

PIC: [Programming Languages & Software Engineering](#)

Title: X10: Creating the next high-computing programming language

Summary: IBM computer scientist Vijay Saraswat talks about the challenges and opportunities stemming from the development of a new programming language that will run on powerful new hardware architectures.

Presenter: [Vijay Saraswat](#), Research Staff Member

Duration: 10 minutes, 40 seconds

Introduction

Programming languages are enduring animals. The best ones instruct computers to do complex computations until another language comes along that can handle more complexity, greater concurrency.

IBM computer scientist Vijay Saraswat is part of a team that is developing a programming language called X10 that is slated to work on a number of super-fast hardware architectures, including the IBM Power7 chip.

In this episode of Open Collaborative Research, Vijay gives an overview of the project.

We have a large project going on, IBM does. It's called the [PERCS](#) project, [Productive, Easy-to-use Reliable Computing Systems] and this is funded by [DARPA](#) [Defense Advanced Research Projects Agency]. The goal of this project is to produce by 2010 a machine which is capable of a petaflop computation. That means, ah, it's an incredible number - 10^{15} operations per second. It's an incredible amount of computation - but in such a way that lots of people can program these machines. In fact, we have proposed that the machine should be about ten times easier to program than a machine using tools available back in 2004.

So, IBM is building new hardware. In fact, it's based on what is called the Power7 chip. So, it's a new chip that we are designing. The architecture is capable of stacking up a large number of these chips to work within a single system. That system, from the pure hardware point of view, is capable of delivering this very large number of instructions per second.

Now, the question is, how do you write programs for such a machine?

We are designing a programming language - X10. So, application writers will be writing code in our language.

Our compiler will then translate those programs into something to run on this large system. But the programmer gets a much simpler abstraction, something much simpler to write to than the concrete details of these very large machines.

Today the way people write programs is using this thing called MPI [[Message Passing Interface](#)]. And it's very, very difficult to do that at a level where you've got, say, 100,000 hardware threads running. So, a tremendous number of processes running on top of such systems.

Our goal is to provide a programming framework that's much more flexible, to make it much easier to write different kinds of programs, which can run on these large machines.

So, what will win, we felt, back in 2004, what will win in the future, is a design that works not just on the PERCS style of machine, but on all of the different kinds of machines that we thought would come out. Like the multicore systems, which are now out there. Like systems with accelerators, such as the [Cell](#). The goal is: One programming model to rule them all, to use [a phrase from The Lord of the Rings](#). As far as the programmer is concerned, he does not have to think about this diversity of hardware architectures. So, we give him a simple illusion, and we deliver that illusion by providing implementations on these different architectures. That's the game plan.

γ

There's a tremendous amount of skepticism that one could get a new programming language out, because, how frequently do new languages that are successful come out? The last one was [Java](#), and that was around ten years ago. Before that it was another ten years since [C++](#) came out.

But the answer to that skepticism is that, if you look at the history of programming languages, new languages have typically succeeded when they are driven by some architectural evolution, some change in architecture. And we are seeing that now. Our hope is that we can ride this next wave of architectural evolution by designing a new programming language.

The second big question, of course, is, are our abstractions the right abstractions? Have we chosen the right design point?

You answer that by first implementing your system so that it delivers good performance on these varieties of architectures. And we are in the process of doing that.

And the second way you answer it is by writing code - by showing that it's easy to write programs in this language. If you can show [that] both it's easy and you can show that it runs fast, then you've got a win on your hands.

γ

We have actually designed X10 deliberately to be an evolutionary language, not a revolutionary language. We've designed it to be derivative from sequential [Java](#). So that brings us to the heart of the matter.

Java was - at least in my eyes, and think of me as an external reviewer of the language - was really organized around a particular view of sequential computation that has proven to be very successful, namely, around objects. And [James] [Gosling](#)'s sort of magic was to remove from C++ a whole bunch of things that were very difficult to use and really came in the way, and so substantially decreased productivity for the vast majority of programmers while providing a few skilled people with lots of tools. So, his slogan was, "[C++ without guns, knives and clubs.](#)" And that does make a lot of sense.

