
RELIABILITY: FALLACY OR REALITY?

AS CHIP ARCHITECTS AND MANUFACTURERS PLUMB EVER-SMALLER PROCESS

TECHNOLOGIES, NEW SPECIES OF FAULTS ARE COMPROMISING DEVICE RELIABILITY.

FOLLOWING AN INTRODUCTION BY ANTONIO GONZÁLEZ, SCOTT MAHLKE AND SHUBU

MUKHERJEE DEBATE WHETHER RELIABILITY IS A LEGITIMATE CONCERN FOR THE

MICROARCHITECT. TOPICS INCLUDE THE COSTS OF ADDING RELIABILITY VERSUS THOSE OF

IGNORING IT, HOW TO MEASURE IT, TECHNIQUES FOR IMPROVING IT, AND WHETHER

CONSUMERS REALLY WANT IT.

Antonio González

Intel

Scott Mahlke

University of Michigan

Shubu Mukherjee

Intel

Resit Sendag

University of Rhode

Island

Derek Chiou

University of Texas at

Austin

Joshua J. Yi

Freescale Semiconductor

Moderator's introduction: Antonio González

..... Technology projections suggest that Moore's law will continue to be effective for at least the next 10 years. Basically, as Figure 1 shows, in each new generation devices will continue to get smaller, become faster, and consume less energy. However, the new technology also brings along some new cotravelers. Among them are variations, which manifest in multiple ways. First, there are variations caused by the characteristics of the materials and the way chips are manufactured; these are called process variations. There are multiple types of process variations: spatial and temporal, within die and between dies. Random dopant fluctuations are one type of process variation. Second, there are voltage variations, such as voltage droops. Third, there are variations caused by temperature. Temperature affects many key parameters, such as delay and energy consumption. Finally, there are variations due to inputs. A given functional unit behaves differently—in terms of delay, energy, and other parameters—depending on the data input sets.

Faults are another group of cotravelers that accompany new technology. Faults have multiple potential sources. One of these is radiation particles; it is expected that

future devices will be more vulnerable to particle strikes. Another source of faults is wear-out effects, such as electromigration. Finally, faults are also caused by variations. When variations are high, we might want to target designs to the common case rather than the worst case. When the worst case occurs, the system would need to take some corrective action, to continue operating correctly.

Basically, the topic of the panel is these faults. We typically classify faults into three main categories:

- *Transient* faults appear for a very short period of time and then disappear by themselves. Particle strikes are the most common type of transient fault.
- *Intermittent* faults appear and disappear by themselves, but the duration can be very long—that is, undetermined. Voltage droops are an example of this type of fault.
- *Permanent* faults remain in the system until a corrective action is taken. Electromigration is an example of this type of fault.

