

Title: How do we bridge the gap between hardware and software?

Summary: Computer scientists David Bacon and Rodric Rabbah talk about the Liquid Metal project at IBM Research.

Duration: 8 minutes, 40 seconds

Introduction:

As computer scientists know, there's a big cultural - and productivity - gap between hardware developers and software writers. In this episode of [Open Collaborative Research](#), [David Bacon](#) and [Rodric Rabbah](#) talk about [Liquid Metal](#), an exploratory programming languages project that will give software programmers a means to realize customized hardware design.

Presentation:

Hi. I'm David Bacon. I'm a research scientist here at IBM Watson Research Center. I work in all sorts of aspects of programming languages. And now in the [Liquid Metal](#) project, we're working on languages for reconfigurable hardware.

And I'm Rodric Rabbah, also a researcher here at IBM Research, and I work primarily on the Liquid Metal project from the language, the compiler and the run-time system.

BACON: Right now this gap between hardware and software is not at all permeable or flexible. The people who program

those two things use very different languages. They even come from really different backgrounds. Most software programming is done by people who have computer science backgrounds. Most hardware programming is done by people who have electrical engineering backgrounds. So, a big part of what we're trying to do is have one language that can be used across hardware and software.

The other thing that we're trying to do is make the kind of dynamicity that's available in software also available in hardware. So, when people are using [reconfigurable gate arrays](#), which can be reconfigured - typically they're reconfigured quite rarely - and we're looking at taking that hardware re-configurability and making it something that's kind of constantly going on in the system as it monitors itself and figures out how to self-optimize and so on.

We have this catchphrase where we say what we're trying to do is "JIT the hardware." JIT stands for [Just-in-Time compilation](#).

RABBAH: People who typically programmed software have until today largely thought in terms of sequential programming. There's a very simple model of computing that has been taught in schools primarily.

For people who work on the hardware side, so, the architects and the engineers, they learn hardware description languages, and the way they think about algorithms is in terms of architecture, which is in terms of parallelism.

You have this strict dichotomy there in terms of sequential programming for software, parallel programming for hardware. And one of the interesting things about this project is to sort of bring these two together in one unified language.

BACON: We've explicitly gone in this different direction for a couple of reasons. One is, you know, we think that multicore is easy for architects to do. Right? They're used to building processor cores and they can't keep making them faster. And so now they say, "Well, we, okay, we have a description of the processor core. Let's just make more of them." That's pretty simple to do. It doesn't really require much innovation. It puts a lot of burden on the software, but it's fundamentally just sort of the obvious thing to do.

Whereas, where we're coming from, our point of view is well, the world is changing radically. The way we make use of [the] silicon area of transistors is going to change a lot over the course of the next few years and there's no

inherent reason to suppose that multicore processors are the right way to make use of those transistors. And so by using reconfigurable hardware, it gives us the chance to explore the design space a lot more thoroughly and flexibly.

One of the things about reconfigurable gate arrays is that one way you can think of them is as a network of very, very, very simple computing cores, right, that just do, a couple of operations on a bit or something like that, or on a few bits at a time. And on the current generation of hardware, there's something like 200,000 computing elements.

So one thing this lets us do is, we're exploring, in effect, parallelism at the level of 200,000 compute elements, whereas people who are working on multicore are working with 64 or 128 computing elements. So, we're looking much more out to the future in the sense that we're dealing with much, much more parallelism.

The other aspect of using [FPGAs](#) is, because we're synthesizing hardware, there's nowhere for us to hide. We can't cheat. There's no virtual memory. There are all sorts of games that people typically play in software we can't do. And so we've deliberately chosen to attack the hardest problem. Because we think if we can do [general purpose](#)

[processors](#) and FPGAs, then when we start looking at things in the middle like the current style of multicore or cell processors or graphics processors, that they're only going to be easier than what we've had to do to work on FPGAs.

RABBAH: At face value, there [are] four technical aspects of the project. And it's really looking at virtually every layer of the software abstraction stack from algorithm all the way down to hardware.

So, there's a language effort. So, we have a new language called [LIME](#), which is short for Liquid Metal, that we're developing.

Then, one layer down below that is the compiler, or the compilation system. An important aspect of that work is something we call an [intermediate representation](#), and in particular in this project a spatial intermediate representation, which we sort of view as a way of expressing parallelism in an economical representation that the compiler, or lots of different software systems, can use to transform your input language down to efficient implementations.

One level below that is the run-time system or virtual machines. This is something similar to a [Java Virtual Machine](#) with a lot more complexity and a lot more

sophistication that looks at all the pieces of computation that you're doing, and then you can do various aspects of either optimizations or matching pieces of code to run on differently parts of a system that are best suited for it.

And then another aspect of that would be an architecture aspect. You know, our FPGAs, the way they're designed today - the right way going forward which is not necessarily part of the project that we've focused on or looked at in a lot of great detail - just because there's so much other work to do in the first four layers.

Since the project is involved - it involves itself in a lot of software engineering - we have to build large complex systems, we've tried to build on existing software tools. So, whether it's existing compilers or existing compiler generators or existing run-time systems in virtual machines, we've tried to take the state-of-the-art that suits our purposes and extend that in various ways.

We've picked open-source technology and we're building on that to generate an entire ecosystem that we can then put out into the community and allow people to use, whether it's in research or in academia for teaching, to where we have an entire ecosystem for looking at the entire stack related to these concepts of new kinds of architectures, in our case related to FPGA-based machines.

So we've produced something that people can actually play with, and that's something we've released to various universities that we're working with, including Berkeley, including the University of Michigan and Georgia Tech. Incidentally, these are places where we also recruited interns from or have the [OCR](#) with.

BACON: So, from my perspective right now, we're at a time in the technology that's analogous to where we were in, say, 1959 in compiling high-level languages into machine language. At that point everyone was still writing assembler. And today people are still developing hardware in languages that are very, very low-level.

I think we're at this cusp where we have the opportunity to really make a quantum leap in terms of the level at which people describe hardware and the flexibility with which they're able to use it. I'm very excited about that. And that's what the project's all about.

IBM. Open Collaborative Research.

Series producer: Barbara Finkelstein

Music: [Hot Swing by Kevin MacLeod](#)