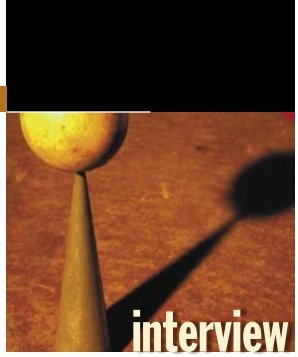


A Conversation with John Hennessy and David Patterson

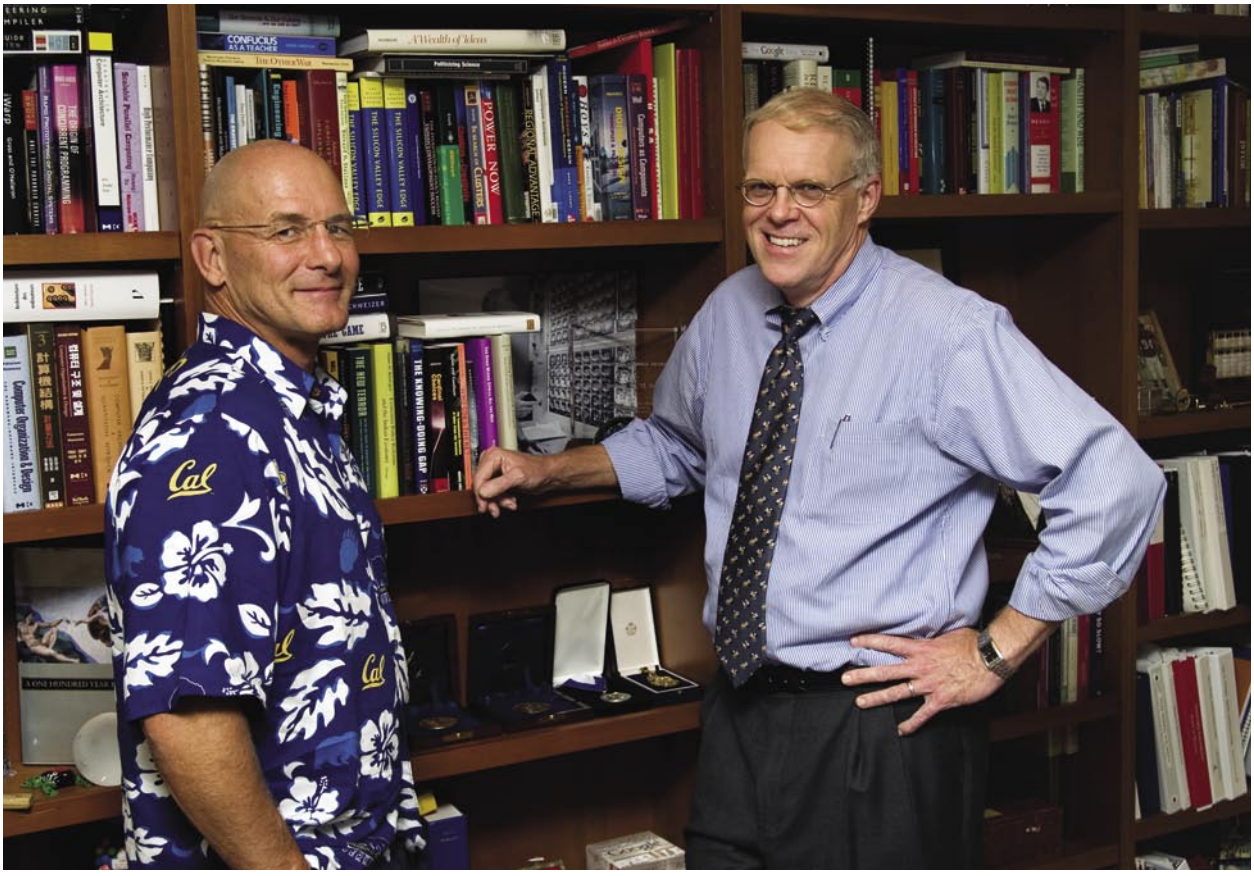
Photography by Jacob Leverich

They wrote the book ON COMPUTING

As authors of the seminal textbook, *Computer Architecture: A Quantitative Approach* (4th Edition, Morgan Kaufmann, 2006), John Hennessy and David Patterson probably don't need an introduction. You've probably read them in college or, if you were lucky enough, even attended one of their classes. Since rethinking, and then rewriting, the way computer architecture is taught, both have remained committed to educating a new generation of engineers with the skills to tackle today's tough problems in computer architecture, Patterson as a professor at Berkeley and Hennessy as a professor, dean, and now president of Stanford University.

