Designing Chips without Guarantees

At the 2010 Design Automation Conference, a session titled ''Computing without Guarantees'' looked at how computing platforms that do not conform to guarantees might be the wave of the future. The process of electronic system design has traditionally conformed to an axiom: that the specification and implementation must be equivalent in a numerical or Boolean sense. A wide range of application domains, however, including digital signal processing, multimedia processing, and wireless communications, do not require such a strong notion of equivalence, owing to the presence of noise in the input data and the limited perceptual ability of humans consuming their output. Emerging workloads of the future, such as recognition, mining, and synthesis, take this ''inherent resilience'' to a different level due to the massive amounts of data they process, statistical nature of the algorithms, and built-in expectation of less than perfect results. Several recent research efforts attempt to exploit this inherent resilience of algorithms to obtain unprecedented levels of performance or energy efficiency in hardware and software implementations.

IEEE Design & Test thanks roundtable participants Mel Breuer (Univ. of Southern California), Srimat Chakradhar (NEC Labs), William Joyner (SRC [Semiconductor Research Corp.]), Rakesh Kumar (Univ. of Illinois), Anand Raghunathan (Purdue Univ.), and Naresh Shanbhag (Univ. of Illinois). *D&T* gratefully acknowledges the help of Fadi Kurdahi (Univ. of California, Irvine), our moderator; Joerg Henkel, the DAC session organizer; and David Yeh (SRC), our Roundtables Editor, who organized the event. Special thanks go to the IEEE Computer Society for sponsoring the roundtable.

D&T: For the past several decades, the prime directive in chip design has been: "Thou shalt conform 100 percent to specs and be 100 percent error-free." What is driving the change to a paradigm of "computing without guarantees" today?

Shanbhag: The term "computing without guarantees" is not precisely right. We are computing with guarantees at the system level, but here we're talking about computing on the circuit fabric, and the fabric's reliability isn't perfect. The move toward computing without guarantees has occurred simply because of the cost involved. Worst-case design is expensive, yet that's been the mode of operation for many years. It's simply no longer feasible for us to design systems where the circuit fabric can be guaranteed to be perfect.

Breuer: Basically, I agree; but the term I use as the reason for what's driving the change is *yield*. It's harder to guarantee high yield with these nanotechnology fabrics. If we don't extend to deep nanotechnology, then circuit reliability can remain fairly high. But as we get to quantum levels, the fabric becomes statistical in nature, and it's hard to guarantee what's going to happen. We either have to say, "I'm not going to use these fabrics for certain applications," or say, "I will use them in applications where I can tolerate noise, or certain errors." We're not guaranteeing deterministic results the way we currently do, but we will guarantee a certain level of performance or that we will work within some tolerance for error. We can say that an answer will be correct within an error tolerance of .01 and with a certainty of .99.

Chakradhar: Multiple factors are driving the change to computing without guarantees—three come to mind. One is deep scaling. We cannot produce working devices completely, so we don't have a choice but to see what we can do with partially working or nonfunctional devices.

The second driver, I think, is system scalability. We're building systems that will be used by millions, and we're beginning to see that it's very hard to

0740-7475/10/$26.00 © 2010 IEEE Copublished by the IEEE CS and the IEEE CASS

build them if we have to stick to traditional guarantees. The third factor is applications. Applications are emerging that can tolerate a lot of errors; they are quite forgiving. By and large, I would say the key factor driving the change to computing without guarantees is applications. If the applications weren't so error-tolerant, we'd be forced to do a lot of gimmicks at the lower level of design.

D&T: The second question for this roundtable concerns the scope of this paradigm. Where and when can we afford to compute without guarantees? And how application-dependent are these approaches? Are we going to see case studies or an application-agnostic methodology to implement computing-without-guarantee paradigms?

Shanbhag: Many of the applications that we talk about have been around for quite some time. Video compression, speech compression, media processing—these have to be considered traditional; they've always been forgiving. It's just that now we exploit the forgiving nature of these applications for tolerating errors in the hardware. The question is, where can these kinds of approaches be applied? I would say in all applications where either media-rich data has to be processed or some sort of decision making is needed: for example, the detection of a feature in an image database or the detection of a biological anomaly in a database of ECG waveforms. It's these predominantly biomedical, energy-related, and media processing applications that will drive the move toward computing without guarantees.