Some of these faults have a changing probability of occurrence over the lifetime

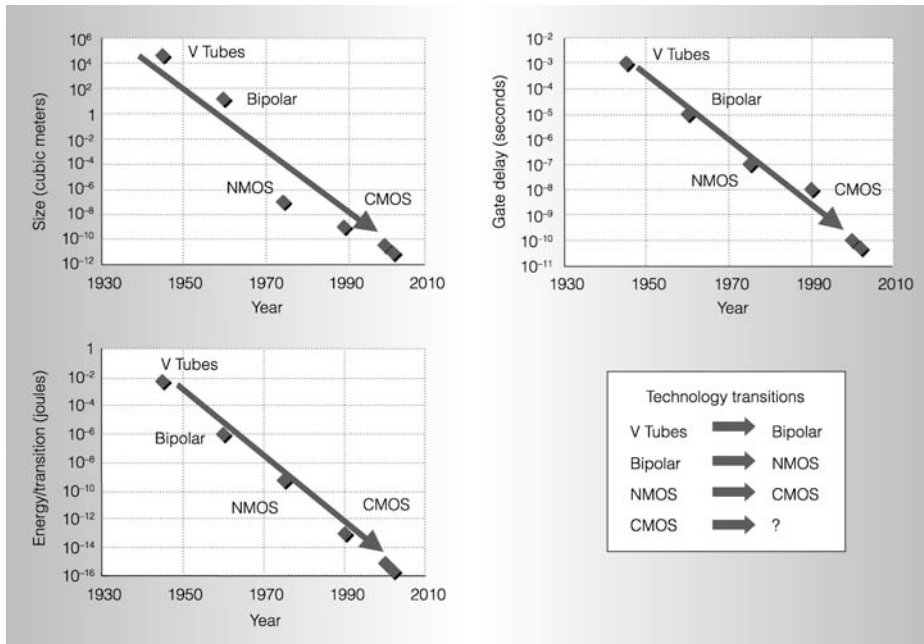


Figure 1. Technology scaling trends: more, faster, less-energy transistors.

of a device. Failure rates during device lifetimes exhibit a bathtub curve behavior. At the beginning, during the period called *infant mortality* (1 to 20 weeks), the probability of a fault occurring is relatively high. Then, during the normal lifetime, the probability is far lower. Finally, during the wear-out period, the probability starts to increase again.

Many questions remain open in reliability research. First, what will be the magnitude of these faults, and what opportunities will arise from exploiting these variations? What will the impact of particle strikes be in the future? What is the degree of wear-out in the typical lifetime of a processor? Will reliability be critical to keep high yields? How much does the processor contribute to the total faults in a system? Is it really an important part of the problem, or can architects just ignore it and take care of the other parts?

Is the microarchitecture the right level at which to address these issues? Are system-, software-, or circuit-level solutions preferable? Which types of solution are the most adequate, feasible, and cost-effective?

Is ignoring reliability an option? Do we need schemes and methods just to anticipate

and detect faults? Or, do we need mechanisms to detect and correct these faults? For instance, some parts that fail (or are likely to fail) might be transparently corrected, just as when you go to a mechanic to replace weak or failing parts of your car. What will be the cost of such solutions? Will all users be willing to pay for the cost of reliability, or would only certain classes of users (for example, users of large servers) be willing to pay? Does reliability depend on the application? Are certain applications more sensitive? Is reliability going to be a mainstream architecture consideration, or is it going to be limited to a niche of systems and applications? All of these remain to be answered.

Until recently, reliability has been addressed in the manufacturing process (through burn-in and testing) and through circuit techniques, whereas microarchitecture techniques have focused primarily on mission-critical systems. However, over the past 5 to 10 years, reliability has moved more into the mainstream of computer architecture research. On the one hand, transient and permanent faults due to CMOS scaling are a looming problem that must be solved. In a recent keynote address,

Shekhar Borkar summed up the emerging design space as follows: “Future designs will consist of 100 billion transistors, 20 billion of which are unusable due to manufacturing defects; 10 billion will fail over time due to wear-out, and regular intermittent errors will be observed.”¹ This vision clearly suggests that fault tolerance must become a first-class design feature.

On the other hand, some people in the computer architecture community believe that reliability will provide little added value for most of the computer systems that will be sold in the near future. They claim that researchers have artificially enhanced the magnitude of the problems to increase the perceived value of their work. In their opinion, unreliable operation has been accepted by consumers as commonplace, and significant increases in hardware failure rates will have little effect on the end-user experience. From this perspective, reliability is simply a tax that the doomsayers want to levy on your computer system.

The goal of this panel is to debate the relevance of reliability research for computer architects. Highlights of the discussion follow the panelists’ position statements.

The need for reliability is a fallacy: Scott Mahlke

I believe there is a need for highly reliable microprocessors in mission-critical systems, such as airplanes and the Space Shuttle. In these systems, the cost of the computer systems is not a dominant factor; the more important reality is that people’s lives are at stake. However, for the mainstream computer systems used in consumer and business electronic devices, the need for reliability is a fallacy. Starting with the most obvious and working toward the least obvious, the following are the top five reasons why computer architecture research in reliability is a fallacy.