In addition to teaching, both have made significant contributions to computer architecture research, most notably in the area of RISC (reduced instruction set computing). Patterson pioneered the RISC project at Berkeley, which produced research on which Sun's Sparc processors (and many others) would later be based. Meanwhile, Hennessy ran a similar RISC project at Stanford in the early 1980s called MIPS. Hennessy would later commercialize this research and found MIPS Computer Systems, whose RISC designs eventually made it into the popular game consoles of Sony and Nintendo.

Interviewing Hennessy and Patterson this month is Kunle Olukotun, associate professor of electrical engineering and computer science at Stanford University. Olukotun led the Stanford Hydra single-chip multiprocessor



research project, which pioneered multiple processors on a single silicon chip. Technology he helped develop and commercialize is now used in Sun Microsystems's Niagara line of multicore CPUs.

KUNLE OLUKOTUN I want to start by asking why you decided to write *Computer Architecture: A Quantitative Approach*.

DAVID PATTERSON Back in the 1980s, as RISC was just getting under way, I think John and I kept complaining to each other about the existing textbooks. I could see that I was going to become the chair of the computer science department, which I thought meant I wouldn't have any time. So we said, "It's now or never."

JOHN HENNESSY As we thought about the courses we were teaching in computer architecture—senior undergraduate and first-level graduate courses—we were very dissatisfied with what resources were out there. The common method of teaching a graduate-level, even an introductory graduate-level computer architecture course, was what we referred to as the supermarket approach.

The course would consist of selected readings—sometimes a book, but often selected readings. Many people used [Dan] Siewiorek, [Gordon] Bell, and [Allen] Newell (authors of *Computer Structures*, McGraw-Hill, 1982), which were essentially selected readings. Course curricula looked as though someone had gone down the aisle and picked one selection from each aisle, without any notion of integration of the material, without thinking about the objective, which in the end was to teach people how to design computers that would be faster or cheaper, and with better cost performance.

KO This quantitative approach has had a significant impact on the way that the industry has designed computers and especially the way that computer research has been done. Did you expect your textbook to have the wide impact that it had?

JH The publisher's initial calculation was that we needed to sell 7,000 copies just to break even, and they thought we had a good shot at getting to maybe 10,000 or 15,000. As it turned out, the first edition sold well over 25,000. We didn't expect that.

DP This was John's first book, but I had done several books before, none of which was in danger of making me money. So I had low expectations, but I think we were shooting for artistic success, and it turned out to be a commercial success as well.

JH The book captured a lot of attention both among academics using it in classroom settings and among practicing professionals in the field. Microsoft actually stocked it in its company store for employees. I think what also surprised us is how quickly it caught on internationally. We're now in at least eight languages.

DP I got a really great compliment the other day when I was giving a talk. Someone asked, "Are you related to *the* Patterson, of Patterson and Hennessy?" I said, "I'm pretty sure, yes, I am." But he says, "No, you're too young." So I guess the book has been around for a while.

JH Another thing I'd say about the book is that it wasn't until we started on it that I developed a solid and complete quantitative explanation of what had happened in the RISC developments. By using the CPI formula

Execution Time/Program = Instructions/Program x Clocks/Instruction x Time/Clock

we could show that there had been a real breakthrough in terms of instruction throughput, and that it overwhelmed any increase in instruction count.

With a quantitative approach, we should be able to explain such insights quantitatively. In doing so, it also became clear how to explain it to other people.

DP The subtitle, *Quantitative Approach*, was not just a casual additive. This was a turn away from, amazingly, people spending hundreds of millions of dollars on somebody's hunch of what a good instruction set would be—somebody's personal taste. Instead, there should be engineering and science behind what you put in and what you leave out. So, we worked on that title.

We didn't quite realize—although I had done books before—what we had set ourselves up for. We both took sabbaticals, and we said, "Well, how hard can it be? We can just use the lecture notes from our two courses." But, boy, then we had a long way to go.

JH We had to collect data. We had to run simulations. There was a lot of work to be done in that first book. In the more recent edition, the book has become sufficiently well known that we have been able to enlist other people to help us collect data and get numbers, but in the first one, we did most of the work ourselves.

