



Announcer You're listening to the Sun Microsystem's Podcast Network. Welcome to another edition of *Innovating@Sun* with your host Hal Stern. Today's topic: *Fortress 1.0*. And now, here's Hal Stern.

Hal:

Hello. Welcome to another edition of Innovating at Sun. I'm your host, Hal Stern, Vice President of Global Systems Engineering, and with me today is Dr. Eric Allen, a principal investigator in the Programming Language Research Group at Sun Labs, and Eric is going to give us an update on where we are with the Fortress language. So, Eric, welcome to the show and tell us a little bit about your Fortress.

Dr. Allen:

Hi. So Fortress is a new programming language designed for high-performance computing but with high programmer productivity. So we've tried to bring in lots of new ideas in programming language design from the past decade and apply that to the specific domain of high-performance computing in an attempt to improve the productivity of programmers in that domain. So Fortress, it includes lots of innovative features such as syntax that's based very closely on mathematical notation which allows programs to look very much like the mathematics that they're implementing. There's also lots of support for parallelism, in fact, implicit parallelism at many levels. In fact, it's very difficult to even write a Fortress program that's not parallel in some sense. And we're expecting that that will be very useful for targeting multi-core architectures as well as super computers and everything in between.

Hal:

So you make reference here to both productivity as well as super computers. Fortress came out of some work we were doing in the DARPA high-productivity compute system, HPCS. Again, I think a lot of people look to the P there as performance, and this is really about making developers more productive with the systems that we see coming out in the next couple of years.

Dr. Allen:

That's right. Yes. One of the goals that DARPA had for HPCS was to decrease the amount of time it took between when a program was needed and when we actually came up with a solution that we could run. And they were interested not in just decreasing the amount of time the program took to run on the computer, but also decreasing the amount of time that it took programmers to write it. So that was very much in the minds of the Fortress designers from day one really and has carried through to this day even though it's not part of HPCS anymore.

Hal:

So you made a comment there about capturing mathematical notation and the ancient APL programming remembers, you know, programs that were simply arcane for the sake of being arcane. I would say right now we're seeing a huge increase in what I call the imperative programming development environments. People want to go do things in scripting languages because they're easy and they have relaxed typing systems, and it's very easy to go construct what you mean in terms of the way you're going to write the program. How much of that would you say that that sort of design influence shows up in the way you translate mathematical ideas into

Fortress codes?

Dr. Allen:

Ah, yeah. So a lot of it does. So the mathematical notation in Fortress really is designed to decrease the space between what the actual specification of a program looks like and what the corresponding code looks like. The hope there is that we'd have fewer errors if we do that. We've also brought in a lot of language features that have proven useful in scripting languages to make it easier just to write code quickly but without sacrificing sort of robustness that you get from static analysis and static type systems.

Hal:

So one of the features of Fortress is an emphasis on dynamic compilation, and I know some of the work that's gone on here is around how we are able to explicitly call out optimizations that we want the compiler to make. How does that show up? I mean, you actually write code and then you get out some other piece of code that you can't edit? Or is it something that you can easily write code, optimize it, fix the optimized code and get back something else you can go edit with your mathematical hat on?

Dr. Allen:

Yeah. One of the ideas behind Fortress is that as much of the language as possible is put into the libraries themselves. So, in fact, programmers, when writing libraries, have the opportunity to write transformations of code that will actually occur when the program is running. So potentially that could be used to do dynamic optimization that's specified in the libraries. But we're also thinking that there's going to be some optimizations below the levels of abstraction we want to expose the programmer to, and that would occur in the virtual machine. That sort of optimization, programmers wouldn't have any access to. Another thing that I think is going to be very important for optimizing Fortress compilers is to have static optimizations occurring. Unlike with Java where all the optimization is done dynamically, with Fortress, because we're not targeting embedded systems and network computing as much, we have the opportunity to do static optimizations before the program is run. Some optimizations are best done at that point.

Hal:

And again, whatever optimization you're going to do, then you can also go capture in your source form, so it's not something that's lost because someone had a clever observation. It actually becomes part of the source base you're working from.

Dr. Allen:

That's right, yes.

Hal:

So tell me a little bit about elements of dimension checking and essentially what the typing system looks like. I think that a lot of people who would look at Fortress perhaps being derived from Fortran may run away from the notion of something being strongly typed.

Dr. Allen:

Yeah. Fortress, it includes a lot of type-checking features that are designed to make it easier to get to scientific programs, right? Like for instance, you mentioned static checking of physical units and dimensions. So these are the sorts of types that scientific programmers actually are interested in. They're not dealing so much with recursively defined data structures and stuff like that. They're really dealing with numbers that need to be annotated with some meaning as to what the physical quantities are that they're computing with. So Fortress

supports that really well, but we've tried to do it in a way that the applications programmer, if he wants, can, for the most part, ignore the static type system. The static type system is there to provide you with very strong checking if you want it, and in the libraries, we expect that programmers are going to make a lot of use of that. But the end programmer doesn't really need to deal with it so much. And, in fact, a lot of the types that are used to check programs are automatically inferred by Fortress.

Hal:

And I think everybody remembers a situation just a few years ago with a space probe that had mixed up meters and feet, failed to have the physical dimension of type checking and ended becoming a dot on the surface of Mars as opposed to a broadcast from the surface of Mars.

Dr. Allen:

Yes, that's right. That cost tens of millions of dollars, yes, that one mistake.