Breuer: Multimedia certainly will be part of that drive. Just watch what young people, specifically, are doing—they're downloading pictures and movies, and they're constantly in communication. No one's walking around anymore talking to somebody in person. They're talking to *somebody*, but it's always over some electronic media. So communication is ubiquitous, and it's going on constantly. There's so much noise in the system, which means we're dealing with information that just doesn't have to be precise. Dealing with imprecise information is one of the main applications of computing without guarantees.

Another consideration is the lifetime of a given unit. Right now, we have graceful degradation built into some products, which means that when things start wearing out, we turn off features. As we start learning more about these stochastic processes, we might be able to say, "Okay. This thing—after 10 years of operation, it's starting to deteriorate. What does the error look like; what is the error profile? Do I have to really downgrade it and turn that portion of the electronics off? Or can we live with this degree of noisiness?"

Chakradhar: Where, and when, can we afford to compute without guarantees? Almost any application where the results are going to be used by humans can, I think, be computed without guarantee. For example, we talked about multimedia: I fully agree that it's easy to see media as forgiving. But let's take an example of an application for cancer diagnosis. There's nothing more sacred than life, which is what cancer diagnosis involves. This application involves digital, high-resolution images that are analyzed by physicians. They look at these images and determine if the diagnosis is cancer or not. Even for an application like that, we were able to use best-effort computing techniques (as described in the DAC paper). We were able to do image segmentation, provide the results, and give it back to the physician or pathologist. He looks at it and is able to make the best effort diagnosis. There are many applications, even life-critical ones, where we can afford to compute without guarantees.

Another aspect of computing without guarantees is the notion of the guarantees themselves. Different situations will be able to tolerate a lack of different types of guarantees, such as loss of performance, loss of power, or loss of function. It may be hard to provide a universal guarantee—if there's really such a thing as a *guarantee* in real life—but I don't think this is how it will be. So, considering the variety of ways in which relaxed guarantees show up, and considering the fact that the end user of the computation is a person, there are many applications, going forward, that will fall into the compute-without-guarantees category.

Raghunathan: Guarantees can take different forms: functional, performance, power, reliability, and so on. In the broadest sense, computing without guarantees covers all of these areas. However, in looking at specifically *functional* guarantees—that's the part that holds the more radical ideas in the area—we should acknowledge that not all applications necessarily can or need to fit into this paradigm. We don't

want our bank's transactions server computing without guarantees, certainly. However, the analytics that the bank runs to decide which product to market to you, based on your behavior, certainly can use this approach. What makes it interesting is that the growth in demand for computing is coming mostly from the latter category. That's why this area is so promising.

Kumar: I believe that the scope for such a computing paradigm is potentially broad and will really be determined by our success at managing the power or the variability problem. Work that has been done recently seems to suggest that we may be able to tolerate errors even in applications that are traditionally not considered error-tolerant. So if there are more and more compelling reasons to move toward hardware that does not provide guarantees, I think we'll see an increasing amount of work, even at the application level, where there might be efforts trying to convert applications that currently aren't considered error-tolerant into forms that will be tolerant of the errors that hardware might produce.

D&T: Could you give an example?

Kumar: For example, we have been doing some work in our group that we call "application robustification." This is where we take an application such as sorting, where we want the final result to be fully and correctly sorted even in the presence of hardware errors. We can work this sort of routine into a stochastic optimization problem, and we show that as long as the mean of the numerical error is around zero, this iterative version of sorting converges, and we can reclaim the exact, fully sorted result despite errors. Although this is still preliminary work, it does encourage us to look into these avenues where we try to convert applications that are not traditionally error-tolerant into forms that are.

Joyner: I'd like to go back to the situation Mel was talking about—everybody walking around with cell phones and sometimes losing calls. That may have something to do more with communication without guarantees, but a lot of it is related to cost, right? It's very annoying when you drop your call—agreed. But if we told these students, "Sure, we can make this service much better, but your cell phone is going to cost three times as much," or "Your monthly bill is going to be three times as much," they'd be more forgiving. We're able to tolerate computing without guarantees, in some applications, because we don't want to pay for the higher level of service that does guarantee computing.

Breuer: Well, yes. The example that I used is kind of the reverse, in the sense that I can sell you a display for your laptop that I can guarantee has no bad pixels. But I'm going to charge you for it. But vendors don't even offer that, because they know people will tolerate two and three bad pixels. So they give you the $1,000 laptop, rather than the one that's $3,000, because of the pixels. As we increase yield by tolerating these things, I would assume that the cost of chips goes down. There's certainly a tradeoff. Of course, this paradigm should make things cheaper for the end user.

