENT A METHOD FOR ESTIMATING THE ARCHITECTURAL VULNERABILITY FACTOR—THE PROBABILITY THAT A FAULT IN A PARTICULAR STRUCTURE WILL RESULT IN AN ERROR.

Shubhendu S. Mukherjee
Intel

Christopher T. Weaver
Intel
University of Michigan

Joel Emer
Intel

Steven K. Reinhardt
Intel
University of Michigan

Todd Austin
University of Michigan

The continuous exponential growth in transistors per chip as described by Moore's law has spurred tremendous progress in the functionality and performance of semiconductor devices, particularly microprocessors. At the same time, each succeeding technology generation has introduced new obstacles to maintaining this growth rate. Transient faults caused by single-event upsets have emerged as a key challenge likely to gain significantly more importance in the next few design generations.

Techniques for dealing with these faults exist, but they come at a cost. Designers need accurate soft-error estimates early in the design cycle to weigh the benefits of error protection techniques against their costs. This article presents a method for generating these estimates.

Soft-error sources and impact

Single-event upsets arise when energetic particles—such as neutron particles from cosmic rays and alpha particles from packaging material—generate electron-hole pairs as they pass through a semiconductor device. Transistor source nodes and diffusion nodes can collect these charges. A sufficient amount of accumulated charge can invert the state of a logic device such as a latch, an SRAM cell, or a gate, thereby introducing a logical fault into the circuit's operation.[1] Because this type of fault does not reflect a permanent device malfunction, we call it *soft* or *transient*.

A device's rate of errors caused by single-event upsets depends on both the particle flux it encounters and its circuit characteristics. The particle flux depends on the environment. In Denver, Colorado, for example, at an altitude of 1.5 km, the neutron flux from cosmic rays is three to five times higher than the flux at sea level. Device circuit parameters that influence the error rate include the amount of charge stored, the vulnerable cross-section area, and the charge collection efficiency. As feature sizes shrink, the smaller amount of charge per device makes a particle strike more likely to cause an error, but the reduced cross section makes a strike on any given device less likely. These effects roughly cancel each other for latches and SRAM cells; thus, the error rate per latch or SRAM bit at a specific altitude will remain roughly constant or decrease slightly for the next several technology generations. However, in the absence of error correction schemes, the chip error rate will grow in direct proportion to the number of bits on the chip.

Thus, while Moore's law gives us exponential transistor count increases, this bounty comes at the cost of exponential error rate increases for unprotected chips.

Soft errors caused by cosmic rays are already making an impact in industry. According to Robert Baumann in an IEEE 2002 Reliability Physics Symposium tutorial, Sun Microsystems acknowledged in 2000 that cosmic ray strikes on unprotected cache memories had caused random crashes at major customer sites in its flagship Enterprise server line, losing a major customer to IBM as a result of this episode.[2] In 1996, Eugene Normand reported numerous incidents of cosmic ray strikes after studying the error logs of several large computer systems.[3] The fear of cosmic ray strikes prompted Fujitsu to protect 80 percent of the 200,000 latches in its recent Sparc processor with some form of error detection.[4]

Various techniques exist to deal with such faults, from special radiation-hardened circuit designs,[5] to localized error detection and correction (e.g., parity, ECC),[4] to architectural redundancy.[6-9] However, all these approaches introduce a significant penalty in performance, power, die size, and design time. Consequently, designers must carefully weigh the benefits of these techniques against their cost. Although a microprocessor with inadequate protection from transient faults might prove useless because of its unreliability, excessive protection can make the resulting product uncompetitive in cost or performance. Unfortunately, tools and techniques for estimating a processor's transient error rates are not readily available or fully understood. Furthermore, because it is best to design a comprehensive error-handling strategy into the processor from the ground up, designers need these estimates early in the design cycle.

Computing a processor's soft-error rate

We classify a processor's soft-error rate into two categories: silent data corruption (SDC) and detected unrecoverable error (DUE). SDC—the topic of this article—occurs when an unprotected bit sustains a single-bit upset leading to undetected incorrect system behavior. In contrast, a DUE event occurs when an error in a bit is detected (via parity checking, for example), but the system cannot recover from that error.