Reason 1: It’s the software, stupid!

The unreliability of software has long dominated that of hardware. Figure 2 shows the failure rates for several system components. The failures per billion hours of operation (also called failures in time, or FITs) in Microsoft Windows are an order of magnitude higher than corresponding values for all hardware components. Bill Gates has stated that the average Windows machine will fail, on average, twice a month. In fact, when operating systems start off, they fail very frequently. Mature operating systems can have a mean time to failure (MTTF) measured in months, whereas a newer operating system might crash every few days.

This is not intended as a bash of Microsoft or other software companies. Software is inherently much more complex than hardware, and software verification is an open research question. The bottom line is that to improve the reliability of current systems for the user, the focus should be on the software, not the hardware.

Reason 2: Electronics have become disposable

One of the big issues that researchers are examining today is transistor wear-out and how to build wear-out-tolerant computer systems. But, the majority of consumers care little about the reliable operation of electronic devices, and their concerns are decreasing as these devices become more disposable. In 2006, the average lifetime of a business cell phone was nine months. The average lifetimes of a desktop and a laptop computer were about two years and one year, respectively. Most electronic devices are replaced before wear-out defects can manifest. Therefore, building devices whose hardware functions flawlessly for 20 years is simply unnecessary. Furthermore, from the economic perspective, reliability can be quite expensive in terms of chip area, power consumption, and design complexity. Thus, it’s often not worth the cost.

Reason 3: A transient fault is about as likely as my winning the lottery

Data from the IBM z900 server system shows that three transient faults occurred in 198 million hours of operation, or about

About this article

Derek Chiou, Resit Sendag, and Josh Yi conceived and organized the 2007 CARD Workshop and transcribed, organized, and edited this article based on the panel discussion. Video and audio of this and the other panels can be found at <http://www.ele.uri.edu/CARD/>.

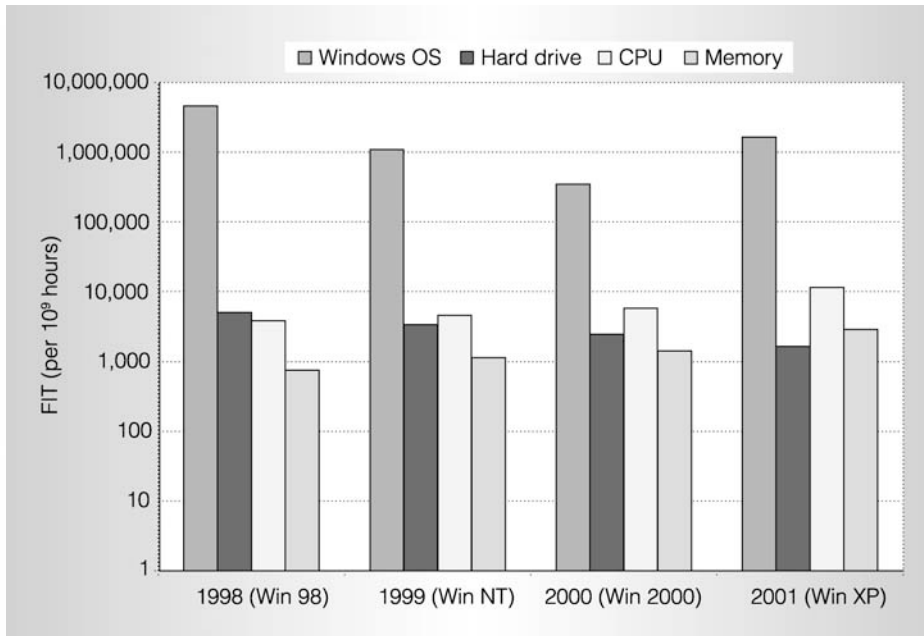


Figure 2. Failures in billions of hours of operation.²⁻⁵

one fault every 2.7 million days. One of my favorite comments about this was that it was much more likely that somebody would walk by and kick the power cord than it was that an actual transient fault would occur.