Chakradhar: The mistake we are all making is that we are treating every human being the same. We're looking at the computation structure. We're saying it's producing information with some amount of errors, with guarantees in the data. What we don't take into account is that different users are willing to forgive things at different rates. We have to take that into account, and the system must be able to adjust to that. With cell phones, we're dropping some frames; the voice quality is bad—yet students are adjusting. One way to look at this situation is: there are 6 billion human beings on earth, and they're all very different. What the electronics industry is trying to do is find the majority of them willing to tolerate these kinds of losses and design devices that meet their needs. Alternatively, we have to start taking into account users' individual quirks—we're all very different, and we forgive things at different levels. The computing infrastructure must be able to match with the users' requirements. Today, it's essentially "one size fits all" on computing devices. I think that's going to change.

Breuer: I disagree. Let's say we're buying a brand-new digital camera. When we go to buy it, we find that the available cameras are essentially all about quality and price. You can get half a million pixels, 2 million pixels, 10 million pixels. These are all good cameras; there are no defects in them. But the vendors are saying to the end user: "How much quality do you need in your picture?" The price of the camera correlates to the quality. For instance, I have

a nice Canon lens that costs more than most people's cameras, but that's because I need it. So vendors are already offering us products on the quality-versus-price scale. They offer a lot of different choices in items, and it always comes down to quality and quantity. There's one more dimension to the story, of course, which is the errors. The vendors haven't added errors into the price-quality trade-off yet.

Chakradhar: I don't think we disagree here. All we are saying is that we need computing systems that are able to adapt to the expectations of specific users, not to those of the general population. That's where the extra dimension of optimization is going to come. A one-size-fits-all computing fabric for everybody is not sustainable.

Breuer: The analog world has been doing this for a long time. When I was a young boy, you could go to the store, look at the audio receivers, and vendors would ask, "What frequency range do you want to carry—from 200 cycles up to 10,000? Or do you want to go down to 100 cycles, with the dB loss and all that?" So vendors essentially just said, "We have different qualities of receivers we could sell you." And it's just part of the analog field. Digital-systems vendors, for some reason, don't think that. They think 32 bits is 32 bits. It's going to be right, period. It's just a different design paradigm.

D&T: Clearly, we're following this computing-without-guarantees approach for a reason—to gain something. The question is: what do we expect to gain from this approach, and by how much? Each of you has a different perspective of the problem—I'd like to hear what it is.

Shanbhag: Let's look at applications where this particular approach would work very well, in terms of robustness—let's say I take a conventional design methodology and a conventional design and look at its energy efficiency and performance. If I compare it with a stochastic processor or a stochastic computation-based implementation, in the presence of errors, I could show an orders-of-magnitude improvement in robustness while accompanying it with a roughly 40 to 50 percent energy efficiency improvement. So we can put hard numbers on specific algorithmic kernels that can benefit from the application of these ideas today. In the future, the potential is even greater in new and emerging areas, such as pattern recognition. So far, we have looked at applications in image processing and data communications, and kernels such as PN code acquisition—motion estimation, and video decoding. The potential to apply the ideas in computing without guarantees is even greater in applications of pattern recognition, image recognition, and decision-making.

Breuer: Well, coming from a test background, what I hope we gain from the computing-without-guarantees approach is yield. Certainly, we can tolerate chips that have defects. Those defects produce some errors, and if the functional performance is still acceptable, then we don't really have to trash those chips. So I see computing without guarantees as a yield enhancement.

D&T: By how much? Do you have an estimate?

Breuer: I don't have the numbers here, off the top of my head, to say, "What does that do, in terms of yield enhancement?" But if we added all the other speculation that goes on—the power and everything else—a significant percent could be gained. For example, when I talk to companies they'll say, "If you can help my yield by one percent, it means millions of dollars." So I think the enhanced yield could be significant.

Chakradhar: What do we expect to gain, and how much, from this approach? First, this approach—best-effort computing or computation without guarantees—can be used at multiple levels. Today we see the use of it at the circuit and the architecture level. Soon we'll see it in software, at multiple levels. How much do we gain? It depends on at what level these abstractions are being done, and so far, we've seen orders-of-magnitude improvements. But that's the wrong question to ask. Let's look at networking. We don't ask the question today: "How much do we gain from IP protocol?" We don't ask that—in fact, what we could gain is limitless. We're saying, "Hey, it's up to you, the applications, how you organize yourself and get the performance out of this network system." The applications have shown a tremendous variety. Look at the applications that we're seeing—Skype, Twitter, Facebook, Google—all those are running on top of IP. And the performance is limitless.