DP We spent time at the DEC Western Research Lab, where we hid out three days a week to get together and talk. We would write in between, and then we would go there and spend a lot of time talking through the ideas.

We made a bunch of decisions that I think are unchanged in the fourth edition of the book. For example, an idea has to be in some commercial product before we put it into the book. There are thousands of ideas, so how do you pick? If no one has bothered to use it yet, then we'll wait till it gets used before we describe it.

KO Do you think that limits the forward-looking nature of the book?
DP I think it probably does, but on the other hand, we're less likely to put a bunch of stuff in that ends up being thrown away.
JH On balance, our approach has probably benefited us more often than it has hurt us. There are a lot of topics that became very faddish in the architecture research community but never really emerged.

For example, we didn't put trace caches in the book when they were first just an academic idea; by the time we did put them in, it was already clear that they were going to be of limited use, and we put in a small amount of coverage. That's a good example of not jumping the gun too early.
DP I think value prediction was another. There was tremendous excitement about its potential, and it ended up having limited applicability.
KO You delayed the third edition for Itanium, right?
JH I think our timing worked out right. It just goes to show the value of the quantitative approach. I think you can make a lot of pronouncements about an architecture, but when the rubber meets the road, does it perform or not?
DP One of the fallacies and pitfalls to consider is that you shouldn't be comparing your performance to computers of today, given Moore's law. You should be comparing yourself to performances at the time the computers come out. That relatively straightforward observation was apparently, to many people in marketing departments and to executives at computer companies, a surprising observation.

The Itanium was very late, which is one of its problems.
KO You've made a commitment to keeping the text up-to-date, so will there be a fifth edition?
DP It's actually a lot more than four editions. We originally wanted to write the book for graduate students, and then our publisher said, "You need to make this information available for undergraduates."
JH We thought somebody else would write an undergraduate textbook, but nobody did.
DP So we've now done three editions of the undergraduate book and four editions of the senior/graduate book.
JH What makes it so much work is that in each edition, 60 percent of the pages are essentially new. Something like 75 percent are substantially new, and 90 percent of them are touched in that process, not counting appendices. We replan each book every single time. It's not a small undertaking.

DP I'm pretty proud of this latest edition. We felt really good about the first edition, but then I think some of the editions just got really big. This one, we've put on a diet and tried to concentrate on what we think is the essence of what's going on, and moved the rest of the stuff into the CD and appendices.

KO How would you characterize the current state of computer architecture? Could you talk about the pace of innovation, compared with what it was in the past?
JH I think this is nothing less than a giant inflection point, if you look strictly from an architectural viewpoint—not a technology viewpoint. Gordon Bell has talked eloquently about defining computers in terms of what I might think of as technology-driven shifts. If you

look at architecture-driven shifts, then this is probably only the fourth. There's the first-generation electronic computers. Then I would put a sentinel at the IBM 360, which was really the beginning of the notion of an instruction-set architecture that was independent of implementation.

I would put another sentinel marking the beginning of the pipelining and instruction-level parallelism movement. Now we're into the explicit parallelism multiprocessor era, and this will dominate for the foreseeable future. I don't see any technology or architectural innovation on the horizon that might be competitive with this approach.

DP Back in the '80s, when computer science was just learning about silicon and architects were able to understand chip-level implementation and the instruction set, I think the graduate students at Berkeley, Stanford, and elsewhere could genuinely build a microprocessor that was faster than what Intel could make, and that was amazing.

Now, I think today this shift toward parallelism is being forced not by somebody with a great idea, but because we don't know how to build hardware the conventional way anymore. This is another brand-new opportunity for graduate students at Berkeley and Stanford and other schools to build a microprocessor that's genuinely better than what Intel can build. And once again, that is amazing.

JH In some ways it's déjà vu, much as the early RISC days relied on collaboration between compiler writers and architects and implementers and even operating-system people in the cases of commercial projects. It's the same thing today because this era demands a level of collaboration and cross-disciplinary problem solving and design. It's absolutely mandatory. The architects can't do it alone. Once ILP (instruction-level parallelism) got rolling, at least in the implicit ILP approaches, the architects could do most of the work. That's not going to be true going forward.