Let's compare the rate of transient faults to some other things that we don't think about in our everyday lives. My chance of winning the lottery is about equal: 1 in 3 million. The chance of getting struck by lightning in a thunderstorm is about twice that of a transient fault: 1 in 1.4 million. How about getting murdered? The chance is about 1 in 10,000 in the United States. The chance of being involved in a fatal car crash is about 1 in 6,000, and the chance of a plane crashing is 1 in 10 million. The point is that we don't constantly worry about these things happening to us, so do we really need to be concerned about a transient fault happening? The chances are so unlikely that the best thing to do about them may be to ignore them.

Reason 4: Does anyone care?

In many situations, 100 percent reliable operation of hardware is not important or worth the extra cost. For instance, no one can tell the difference if a few pixels are incorrect in a picture displayed on their

laptop. Imagine a streaming video coming through; no one would care if occasionally a pixel on a frame had the wrong color. How often do you lose a call on your cell phone? How often is a word garbled and you have to ask the person on the other end to repeat something? The majority of consumers today either do not notice or readily accept imperfect operation of electronic devices. There is also a lot of redundancy in applications such as streaming video, so maybe the answer is that software, rather than hardware, should be made more resilient.

Reason 5: This problem is better solved closer to the circuit level

Even if you accept reliability as a problem for microprocessor designers, one important question is at what level of design should the problem be solved—architectural or circuit? An example of the circuit-level approach is Razor, which introduced a monitoring latch.⁶ Essentially, Razor is an in situ sensor that measures the latency of a particular circuit. It enables timing speculation and the detection of transient pulses caused by energetic particle strikes. The point is that architects may not need to worry about solving these problems. Rather, techniques

like Razor can handle reliability problems beneath the surface, where the area and power overhead is lower. Another important factor is that many designs can benefit from circuit-level techniques, so point solutions need not be constructed. Finally, in situ solutions naturally handle process variation.

The need for reliability is a reality: Shubu Mukherjee

Captain Jean-Luc Picard of the starship *USS Enterprise* once said that there are three versions of the truth: your truth, his truth, and the truth. The God-given truth is that circuits are becoming more unreliable due to increasing soft errors, cell instability, time-dependent device degradation, and extreme device variations. We have to figure out how to deal with them.

The user's truth

Users care deeply about the reliability of their systems. If they get hundreds of errors a day, they will be unhappy. But, if the number of hardware errors is in the noise relative to other errors, hardware errors may not matter as much to them. That is exactly the direction in which the entire industry is moving. The goal of hardware vendors is to maintain a low enough hardware error rate that those errors continue to be obfuscated by software crashes and bugs. However, there have always been point risks that make certain individual corruption or crashes critical—for example, a Windows 98 crash during a Bill Gates demo—even if such errors occur rarely.

The IT manager or vendor's truth

The truth, however, is very different from the perspective of an IT manager who has to deal with thousands of users. The greater the number of user complaints per day, the greater is her company's total cost of ownership for those machines. It is like the classic light bulb phenomenon: The more you have, the sooner at least one of them will fail. And that's what we see in many houses. In a house with 48 light bulbs, each with 4 years MTTF, we replace a light bulb every month. These failures negatively impact business, because billions

of dollars are involved. User-visible errors, even when few surface, have an enormous impact on the industry. They increase cost because companies start getting product returns. Companies can face product returns even for soft errors, because users sometimes demand replacement of parts. In addition, there is the issue of loss of data or availability.

The designer's awakening

The designer's awakening is an experience similar to going through the four stages of grief. First, you have the shock: "Soft errors (SERs) are the crabgrass in the lawn of computer design." This is followed by denial: "We will do the SER work two months before tape out." Then comes anger: "Our reliability target is too ambitious." Finally, there is acceptance: "You can deny physics only so long." These are all real comments by my colleagues. The truth is, designers have accepted silicon reliability as a challenge they will have to deal with.