A key aspect of generating SDC rate estimates is that not all faults in a microarchitectural structure affect a program's final outcome. As a result, an estimate based only on raw device fault rates will be pessimistic, leading architects to overdesign their processor's fault-handling features. We call the probability that a fault in a processor structure will result in a visible error in a program's final output that structure's *architectural vulnerability factor* (AVF). For example, a single-bit fault in a branch predictor will not affect the sequence or results of any committed instructions. The branch predictor's AVF is thus 0 percent. In contrast, a single-bit fault in the committed program counter will cause the wrong instructions to execute, almost certainly affecting the program's result. Hence, the AVF for the committed program counter is effectively 100 percent.

The overall SDC rate of a microarchitectural structure is the product of its raw fault rate and its AVF. By summing the contributions of all on-chip structures, a processor architect can map the raw fault rate (dictated by process and circuit issues) to an overall processor SDC rate and thus determine whether the design meets its SDC rate goals (set for the target market). Significantly, this lets the architect examine relative contributions of various structures and identify the most cost-effective areas in which to use fault protection techniques.

To estimate AVFs, we use a new approach that tracks the subset of processor state bits required for *architecturally correct execution* (ACE)—any execution that generates results consistent with a system's correct operation as observed by a user. Any fault in a storage cell that contains one of these bits, which we call *ACE bits*, will cause a visible error in a program's final output in the absence of error correction techniques. We call the remaining processor state bits *un-ACE bits* because their specific values are *unnecessary for architecturally correct execution*. A fault that affects only un-ACE bits will not cause an error. The AVF of a single-bit storage cell is simply the fraction of time that it holds ACE bits. Assuming that all cells have equal raw fault rates, a structure's AVF is the average AVF of its storage cells or the average fraction of its cells holding ACE bits at any time.

The branch predictor's AVF is thus 0 percent because all predictor bits are always un-ACE bits. Similarly, all the bits in the committed program counter are always ACE bits, leading to an AVF of 100 percent. The real power of ACE-bit analysis lies in computing AVFs for structures that hold ACE bits at some times and un-ACE bits at other times—that is, most other processor structures. Rather than enumerate for each structure which bits might matter and which might not, we simply track the ACE bits through the pipeline, determine the average number of ACE bits in each structure in each cycle, and obtain the ratios of these numbers to the structures' bit capacities. We can use average ACE bits for this calculation because our fault-inducing particle strikes are randomly and uniformly distributed in a structure.

One straightforward application of our methodology is to count the ACE bits in a structure directly, using a performance model. We can also estimate a structure's AVF by counting the ACE bits that flow through the structure and applying Little's law, which states that the average capacity of an open system is the product of the bandwidth of individual objects flowing through the system and the average residence time of each object in the system.

It is difficult to precisely classify ACE and un-ACE bits over a program's entire execution. Instead, we assume conservatively that every bit is an ACE bit unless we can prove it is un-ACE. We thus compute an upper bound on the AVF number, obtaining a conservative estimate of a processor's SDC rate. We identify un-ACE bits at both the architectural and microarchitectural levels.

We identify five sources of architectural un-ACE bits:

- no-op instructions,
- performance-enhancing instructions (such as prefetches),
- predicated-false instructions,
- dynamically dead code, and
- logically masked values.

The results of dynamically dead instructions either are never used by any subsequent instruction in a program or are used only by other dynamically dead instructions. We can classify most bits of such instructions—except the opcode and some specific bits—as un-ACE when they are stored in processor structures.

Similarly, we identify four classes of microarchitectural un-ACE bits:

- idle or invalid,
- misspeculated (such as those for incorrect-path instructions),
- prediction only, and
- ex-ACE.

ACE bits become ex-ACE after their last use. Thus, for example, ACE instruction bits sitting in an instruction queue awaiting possible replay become ex-ACE if the replay does not occur.