But then Gosling also took the step forward to add threads to the language - a particular way of doing concurrency. In retrospect, the simple thing to be said is that at that time, that was how one understood concurrency, was through the use of these big heavyweight threads. There simply wasn't the architectural background in which to understand that, look, this may not be the way to program a machine like the [Blue Gene](#), which has 100,000 processes, or a heterogeneous machine like the Cell, which has different computing elements with very different characteristics.

So, Java does really well at sequential computation, and I would say that its concurrency-related features are nowhere near as successful or as well-designed as its sequential features.

What we have done with X10 is to start with that successful sequential base and add a whole new set of concurrency control constructs that are in particular designed to let a programmer write code that runs on thousands and thousands of processes. In Java it's really difficult to do, so you write Java code to run in one single [Java Virtual Machine](#) instance, JVM instance, in one process. So in X10 you can write code that runs in thousands of processes.

γ

The DARPA initiative that ended up funding X10 through PERCS also ended up supporting two other language initiatives, one at Sun, and that's called the [Fortress](#) system. The design has been led by [Guy Steele](#). And another project called [Chapel](#) at Cray, Cray Computers.

We have sort of agreed on a number of common things, and you'll find them common to the three languages. There's also very different foci of these three languages. We have chosen to be, as I have said, much more incremental, much more evolutionary in nature, whereas for the Fortress guys, I'd say they have chosen to, sort of, tear up the sheet and start afresh. That's a very revolutionary design.

We are very confident about where we are with respect to our value-add versus cost-of-adoption trade-off. We think we add substantial value, but the cost-of-adoption for those who write Java code is substantially lower than if they were to go to a completely newly designed language.

Guy Steele is one of the authors of the Java language specification, though, as he says, he came to the language after the language design had been more or less concluded. Nevertheless, he's, I would say, one of our preeminent language designers, and he has had a long-standing interest in things like making it much easier to write programs that look like math and that directly reflect nice, clean mathematical abstractions.

I would say the things that motivate him and his team are somewhat different from those that motivated us.

For us, a central element was that IBM has many different concurrent architectures, and we need a common programming model to deal with these. So, concurrency and distribution are the key elements. And then when you do this for high-performance computing, arrays tend to play such a central role. We've also redesigned arrays in X10 over Java. That's what X10 sets out to do.

γ

We want to be seen as leaders in innovative architectures for solving problems that scale in the commercial space, in the high-performance computing space, and so on. We have to work with the industry to get new programming models in place which will let a large number of people use these machines effectively.

And by the way, others are coming out with new architectures and new designs, and they face the same problem. How will people program these?

As an industry, we face this common set of problems. Multicore: How do we exploit this - of really taking a mainstream programmer from the world of sequential programming to the world of parallel programming.

So, I think this work on X10 really helps us stay very much at the center of the tremendous amount of work going on now in tools, in programming models, in programming languages, and lets us establish a clear positioning with respect to cleanliness of programming models and integrating tools and languages together, with Eclipse on the one hand and with valid debugging tools on the other.

γ

It's clear that the high-performance computing space stands most easily to gain with going to these sorts of massively parallel systems.

Second, energy exploration. Oil exploration. There's a tremendous amount of engineering analysis that needs to be done. And again, that's related to the sort of workloads in high-performance computing. So they can stand to gain by working with these large numbers of computers.

Our financial industries. There's a tremendous amount of data that's moving around the Internet now that has to be processed in real time, and you generate information from it, which is the basis for real transactions. So, again, tremendous amounts of computing power can help there.

Within the next five or ten years, we will start to see a large number of people use these systems, and that is going to be just so exciting.

You've been listening to Open Collaborative Research, an IBM podcast. For more information about the X10 programming language, contact Vijay Saraswat at vsaraswa@us.ibm.com.

See also: <http://x10.sf.net>

Series editor: Barbara Finkelstein

Music: "Phat n Sazzy" by The Dejunair Project