The designer's challenge

The industry is addressing the reliability problem with the help of research community. We need solutions at every level. Protection comes at many levels: at the process level through improved process technology; at the materials level through shielding for alpha particles; at the circuit level through radiation-hardened cells; at the architecture level through error-correcting code (ECC), parity, hardened gates, and redundant execution; and at the software level through higher-level detection and recovery. Companies are doing a lot. They are constantly making trade-offs between the cost of protection (in terms of performance and die size) and chip reliability, without sacrificing the minimum performance, reliability, and power thresholds that they must achieve.

Industry needs universities' help with research

Industry needs help from academia, but academia has some misconceptions about reliability research. One of the misconceptions concerns mean time between failures (MTBF), which is only a rough estimate of

an individual part's life. Using MTBF to predict the time to failure (TTF) of a single part is fundamentally flawed, because MTBF does not apply to a specific part. Thus, we cannot start optimizing lifetime reliability on the basis of MTBF.

Another common misconception is the notion that a system hang doesn't cause data corruption. However, if you cannot prove otherwise, you should assume it does cause data corruption because your data might already have been written to the disk before the system hangs. Finally, one other misconception is that adding protection without correction reduces the overall error rate, but in reality it does not.

Many questions remain unanswered in different areas of silicon reliability, and industry needs help from the universities. How do we predict and/or measure error rate from radiation, wear-out, and variability? How do we detect soft errors, wear-out, and variability on individual parts? Many traditional solutions exist, but how do we make them cheaper?

Cost of reliability—Are users willing to pay?

Mahlke: The big question is how much are the people willing to pay? This is very market dependent. For example, a credit card company trying to compute bills would be willing to pay a fair amount of money. But what about the average laptop user, how much extra are they willing to pay? My theory is that end users are not willing to pay. Either they are used to errors, they accept them, or they don't care about infrequent errors.

González: I want to add that cost could also be reflected in some performance penalty. Reliability can be sometimes provided at the expense of some decrease in performance—for example, lower frequency—or some increase in power due to the extra hardware.

Mukherjee: If you look back, people pay for ECC. We do. We pay for parity, we pay for RAID systems (redundant arrays of independent disks), we pay a lot for fault-tolerant file servers from EMC. So, people pay. The main thing that we need to do is

to measure and show them what they are paying for. It turns out that every company has some applications that they need to run with much more reliability than others. E-mail, surprisingly, is one of them. Financial applications are another. So, yes, they are willing to pay if we show them what they are paying for.

González: Following up on that, do we have any quantification of reliability in terms of area or any other metric? How much is already in the chip today to guarantee certain levels of reliability?

Mukherjee: The amount of area that we are putting in for error correction logic is going up exponentially in order to keep a constant MTTF. And it will continue to grow.

Mahlke: Yes, but most of that error protection logic is in the memory, right? There is not much in the actual processor.

Mukherjee: Not necessarily. I cannot publicly reveal the details. Mean time to failure—what does it tell us?

Audience member: Why do we make products with MTTF of seven years, when most of the users are going to throw them away in one year? It's just a matter of doing the mathematics. It all depends on the distribution of the failures versus time. You can have 90 percent of your population failing in one year, and still have an MTTF of seven years with the right distribution. So, just because my MTTF is seven years doesn't mean that unacceptably large fractions of the people are not going to see failures in one year.

Mahlke: I think you are right. Just because the MTTF is seven years, it doesn't mean that all fail at seven years. Many of them will fail before that. But, I think if you look at it, people are keeping these things 11 months and then throwing them away. If you look at the data for how many phones actually fail after 11 months, I believe that it is a very small number, even when there are hundreds of millions of phones sold each year. And, if we just replace those phones, and each phone is

\$100 or something, the cost is relatively small.