Using Little's law to approximate AVFs

With Little's law,[10] we can compute the average number of ACE bits resident in a structure and, therefore, the structure's AVF. We translate Little's law as $N = B \times L$, where N is the average number of bits in a processor structure, B is the average bandwidth of bits per cycle into the structure, and L is the average residence time of an individual bit in the structure. Applying this equation to ACE bits, we obtain the average number of ACE bits in a structure as the product of the average bandwidth of ACE bits going into the structure (B_{ACE}) times the average residence cycles of an ACE bit in the structure (L_{ACE}). Thus, we can express a structure's AVF as

$$\frac{B_{\text{ACE}} \times L_{\text{ACE}}}{\textit{Total number of bits in hardware structure}}$$

This formulation is particularly useful in very early stages of an industrial processor's design cycle when even a performance model may not be available. Alternatively, in many cases, we can use hardware performance counters to compute the bandwidth of ACE bits going into a structure and the average residence cycles of ACE instructions, allowing AVF estimation without a performance model.

Using a performance model to compute AVFs

Using the methodology described earlier, dynamic slices of the SPEC CPU2000 benchmark suite, and a performance model written in the Asim framework,[11] we computed the

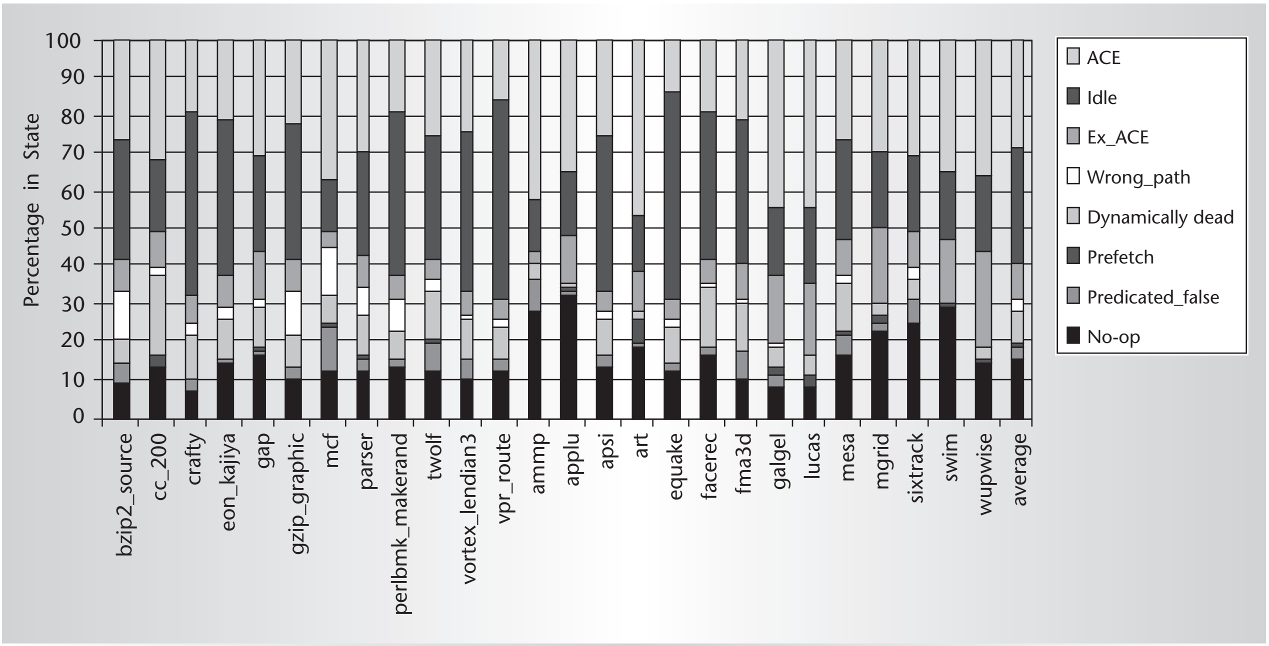

Figure 1. Breakdown of architectural and microarchitectural states for the Itanium 2-like processor's instruction queue. The AVF is the percentage of bits in the instruction queue that are ACE bits. The horizontal axis plots the SPEC CPU2000 benchmark suite.