Hal:

And it's really hard to ship patch tapes out there once things have already been launched. So one of the things I hear regularly from customers is they look at where the entire universe of processor design is going now, multi-core, multi-thread and sort of get a hand waving saying, you know, help us figure out how we're going to go take advantage of all this inherent parallelism. How do we write code that's able to utilize the much faster, much more internally explicit programming model that has been produced now, you know, processors like Niagara and Niagara 2, looking forward to where Sun – I would say Sun [...?...]all going with processor design. How much of that is captured in some of the design goals of Fortress?

Dr. Allen:

Oh, yes. A lot of that is captured into the design goals of Fortress. In fact, you can – we view Fortress really as our best attempt to design a programming model that meets the needs for the 21st century in terms of multi-core architectures, the sorts of parallelism that we're going to need to expose in order to make the best use of all this hardware. So we've built in a lot of implicit parallelisms for Fortress. So, you know, programmers are going to get parallelism in their code even when they're not aware of it in some cases. And the idea is just to make the learning curve easier for the sequential programmer to migrate to this parallel world.

Hal:

As you do that re-skilling, whether it's explicit or implicit taken care of by the tools, generally I think you run into issues of synchronization, consistencies or all the properties you worry about when you're trying to handle multiple threads of execution. How does that show up? Where is the interesting work being done in Fortress around making that easier for the developer who may not be familiar with handling multiple parallel threads of execution?

Dr. Allen:

Right. So the notion of transactions is built very much into the core of Fortress. That's really the key language feature that we provide in order to manage all these concurrent threads that are running in a program. And our experience has been that it's much easier to manage that concurrency if you're dealing at this higher level of just thinking in terms of what actions you want to occur autotically [?] in a program as opposed to the lower-level way of thinking where you're explicitly managing locks on various pieces of data. Of course, Sun is a leader in terms of implementing transactions, so we've leveraged that quite a bit in implementing Fortress.

Hal:

So you'd be able to take advantage of hardware support, OS level support, memory system support, whatever – there's quite a lot of research being done now, University of Wisconsin and other places, looking at how do you provide transactional support for software in the hardware. So you'll be able to take advantage of any of those mechanisms?

Dr. Allen:

That's right. Yeah. The idea is really to take a hybrid approach. We utilize hardware support for transactions when it's available and otherwise, fall back to software.

Hal:

How about the overall sense of the flavors of concurrency? I think there's a lot of development working so with a very pessimistic sense of concurrency control as you've always have to assume there's going to be a conflict or always have to assume that there will be contention for a lock versus, I would say, a very optimistic sense or one in which you simply go and resolve the conflicts if and when they occur. You detect and then go fix. What was the attitude going in with Fortress in terms of balancing out the needs for always being correct as opposed to having the developer go back and fix things versus going really fast and being very scalable?

Dr. Allen:

Right. We don't see those things as necessarily being in conflict so much. We tried to design the language in such a way that that conflict is minimized. One way that you can minimize that sort of conflict is to limit the amount of side effects that are actually occurring in the program. So there's lots of support in Fortress for going computations without actually using lots of side effects. When you have that, of course, you don't have as much contention just to begin with. That makes contention management much easier. We've tried to build in the notion of concurrency into the type system as closely as possible, so you get a lot of static checking when possible over, you know, this sort of concurrency that's going on in your program. To some extent though, what we're doing is an experiment. Nobody really knows what the best programming model is going to be for these new architectures that are coming out. And we have to do some investigation really, put something out there and see how it plays out, see what works, what doesn't and go from there.

Hal:

Ah. So you may have asked the – or lead to the next question. Clearly, a lot of the interest in languages these days is around things that have large communities of developers around them. People try something out. They will build libraries or build frameworks, publish those as well. How are you going to get this into the hands of developers?

Dr. Allen:

Ah, yes. So right now, Fortress is an open-source project and we're building a community as we go. We're already getting contributions from the outside. People are implementing things and just giving the code back to us because they're excited about the language itself. We're also starting to teach courses on Fortress and getting people involved that way. I think ultimately, that's going to be essential for getting lots of adoption of Fortress and lots of participation in its implementation. So we're trying to build a community, and that's very hard, but so far, we've had some success. People are giving us libraries even now. So I think that just by being open and transparent in terms of developing the language, taking feedback on language features as we get them from the community, that's the way that we're hoping to spur adoption.

Hal:

And just to be really clear, so Fortress is released. Open source available for someone who wants to go download it and start playing both with our contributions as well as those from the community?

Dr. Allen:

That's correct, yes.

Hal:

So clearly, Eric, you and the programming languages team have been thinking about how to make developers' lives easier and also how to take advantage of the sorts of compute services that you see being developed by companies like Sun and other hardware vendors. What do you think's coming next? What do you want developers to really go think about?

Dr. Allen:

Well, so what we think that Fortress is going to be great at is just writing code in all sorts of scientific domains that typically have been inaccessible to the average programmer. We're hoping that Fortress will help out with that. So we're hoping that people will start thinking about new libraries that they can write once Fortress is in their hands and coming up with those libraries and contributing them to the community.

Hal:

Great. And so really it's the work of the entire community, not just the people who create the language, but the people who go and do the applied engineering around this. Well, Eric, thanks for joining us today. And you've been listening to another edition of Innovating@Sun, and I'm your host, Hal Stern.

Announcer You've been listening to Innovating at Sun. Join us next time for the latest in innovation from Sun Microsystems. Only on the Sun Microsystem's Podcast Network.