The counter-argument is this: Let's say I am going to add \$10 worth of electronics for reliability to each phone. Does the average phone customer want to pay that money to get that little bit of extra reliability? I think there is a big distinction between servers that are doing important computation and disposable electronic devices that people use. Maybe we need two different reliability strategies for these two domains. Because, for disposable electronics, where we try to reduce the cost, it may be too much overhead if we blindly incorporate reliability mechanisms such as parity bits or dual modular redundancy (DMR) into the hardware.

Mukherjee: We don't specify lifetime reliability based on the mean, but rather on a very high percentile of the chips surviving whatever number of years we internally think the chips should survive. So, it is not based on the mean.

Audience member: I would like to add one sentence to that. I have seen graphs from Intel that are publicly available. They show that this number of years for this 99.99+ percent of chips is going down. The reason is that it is harder to give the same guarantee.

Mukherjee: Good observation.

Error detection and correction

Audience member: One of the things I find disturbing is silent failure. If the hardware failure is a silent failure, I don't know if my data is corrupted, I have no indication. So, in these systems, if I have error detection, then I can track the data being corrupted and follow on. But, if we are having bit-flipping hardware, I don't have detection and neither have I correction. Detection is not correction, I agree, but detection is one of the reasons we tolerate failures in software and supplement systems.

Mukherjee: That's a very good point. I was in the same camp for a while, but after interacting with some of the customers, I am beginning to think otherwise, because

fault detection raises your overall error rate. Silent data corruption falls into two classifications: the one that you care about, and the one that you don't care about. When you put fault detection to prevent silent data corruption, you end up flagging both types of these errors, and the customer annoyance factor goes up. The bottom line is that detection alone is not enough. You have to go for full-blown correction.

González: So, there are errors that matter and errors that don't—but are we not using the same kind of systems for both? For instance, you may be checking your bank account with the same computer that you're using to run your media entertainment.

Mahlke: If you are accessing something and you knew there was an error in something small, maybe your address book, you can download a copy of it, right? But, if there was something larger than that, and maybe you didn't have a copy of the data—or maybe, as Antonio [González] said, you were doing something critical like transferring money from your bank account—then I kind of agree with Shubu [Mukherjee] that detection alone may not be good enough. If you want to go down this reliability path, you may need to detect and correct, because detection just throws many red flags and you start worrying about what you lost versus fixing it behind the scenes. If a fault actually leads to a system hanging or crashing, then you will know about it. This may reduce the number of things we need to worry about to the things that lead to silent data corruption. Because, if that is a relatively small number, and I can figure out the other ones when they occur, maybe I don't need to worry about the small subset of faults.

Mukherjee: A system hang is not necessarily a detected error; it can cause silent data corruptions.

Audience member: In Scott's [Mahlke] presentation, he used the *z* series from IBM as an example, but these are systems that are all about the reliability, and they are enormously internally redundant and fault tolerant. So, when you talked about the low

error rate, that is the low error rate after all the expense devoted to reliability and adding everything for reliability. It would be very interesting to understand the non-z series experience that people have.

Mahlke: The errors that I mentioned did not cause corruption, but were actually parity errors that were caught and corrected in their system. These weren't the errors that got through all the armor that they have put up.

Mukherjee: I have an example of that, from a recently published paper from Los Alamos National Lab.⁷ There is a system of 2048 HP AlphaServer ES45s, where the MTTF is quite small. It is proven that cosmic-ray-induced neutrons are the primary cause of the BTAG (board-level cache tag) parity errors that are causing the machines to fail.

Will classical solutions be enough?

Audience member: I think one thing that both of you agreed on is that reliability has a cost, either in area or in performance. You may not see the errors, because the system is over-dimensioned. Perhaps, we are not doing our jobs as microarchitects to actually look at the trade-offs between performance and errors that we are able to tolerate, or to look at dimensioning the system to tolerate more errors, and perhaps to make the system cheaper using cheaper materials or architectures. Perhaps the tragedy here is that a lot of these trade-offs at this point are done at the semiconductor level rather than at microarchitecture level. But, in the future it may be done at the microarchitecture level, where you can compute the trade-offs between performance versus reliability versus cost.