AVF for an Itanium 2-like processor's instruction queue. Figure 1 plots the state bit breakdown for the instruction queue; a similar plot for the execution units is available in another publication.[12] We found that the instruction queue's AVF ranges from 14 percent to 47 percent, and the execution units' AVF ranges from 4 percent to 27 percent. Because our methodology is conservative, these fractions are upper bounds on the AVF numbers.

Further refinement of this analysis (for example, derating the hint bits in an IA-64 load instruction) could further lower the AVF estimates. Wang, Fertig, and Patel describe additional instances in which a fault in a conditional branch instruction might not result in an error in a program's final output.[13] Nevertheless, we believe we have captured most of the dominant AVF effects for the the Itanium2-like microarchitecture we examined, and, hence, we expect the contribution of further refinement to be small.

Related work

Earlier work in estimating AVFs has applied statistical fault injection to hardware register-transfer-level (RTL) models. Kim and Somani made a systematic fault injection study of Sun Microsystems' publicly disclosed picoJava II RTL model and reported widely varying AVFs for picoJava II hardware structures.[14] Wang and Patel injected faults into an RTL model of the Alpha 21164 processor and reported AVFs of less than 10 percent for the pipeline latches at the Center for Circuits, Systems, and Software's second annual review in 2003. The greatest advantage of using an RTL model is that it usually has all the hardware structures necessary to create a processor. In contrast, a performance model, which we used in our study, has only components that affect a processor's performance. Consequently, we can report only the AVFs of modeled components. Nevertheless, the highly detailed performance models used in the industry capture significant portions of the processors under design.

Our methodology improves upon statistical fault injection into RTL models in four ways. First, ACE-bit analysis provides deterministic AVF estimates in a single experiment for any benchmark. In contrast, statistical fault injection requires many experiments per benchmark to reach a statistically significant sample size. Second, by recognizing values that are dead or masked, ACE analysis provides a more comprehensive determination of whether faults affect processor operation, resulting in tighter AVF estimates. Third, ACE analysis gives useful insight into system behavior, such as a breakdown of why un-ACE bits do not contribute to the program result. The application of Little's law to rough AVF estimation exemplifies the usefulness of the more abstract nature of ACE analysis. Finally, fault injection into an RTL model requires an RTL model, which is generally not available during a microprocessor design project's early stages. In contrast, a performance model, which we use for ACE analysis, is usually available at the beginning of a microprocessor design project.

Using the techniques we've described, microprocessor designers can estimate the per-structure AVF numbers and hence an entire chip's SDC rate relatively quickly. If the microprocessor does not meet the target SDC rate (as set by the company or market), these AVF estimates can help designers choose the appropriate error detection or correction schemes, such as parity or error-correcting codes, to make specific structures invulnerable to single-bit upsets. Large structures with high AVFs, for example, would be obvious candidates for such error protection. Thus, microprocessor designers can iteratively lower the chip's SDC rate by adding increased error protection, using AVF estimates as a guide. MICRO