DP This parallelism challenge involves a much broader community, and we have to get into applications and language design, and maybe even numerical analysis, not just compilers and operating systems. God knows who should be sitting around the table—but it's a big table.

Architects can't do it by themselves, but I also think you can't do it without the architects.

KO One of the things that was nice about RISC is that with a bunch of graduate students, you could build a 30,000- or 40,000-transistor design, and that was it. You were done.

DP By the way, that was a lot of work back then. Computers were a lot slower!

JH We were working with hammers and chisels.

DP We were cutting Rubylith with X-acto knives, as I remember.

KO Absolutely. So today, if you really want to make an impact, it's very difficult to actually do VLSI (very large scale integration) design in an academic setting.

JH I don't know that that's so true. It may have gotten easier again. One could imagine designing some novel multiprocessor starting with a commercial core, assuming that commercial core has sufficient flexibility. You can't design something like a Pentium 4, however. It's completely out of the range of what's doable.

DP We recently painfully built a large microprocessor. At the ISCA (International Symposium on Computer Architecture) conference in 2005, a bunch of us were in the hallway talking about exactly this issue. How in the world are architects going to build things when it's so hard to build chips? We absolutely have to innovate, given what has happened in the industry and the potential of this switch to parallelism.

That led to a project involving 10 of us from several leading universities, including Berkeley, Carnegie-Mellon, MIT, Stanford, Texas, and Washington. The idea is to use FPGAs (field programmable gate arrays). The basic bet is that FPGAs are so large we could fit a lot of simple processors on an FPGA. If we just put, say, 50 of them together, we could build 1,000-processor systems from FPGAs.

FPGAs are close enough to the design effort of hardware, so the results are going to be pretty convincing. People will be able to innovate architecturally in this FPGA and will be able to demonstrate ideas well enough that we could change what industry wants to do.

We call this project Research Accelerator for Multiple Processors, or RAMP. There's a RAMP Web site (http://ramp.eecs.berkeley.edu).

KO Do you have industry partners?

DP Yes, we've got IBM, Sun, Xilinx, and Microsoft. Chuck Thacker, Technical Fellow at Microsoft, is getting Microsoft back into computer architecture, which is another reflection that architecture is exciting again. RAMP is one of his vehicles for doing architecture research.

JH I think it is time to try. There are challenges, clearly, but the biggest challenge *by far* is coming up with sufficiently new and novel approaches. Remember that this era is going to be about exploiting some sort of explicit parallelism, and if there's a problem that has confounded computer science for a long time, it is exactly that. Why did the ILP revolution take off so quickly? Because pro-

grammers didn't have to know about it. Well, here's an approach where I suspect any way you encode parallelism, even if you embed the parallelism in a programming language, programmers are going to have to be aware of it, and they're going to have to be aware that memory has a distributed model and synchronization is expensive and all these sorts of issues.

DP That's one of the reasons we're excited about what the actual RAMP vision is: Let's create this thing where the architects supply the logic design, and it's inexpensive and runs not as fast as the real chip but fast enough to run real software, so we can put it in everybody's hands and they can start getting experience with a 1,000-processor system or a lot bigger than you can buy from Intel. Not only will it enable research, it will enable teaching. We'll be able to take a RAMP design, put it in the classroom, and say, "OK, today it's a shared multiprocessor. Tomorrow it has transactional memory." The plus side with FPGAs is that if somebody comes up with a great idea, we don't have to wait four years for the chips to get built before we can start using it. We can FTP the designs overnight and start trying it out the next day.

KO I think FPGAs are going to enable some very interesting architecture projects.

DP Architecture is interesting again. From my perspective, parallelism is the biggest challenge since high-level programming languages. It's the biggest thing in 50 years because industry is betting its future that parallel programming will be useful.

Industry is building parallel hardware, assuming people can use it. And I think there's a chance they'll fail since the software is not necessarily in place. So this is a gigantic challenge facing the computer science community. If we miss this opportunity, it's going to be bad for the industry.

Imagine if processors stop getting faster, which is not impossible. Parallel programming has proven to be a really hard concept. Just because you need a solution doesn't mean you're going to find it.

JH If anything, a bit of self-reflection on what happened in the last decade shows that we—and I mean collectively the companies, research community, and government funders—became too seduced by the ease with which instruction-level parallelism was exploited, without thinking that the road had an ending. We got there very quickly—more quickly than I would have guessed—but now we haven't laid the groundwork. So I think Dave is right. There's a lot of work to do without great certainty that we will solve those problems in the near future.