Mahlke: One of the problems is that as you go towards more multipurpose systems, these systems tend to do both critical and noncritical things. You might have different trade-offs for reliability versus performance for different tasks. For instance, for a video encoder, the system requires maximum performance and lower reliability. Therefore, as we go towards less-programmable systems, the trade-off is more obvious. On the other hand, as we go towards more-

programmable systems doing both critical and noncritical tasks, the trade-off becomes a little bit harder, or a little bit foggy, with these multipurpose systems.

González: Do you have a good example of a potential area where you believe that current approaches—for example, ECC—won't be enough? Do you have a good example to motivate what can be done at the microarchitecture level, Shubu?

Mukherjee: If you look at logic gates today, their contribution to error rate is hidden in the noise of all the factors that cause the errors, such as soft errors and cell instability. But, if you look five to 10 years ahead—once timing problems start to show up, maybe due to variations or wear-outs—logic is going to become a problem. So, in that case, classical ECC is not going to buy you anything. We can start looking at residue-checking or parity-prediction circuits to detect logic errors.

It also comes down to this fundamental point that you have a full stack, starting from the software all the way down to the process. As you go up from one layer to another, the definition of errors makes it clearer whether it is an error or not. That's what you need to track and what you need to expose. That's what Joel Emer and I have worked on for a long time. We are convinced that if you look at different levels of a system, the resilience is very different. In some cases, if you hit the bit, you immediately see the error, but sometimes you don't see it at all. There is a wide degree of variability.

Measuring reliability

Audience member: Can you give us an idea of how much reliability we gain by what you are putting in the processor, compared to when we have nothing?

Mukherjee: I have some data on a cancelled processor project. I was the lead architect for reliability of that processor. Our data showed that if you didn't have any protection, that chip would be failing in months due to all kinds of reliability issues.

Audience member: How much can we put in the commodity processors that customers are willing to pay extra pennies for?

Mukherjee: That goes to the fundamental problem of how to let customers know what they are getting for the extra price. The problem is that if you look at soft errors, we cannot tell them what extra benefit they are going to get. We can measure the performance by clock time, but we don't have a good measurement of a system's reliability.

González: Any idea on how we can measure reliability?

Mukherjee: We fundamentally need a mechanism to measure these things. For hard errors, the problem may be tractable. For soft errors—induced by radiation—this is still a hard problem. For gradual errors, such as wear-out, we still don't know how to measure the reliability of an individual part. So, the answer is that, in many cases, we don't know how to measure reliability.

Mahlke: There may be a different angle of looking at how reliability can be measured. Instead of thinking of reliability as a tax that you have to pay, and trying to justify this tax, maybe the right way to go about this is thinking of what else we get in addition to reliability.

Let's take Razor as an example. Razor can identify events like transient faults, and it also allows you to drive voltage down and essentially operate at the lowest voltage margin possible. It identifies when the voltage goes too low and self-corrects the circuit. If we talk about adaptive systems and how to make systems more adaptable for reliability or power consumption, then it may be about justifying the cost of some feature that the user really wants, and reliability just kind of happens magically behind your back.

How much extra hardware is needed for reliability?

Audience member: How much can Intel afford to put in the chip for reliability?

Mukherjee: We will put as much as we need to hide under the software errors.

Mahlke: I guess you are saying that you are putting in too much, since the software errors are two orders of magnitude greater than the hardware errors.