References

1. J.F. Ziegler et al., "IBM Experiments in Soft Fails in Computer Electronics (1978–1994)," *IBM J. Research and Development*, vol. 40, no. 1, Jan. 1996, pp. 3-18.
2. R. Baumann, "Soft Errors in Commercial Semiconductor Technology: Overview and Scaling Trends," *IEEE 2002 Reliability Physics Symp. Tutorial Notes, Reliability Fundamentals*, IEEE Press, 2002, pp. 121-01.1–121-01.14.
3. E. Normand, "Single-Event Upset at Ground Level," *IEEE Trans. Nuclear Science*, vol. 43, no. 6, Dec. 1996, pp. 2742-2750.
4. H. Ando et al., "A 1.3GHz Fifth Generation SPARC64 Microprocessor," *Proc. IEEE Int'l Solid-State Circuits Conf.* (ISSCC 03), IEEE Press, 2003, pp. 246-247.
5. T. Calin, M. Nicolaidis, and R. Velazco, "Upset Hardened Memory Design for Submicron CMOS Technology," *IEEE Trans. Nuclear Science*, vol. 43, no. 6, Dec. 1996, pp. 2874-2878.
6. T.J. Slegel et al., "IBM's S/390 G5 Microprocessor Design," *IEEE Micro*, vol. 19, no. 2, Mar.-Apr. 1999, pp. 12-23.
7. T.M. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," *Proc. 32nd Ann. Int'l Symp. Microarchitecture* (MICRO-32), IEEE CS Press, 1999, pp. 196-207.
8. 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 02), IEEE CS Press, 2002, pp. 99-110.
9. S.K. Reinhardt and S.S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," *Proc. 27th Ann. Int'l Symp. Computer Architecture* (ISCA 2000), ACM Press, 2000, pp. 25-36.
10. E.D. Lazowska et al., *Quantitative System Performance*, pp. 42-46, Prentice-Hall, 1984.
11. J. Emer et al., "Asim: A Performance Model Framework," *Computer*, vol. 35, no. 2, Feb. 2002, pp. 68-76.
12. S.S. Mukherjee et al., "A Systematic Methodology to Compute the Architectural Vulnerability Factors of a High-Performance Microprocessor," to be published in *Proc. 36th Ann. Int'l Symp. Microarchitecture* (MICRO-36), IEEE CS Press, 2003.
13. N. Wang , M. Fertig, and S. Patel, "Y-Branches: When You Come to a Fork in the Road, Take It," *Proc. 12th Int'l Conf. Parallel Architectures and Compilation Techniques* (PACT 03), IEEE CS Press, 2003, pp. 56-67.
14. S. Kim and A.K. Somani, "Soft Error Sensitivity Characterization for Microprocessor Dependability Enhancement Strategy," *Proc. Int'l Conf. Dependable Systems and Networks* (DSN 02), IEEE CS Press, 2002, pp. 416-428.

Shubhendu S. Mukherjee is a senior staff engineer at Intel's Massachusetts Microprocessor Design Center, where he leads the Fault-Aware Computing Technology (FACT) project. His research interests include processor reliability and interconnection networks. Mukherjee has a BTech in Computer Science and Engineering from the Indian Institute of Technology, Kanpur, and an MS and a PhD in Computer Science from the University of

Wisconsin-Madison. He is a member of ACM and and a senior member of IEEE.

Christopher T. Weaver is a hardware engineer in the VSSAD labs at Intel's Massachusetts Microprocessor Design Center, where he works in the Fault-Aware Computing Technology project. His research interests include reliable computing, performance modeling, semi-custom and ASIC design, and power and fault modeling. Weaver has BS degrees in electrical engineering, computer engineering, and multidisciplinary studies from North Carolina State University and an MS in computer science and engineering from the University of Michigan.

Joel Emer is an Intel fellow in the VSSAD group at Intel. His research interests include high-performance microarchitecture, multithreaded processors, processor pipeline organization, processor reliability, and performance-modeling frameworks. Emer has a PhD in electrical engineering from the University of Illinois. He is a Fellow of the IEEE.

Steven K. Reinhardt is an associate professor of electrical engineering and computer science at the University of Michigan in Ann Arbor, and a consultant to Intel. His research interests include processor architecture, memory systems, and computer system simulation. Reinhardt has a BS from Case Western Reserve University and an MS from Stanford University, both in electrical engineering, and a PhD in computer science from the University of Wisconsin-Madison. He is a member of the IEEE Computer Society and of ACM.

Todd Austin is an associate professor of electrical engineering and computer science at the University of Michigan in Ann Arbor. His research interests include computer architecture, compilers, computer system verification, and performance analysis tools and techniques. Austin has a PhD in computer science from the University of Wisconsin.

Direct questions and comments about this article to Shubhendu S. Mukherjee, MMDC, Intel Corp., 334 South St., Shrewsbury, MA 01545; shubu.mukherjee@intel.com.