KO One of the things that we had in the days when you were doing the RISC research was a lot of government funding for this work. Do we have the necessary resources to make parallelism what we know it has to be in order to keep computer performance going?

DP I'm worried about funding for the whole field. As ACM's president for two years, I spent a large fraction of my time commenting about the difficulties facing our field, given the drop in funding by certain five-letter government agencies. They just decided to invest it in little organizations like IBM and Sun Microsystems instead of the proven successful path of universities.

JH DARPA spent a lot of money pursuing parallel computing in the '90s. I have to say that they did help achieve some real advances. But when we start talking about parallelism and ease of use of truly parallel computers, we're talking about a problem that's as hard as any that computer science has faced. It's not going to be conquered unless the research program has a level of long-term commitment and has sufficiently significant segments of strategic funding to allow people to do large experiments and try ideas out.

DP For a researcher, this is an exciting time. There are huge opportunities. If you discover how to efficiently program a large number of processors, the world is going to beat a path to your door. It's not such an exciting time to be in industry, however, where you're betting the company's future that someone is going to come up with the solution.

KO Do you see closer industry/academic collaboration to solve this problem? These things wax and wane, but given the fact that industry needs new ideas, then clearly there's going to be more interest in academic research to try to figure out where to go next.

JH I would be panicked if I were in industry. Now I'm forced into an approach that I haven't laid the groundwork for, it requires a lot more software leverage than the previous approaches, and the microprocessor manufacturers don't control the software business, so you've got a very difficult situation.

It's far more important now to be engaging the universities and working on these problems than it was, let's say, helping find the next step in ILP. Unfortunately, we're not going to find a quick fix.

DP RAMP will help us get to the solution faster than without it, but it's not like next year when RAMP is available, we'll solve the problem six months later. This is going to take a while.

For RISC, the big controversy was whether or not to change the instruction set. Parallelism has changed the programming model. It's way beyond changing the instruction set. At Microsoft in 2005, if you said, "Hey, what do you guys think about parallel computers?" they would reply, "Who cares about parallel computers? We've had 15 or 20 years of doubling every 18 months. Get lost." You couldn't get anybody's attention inside Microsoft by saying that the future was parallelism.

In 2006, everybody at Microsoft is talking about parallelism. Five years ago, if you had this breakthrough idea in parallelism, industry would show you the door. Now industry is highly motivated to listen to new ideas.

So they are a ready market, but I just don't think industry is set up to be a research funding agency. The one organization that might come to the rescue would be the SRC (Semiconductor Research Council), which is a government/semiconductor industry joint effort that funnels monies to some universities. That type of an organization is becoming aware of what's facing the microprocessor and, hence, semiconductor industry. They might be in position to fund some of these efforts.

KO There are many other issues beyond performance that could impact computer architecture. What ideas are there in the architecture realm, and what sort of impact are these other nonperformance metrics going to have on computing?

JH Well, power is easy. Power is performance. Completely interchangeable. How do you achieve a level of improved efficiency in the amount of power you use? If I can improve performance per watt, I can add more power and be assured of getting more performance.

DP It's something that has been ignored so far, at least in the data center.

JH I agree with that. What happened is we convinced ourselves that we were on a long-term road with respect to ILP that didn't have a conceivable end, ignoring the fact that with every step on the road we were achieving

lower levels of efficiency and hence bringing the end of that road closer and closer. Clearly, issues of reliability matter a lot, but as the work at Berkeley and other places has shown, it's a far more complicated metric than just looking at a simple notion of processor reliability.

DP Yes, I guess what you're saying is, performance per watt is still a quantitative and benchmarkable goal. Reliability is a lot harder. We haven't successfully figured out thus far how to insert bugs and things and see how things work. Now, that's something we talked about at Berkeley and never found a good vehicle for.

I'm personally enthusiastic about the popularity of virtual machines for a bunch of reasons. In fact, there's a new section on virtual machines in our latest book.

JH Whether it's reliability or security, encapsulation in some form prevents a failure from rippling across an entire system. In security, it's about containment. It's about ensuring that whenever or wherever attacks occur, they're confined to a relatively small area.