Mukherjee: Microsoft has actually shown that Windows causes very few of the problems. It is the device drivers that cause many of the problems. Memory is also a big problem, since more than 90 percent of memories out there don't have any fault detection or error correction in them. Stratus is a company that actually builds fault-tolerant systems using Windows boxes running on Pentium 3s. How did they do that? They tested all the device drivers. And they don't let anyone install any device driver arbitrarily on those systems. So, they have a highly reliable, fault-tolerant Windows system running on Pentium 3s. Believe it or not, that exists. So, blaming Windows is not the right way. Microsoft has done a phenomenal job showing that it is not Windows itself that causes most of the reliability problems in today's computers. MICRO

Acknowledgments

All views expressed by Antonio González in this article are his alone, and all views expressed by Shubu Mukherjee are his alone. Neither author represents the position of Intel Corporation in any shape or form. Although Mahlke argues the fallacy viewpoint in this article, his research group actively works in the areas of designing reliable and adaptive computer systems.

References

1. S. Borkar, "Microarchitecture and Design Challenges for Gigascale Integration," keynote address, 37th Ann. IEEE/ACM Int'l Symp. Microarchitecture, 2004.
2. National Software Testing Labs, <http://www.nstl.com>.
3. R. Mariani, G. Boschi, and A. Ricca, "A System-Level Approach for Memory Robustness," *Proc. Int'l Conf. Memory Technology and Design (ICMTD 05)*, 2005; <http://www.icmtd.com/proceedings.htm>.
4. J. Srinivasan et al., "Lifetime Reliability: Toward an Architectural Solution," *IEEE Micro*, vol. 25, no. 3, May-June 2005, pp. 70-80.

5. Center for Advanced Life Cycle Engineering, Univ. of Maryland; <http://www.calce.umd.edu>.
6. D. Ernst et al., "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," *Proc. 36th Ann. Int'l Symp. Microarchitecture (MICRO 03)*, IEEE CS Press, 2003, pp. 7-18.
7. S.E. Michalak et al., "Predicting the Number of Fatal Soft Errors in Los Alamos National Laboratory's ASC Q Supercomputer," *IEEE Trans. Device and Materials Reliability*, vol. 5, no. 3, Sept. 2005, pp. 329-335.

Antonio González is the founding director of the Intel-UPC Barcelona Research Center and a professor of computer architecture at Universitat Politècnica de Catalunya. His research focuses on computer architecture, with particular emphasis on processor microarchitecture and code generation techniques. González is an associate editor of *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *ACM Transactions on Architecture and Code Optimization*, and *Journal of Embedded Computing*.

Scott Mahlke is an associate professor in the Electrical Engineering and Computer Science Department at the University of Michigan, where he directs the Compilers

Creating Custom Processors research group. His research interests include application-specific processors, high-level synthesis, compiler optimization, and computer architecture. Mahlke has a PhD from the University of Illinois, Urbana-Champaign. He is a member of the IEEE and ACM.

Shubu Mukherjee is a principal engineer and director of SPEARS (Simulation and Pathfinding of Efficient and Reliable Systems) at Intel. The SPEARS Group spearheads architectural innovation in the delivery of microprocessors and chipsets by building and supporting simulation and analytical models of performance, power, and reliability. Mukherjee has a PhD computer science from the University of Wisconsin-Madison.

The biographies of **Resit Sendag**, **Derek Chiou**, and **Joshua J. Yi** appear on p. 24.

Direct questions and comments about this article to Antonio González, Intel and UPC, c/Jordi Girona 29, Edifici Nexus II, 3a. planta, 08034 Barcelona, Spain; antonio.gonzalez@intel.com.

For more information on this or any other computing topic, please visit our Digital Library at <http://computer.org/csdl>.

Join the IEEE Computer Society online at

www.computer.org/join/



Complete the online application and get

- immediate online access to **Computer**
- a free e-mail alias — **you@computer.org**
- free access to 100 online books on technology topics
- free access to more than 100 distance learning course titles
- access to the IEEE Computer Society Digital Library for only \$118

Read about all the benefits of joining the Society at
www.computer.org/join/benefits.htm