The same thing will happen for the computing fabric as well. Today, because computation without

guarantees is new, we worry about how much improvement we'll see. But the real payoff will come when we standardize the infrastructure and tell people, "Look, this is what we can build—it's done. Now as far as applications go, let your ingenuity fly. Find out ways you can use this kind of infrastructure we've built." I say, when that happens, that will be the real takeoff point for the whole industry. I'm sure about that—the way we're seeing it in IP networking is how it will happen in computing everywhere.

Raghunathan: From an initial adoption point of view, we have a conceptual barrier that we're trying to break through here—that we're dropping functional guarantees. We're breaking a deeply entrenched axiom that people have come to accept, right or wrong. We need to have at least an initial quantum improvement for this to be adopted.

But such improvements are possible, especially if we take a cross-layer holistic approach to taking advantage of these relaxed criteria at different layers. For example, we've demonstrated that order-of-magnitude improvements can be achieved, either in performance or in energy. I don't think a 10 or 20 percent improvement is going to be enough to push adoption. From another perspective, if there's an unbearable pain in the design of a computing platform with unreliable devices, and there's just no other way to design it, that could be another driver for the initial adoption of this paradigm.

Shanbhag: I'd like to make one observation here regarding the potential benefit of computing without guarantees: we've heard a lot about exploiting the intrinsic resiliency of emerging applications, but that's only half of the story. The other half is about error compensation and error resilience through proactively introducing ideas such as stochastic computing—for example, those from communication-inspired design techniques. That's the second half of the computing-without-guarantees story, which we haven't talked about here. If we add these two parts together, we can obtain even more benefits. So far, most of the discussions have focused on just exploiting the intrinsic resilience of these new applications.

Kumar: We need to be careful, however: are we simply pushing the complexity from hardware to software? When we quantify the benefits from computing without guarantees, we should be very careful in how we estimate the benefits. For one thing, while we may see benefits in terms of hardware parameters when we're relaxing the guarantees, perhaps the software now has to do more work. And if the software cannot tolerate that situation seamlessly, then perhaps the overall system benefits may not be as much as we assume. So one of the challenges, going forward, would be to look at this problem in a vertical, holistic fashion, as opposed to looking at it piecemeal, on an individual-stack level.

D&T: How accepting do you think industry would be of this computing-without-guarantees paradigm? As an example, built-in self-test was a luxury not so long ago; now, it's indispensable. Do you see computing without guarantees as following the same path?

Shanbhag: Industry is already open to these ideas. In fact, at this year's International Solid State Circuits Conference [ISSCC], Intel presented a paper on an error-resilient process architecture. We know of a number of companies that are actively pursuing these approaches; they're very interested in looking at what's feasible, and in building resiliency into their chips and designs. What's holding the industry back is a guaranteed method of delivering high-quality products and getting comfortable with a statistical or a stochastic design paradigm. That's the cultural change associated with adopting computing without guarantees. The research community is also grappling with this change, but it will happen eventually.

Breuer: I'm very hopeful that it will happen, but my experience to date has been otherwise—I've found a lot of resistance to this idea. Computing without guarantees deals with a different model for binning things. Right now, people seem to accept the fact that we bin things on functionality and clock rate. This new approach has to deal with a new terminology that companies have to develop, and it is just not in their models. Their focus is, of course, on high yield, but it's also got to be on performance and reliability. The companies that we talk to want to make sure that the chips they sell are good, so they have very low defects or bad chips per million, and that they guarantee to work for 10 years or so. That's their focus. I'd like to see this approach be used more extensively, and I think it will be at some point.

Chakradhar: How accepting do you think industry would be of computing without guarantees? Computing without guarantees is a new technology. What did we do with another new technology—parallel computing? Today, and even in my kids' generation, the only form of computing we'll see is parallel computing. In fact, "parallel computing" has become a redundant term because the only way we compute today is parallel. Now how did it arrive at that point? Simple: parallel computing is the only approach we can take today to deliver performance. The computing industry must adjust to what we can build, and so they've adjusted by making the applications parallel. I clearly envision that the only form of computing my grandkids will know is computing without guarantees. Why? Two reasons. First, that's the only way we will know how to build large, high-performance systems 15, 20, or 25 years from now. And if that's the only way we're building, the industry will have to adjust, just as the industry is doing for parallel computing.

Second, I also see another strong force coming in. The killer applications that we are seeing today and the killer applications in social and collaborative computing that are expected in the near future—all of them are very forgiving, very tolerant. It's a great fit for building systems that compute without guarantees. The killer applications that we're concerned about—developers of those apps also want exactly that type of fabric. They're not looking for guarantees.

Kumar: What we currently call Moore's law has actually held for a long time, even before integrated circuits. Ray Kurzweil has shown a nice graph to demonstrate how the amount of computation per dollar has changed over time. Moore's law has been, essentially, a portion of that graph in the IC era. Now, every time that curve was threatened, we moved on to a new technology. So in my mind, the question is, whether we come up with a new technology that lets us ride Moore's law. Unless we come up with a different kind of technology that lets us do computing deterministically and still lets us ride this exponential growth, we have to figure out how to do computing with present devices, which we know have variability that we can't tame using older design techniques. In the latter case, the only way to move forward is, perhaps, through computing without guarantees. So, just as with multicore architectures, where the industry is accepting of the technology even as the entire software ecosystem needs to change, if computing without guarantees is the only way to tame the variability problem, industry might adopt it.

D&T: It's already hard enough to design chips with crisp specs. How do we change today's "exact" design technologies to accommodate this new paradigm? I'm talking about synthesis, verification, validation, and testing.

Shanbhag: If we look at modern-day CAD methodology, at the very top, the specifications are already statistical for most of these applications. The performance metrics are statistical. We talk about signal-to-noise ratio, bit-error rates, quality of the image that has been reconstructed, and so on. At the lowest level, when we look at the device fabrics—circuit fabrics—and the process parameters, they are also characterized in a statistical manner. So there are statistics at the top and at the bottom. Right in the middle, a switch occurs, going from a system or algorithm design to architectures, and this is where the philosophy of determinism gets hold of the designer. The challenge is then to convert the middle part—architecture, design, logic, and circuit design—into this statistical framework. The first step in achieving this change is to have statistical models of behavior available to the designer at various levels. These behavior models should emerge from the underlying circuit fabric—from device fabrics—and be propagated up through the design hierarchy. We might need to revisit and redefine some of the levels in the design hierarchy in order for this to occur. But that's the first, most important step: the availability of statistical models of behavior. It needs to be developed and made available for system designers, and then the rest of the methodology will follow.

Breuer: We can accommodate the computing-without-guarantees paradigm with two things. One is parallelism. From the case studies we've looked at, we see that highly parallel architectures are much more tolerant to faults and errors. The more parallel a unit is, the better—and the better the quality, the smaller the degradation in performance. Parallelism is more robust to defects, as we've seen. The second thing, as was just said, is that we have error measures: error rates, error threshold values. We're trying to figure out how to propagate error measures—either push forward to the outputs or push back to the

inputs. If someone says, "I can accept a certain error rate at the output of this module," what does that mean in terms of the input? We're trying to find these mappings, where we map output specifications back through the design, to ultimately be able to say, "This unit only has to be so good to produce results that are acceptable at the output." At the moment, we don't know how to do that very well, and we're looking at that problem.

Chakradhar: If it's already hard enough to design chips with crisp specs, you're asking how we change today's exact technologies and still design chips while allowing for computing without guarantees? I think the reason it's hard to design chips is that we're doing the wrong thing. Once we start designing hardware according to computing without guarantees, the actual design process becomes easier. If we look at the networking space, let's say, and drop the requirement of reliability, just like the Internet Protocol did, the complexity of networking software drops by 90 percent. It's a fact; it happened. Similarly, if we start designing chips—designing platforms that compute without guarantees—the design process will become much simpler and easier because we don't have to worry about guarantees. And there's another aspect: we design chips today using a lot of CAD tools, a lot of software. If we look within the CAD tools, what are they doing? They're using heuristics to help us compute with exact guarantees. So if we can tolerate errors, and if we produce a platform that computes without guarantees, these heuristics can also be very simple. In the process, we can probably eliminate 90 percent of the complexity we're seeing now in the software tools as well. Computing without guarantees is a blessing. It's going to dramatically simplify the design process and dramatically simplify the tools we use to design chips.

Raghunathan: It doesn't make sense to completely throw away well-established methodologies. We, the researchers, must look for approaches that can use this paradigm without adding significant design complexity. A good example is the parallel programming model we've talked about. There would be minimal additional programming complexity if the software developers, in this case, can specify their algorithms while naturally exposing the parts of their applications that can tolerate errors or a loss of guarantees. We're taking the effort that is applied to timing and power closure, and signal integrity, and transferring some of that into a different way of thinking about the specifications. So, I second the opinion that if it's done correctly, computing without guarantees can actually reduce, not increase, design complexity.

Kumar: I agree with Naresh that we need to carefully figure out what the mapping is between the error rate that a design produces and the output quality, and bring that notion into the design flow. Current design methodologies are agnostic to the workloads. It's unfortunate, because when architects build chips, they optimize the processors based on the expected workloads that are supposed to run on them. When EDA designers build EDA tools, they rely on methodologies such as static analysis that are workload-agnostic. Future EDA tools need to be workload-aware in order to optimize a design to be "acceptable" for common workloads. I also think that initial advances to design flow should be minimally disruptive to the existing flow. We know it can be done. In fact, we just presented a paper at DAC 2010 where our CAD flow that uses standard EDA tools takes a target error rate as input, and it moves away from static analysis and instead relies on the frequency of execution of different paths by different workloads, and uses that information to allow errors. So we know it can be done. That understanding of how different design optimizations might impact the reliability behavior of workloads needs to be brought into the community at large.

D&T: If you look out five years from now, what would a SoC look like, vis-à-vis computing without guarantees? How different will it be from today's SoCs?

Raghunathan: Since we're talking about five years from now, I don't think the control processors in an SoC will use this model of computation. It's going to take more time for these ideas to pervade the entire SoC. I believe the application-specific accelerators would be the first to adopt this approach, just because we can vertically guarantee the end-user experience, probably followed by the many-core domain-specific processing units. For example, the graphics arrays or maybe the recognition and mining applications using many core arrays. Domain-specific arrays would be the next application to use computation

About the participants

Mel Breuer is a professor and the Charles Lee Powell Chair in Electrical Engineering and Computer Science, University of Southern California.

Srimat Chakradhar is head of the Dept. of System Design, NEC Laboratories, Princeton, N.J.

Joerg Henkel is a professor of computer science at Karlsruhe Institute of Technology, Germany.

William Joyner is director of Computer-Aided Design and Test at SRC, Research Triangle Park, No. Carolina.

Rakesh Kumar is an assistant professor in the Dept. of Electrical and Computer Engineering, and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.

Fadi J. Kurdahi, our moderator, is a professor of electrical engineering and computer science at the University of California, Irvine.

Anand Raghunathan is a professor of electrical and computer engineering at Purdue University, West Lafayette, Ind.

Naresh Shanbhag is a professor in the Dept. of Electrical and Computer Engineering, and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.

David Yeh is director of Integrated Circuit and Systems Sciences at SRC, Research Triangle Park, No. Carolina.

without guarantees. In the next five years, that's probably what we'll see. Looking even further beyond, there's certainly a possibility of this approach being adopted more pervasively.

Chakradhar: The SoC will have accelerators and components that do best-effort computing. But let's not forget: applications don't use chips, applications use systems. So we need chips, as well as software that runs on the chips, which will help the application developers. What we need is a combined look at both the software infrastructure and the chips to work with it. More and more of the functionality burden will be shifted to the software. The SoC itself will become much simpler when we design it for computation without guarantees.

Shanbhag: I'll try to answer a slightly different version of this question in the following sense: how will a SoC look in the near future if computing without guarantees were to take hold? In the first place, I agree with both Anand and Srimat that accelerator cores—application-specific cores—are where these ideas can first be shown to provide benefits. And, therefore, communication SoCs where these application-specific cores are used would be the ideal platform for demonstrating these ideas. We can do that today. We've already shown the benefits for motion estimation cores, and for Viterbi decoder cores. We can build these accelerator cores today and show benefits of computing without guarantees. In the future, a collection of these cores on a SoC platform would be the right place for these ideas to show up.

Chakradhar: Let me also add that we have done work on GPUs as well—best-effort computing on GPUs. We're beginning to see that the techniques that worked well for us on a general-purpose multicore processor are the same techniques that work for the GPUs. The focus there was mechanisms: provide mechanisms for computation without guarantees, and build hardware that is very simple. Build software that's very simple and provides no guarantees—just like we have been doing for a network. Also, we must let the applications get more creative on how to use the infrastructure to enhance performance. We're seeing that in the networking space; we're seeing that in the storage space. Why treat computing any differently?

D&T: Thank you, all, very much.