DP We could use virtual machines to do fault insertion. What we're doing right now at Berkeley is looking into using virtual machines to help deal with power. We're interested in Internet services. We know that with Internet services, the workload varies by time of day and day of the week. Our idea is when the load goes down, move the stuff off some of the machines and turn them off. When the load goes up, turn them on and move stuff to them, and we think there will be surprisingly substantial power savings with that simple policy.

KO People will come up with new ideas for programming parallel computers, but how will they know whether these ideas are better than the old ideas?

DP We always think of the quantitative approach as pertaining to hardware and software, but there are huge fractions of our respective campuses that do quantitative work all the time with human beings. There are even elaborate rules on human-subject experiments.

It would be new to us to do human-subject experiments on the ease of programming, but there is a large methodology that's popular on campuses that computer science uses only in HCI (human-computer interaction) studies. There are ways to do that kind of work. It will be different, but it's not unsolvable.

KO Would you advocate more research in this area of programmability?

DP Yes. I think if you look at the history of parallelism, computer architecture often comes up with the wild idea of how to get more peak performance out of a certain

fixed hardware budget. Then five or 10 years go by where a bunch of software people try to figure out how to make that thing programmable, and then we're off to the next architecture idea when the old one doesn't turn out. Maybe we should put some science behind this, trying to evaluate what worked and what didn't work before we go onto the next idea.

My guess is that's really the only way we're going to solve these problems; otherwise, it will just be that all of us will have a hunch about what's easier to program.

Even shared memory versus message passing—this is not a new trade-off. It has been around for 20 years. I'll bet all of us in this conversation have differing opinions about the best thing to do. How about some experiments to shed some light on what the trade-offs are in terms of ease of programming of these approaches, especially as we scale?

If we just keep arguing about it, it's possible it will never get solved; and if we don't solve it, we won't be able to rise up and meet this important challenge facing our field.

KO Looking back in history at the last big push in parallel computing, we see that we ended up with message passing as a de facto solution for developing parallel software. Are we in danger of that happening again? Will we end up with the lowest common denominator—whatever is easiest to do?

JH The fundamental problem is that we don't have a really great solution. Many of the early ideas were motivated by observations of what was easy to implement in the hardware rather than what was easy to use: how we're going to change our programming languages; what we can do in the architecture to mitigate the cost of various things, communication in particular, but synchronization as well.

Those are all open questions in my mind. We're really in the early stages of how we think about this. If it's the case that the amount of parallelism that programmers will have to deal with in the future will not be just two or four processors but tens or hundreds and thousands for some applications, then that's a very different world than where we are today.

DP On the other hand, there's exciting stuff happening in software right now. In the open source movement, there are highly productive programming environments that are getting invented at pretty high levels. Everybody's example is Ruby on Rails, a pretty different way to learn how to program. This is a brave new world where you can rapidly create an Internet service that is dealing with lots of users.

There is evidence of tremendous advancement in part of the programming community—not particularly the academic part. I don't know if academics are paying attention to this kind of work or not in the language community, but there's hope of very different ways of doing things than we've done in the past.

Is there some way we could leverage that kind of innovation in making it compatible with this parallel future that we're sure is out there? I don't know the answer to that, but I would say nothing is off the table. Any solution that works, we'll do it.

KO Given that you won't be able to buy a microprocessor with a single core in the near future, you might be optimistic that the proliferation of these multicore parallel architectures will enable the open source community to come up with something interesting. Is that likely?

DP Certainly. What I've been doing is to tell all my colleagues in theory and software, "Hey, the world has changed. The La-Z-Boy approach isn't going to work anymore. You can't just sit there, waiting for your single processor to get a lot faster and your software to get faster, and then you can add the feature sets. That era is over. If you want things to go faster, you're going to have to do parallel computing."

The open source community is a real nuts-and-bolts community. They need to get access to parallel machines to start innovating. One of our tenets at RAMP is that the software people don't do anything until the hardware shows up.

JH The real change that has occurred is the free software movement. If you have a really compelling idea, your ability to get to scale rapidly has been dramatically changed.

DP In the RAMP community, we've been thinking about how to put this in the hands of academics. Maybe we should be putting a big RAMP box out there on the Internet for the open source community, to let them play with a highly scalable processor and see what ideas they can come up with.

I guess that's the right question: What can we do to engage the open source community to get innovative people, such as the authors of Ruby on Rails and other innovative programming environments? The parallel solutions may not come from academia or from research labs as they did in the past. Q

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums