High-performance computing programming with ease

June 17 2014

As high-performance computing (HPC) bends to the needs of "big data" applications, speed remains essential. But it's not only a question of how quickly one can compute problems, but how quickly one can program the complex applications that do so.

"In recent years, people have started to do many more sophisticated things with big data, like large-scale data analysis and large-scale optimization of portfolios," says Alan Edelman, a professor of applied mathematics who is affiliated with MIT's Computer Science and Artificial Intelligence Laboratory. "There's demand for everything from recognizing handwriting to automatically grading exams."

The challenge is that there are only so many programmers capable of such wizardry, and the programs are getting more and more complex and time-consuming to develop. "At HPC conferences, people tend to stand up and boast that they've written a program so it runs 10 or 20 times faster," Edelman says. "But it's the human time that in the end matters the most."

A few years ago, when an HPC startup Edelman was involved in—called Interactive Supercomputing—was acquired by Microsoft, he launched a new project with three others. The goal was to develop a new programming environment that was designed specifically for speed, but which would also reduce development time.

The group, which includes Jeff Bezanson, a PhD student at MIT, and

Stefan Karpinski and Viral Shah, both formerly at the University of California at Santa Barbara, had all tried MPI (message-passing interface), which was specifically targeted at parallel processing. But MPI was tough going even for top-level programmers. "When you program in MPI, you're so happy to have finished the job and gotten any kind of performance at all, you'll never tweak it or change it," Edelman says.

The group set out to develop a programming language that could match MPI's parallel-processing support, while generating code that ran as fast as C. The key point, however, was that it would need to be as easy to learn and use as Matlab, Mathematica, Maple, Python, and R. To encourage rapid development of the language, as well as enhance collaboration, the language would need to be open-source, like Python and R.

In 2012, the project released the results of its labor, called "Julia," under an MIT open-source license. Although it's still a work in progress, Julia has already met and far exceeded its requirements, Edelman says.

"Julia allows you to get in there and quickly develop something usable, and then modify the code in a very flexible way," Edelman says. "With Julia, we can play around with the code and improve it, and become very sophisticated very quickly. We're all superheroes now—we can do things we didn't even know we could do before."

On the surface, Julia is much like Matlab, and offers Lisp-like macros, making it easier for programmers to get started. It provides a zippy LLVM-based just-in-time compiler, distributed parallel execution, and high numerical accuracy. Julia also features a mathematical function library, most of which is written in Julia, as well as C and Fortran libraries.

But Julia differs significantly from Matlab and the other environments in ways that Edelman is only now beginning to understand. "It's one of those things where you just have to try it awhile," he says. "Once you get in there, you see it's like nothing you've ever seen before. With Julia, we're trying to change the way people solve a problem, almost by solving the problem without immediately trying to. It lets your program evolve to be the thing that you really imagined it to be, not just the first thing you wanted."

One innovation is Julia's concept of "multiple dispatch," which lets users define function behavior across combinations of argument types. This provides a dynamic type system broken down into types, enabling greater abstraction.

"Julia gives us the power of abstraction, which gives us performance, and allows us to deal with large data and create programs very quickly," says Edelman. "We sometimes have races between two equally good programmers, and the Julia programmer always wins."

Matlab and the other environments take previously written Fortran or C, or proprietary code, "and then glue it together with what I call bubble gum and paper clips," Edelman says. This offers the advantage of easy access to programs written in more difficult languages, but at a cost. "When you're ready to code yourself, you don't have the benefit of the Fortran or C speeds," he adds.

Julia, too, can integrate programs written in other languages. But "we also make it really easy to develop in Julia all the way down," Edelman explains. "With Julia, you don't face a big barrier when you need to get higher speeds. If you want to use other languages, it's fine, but if you want to do fancier things, the barrier to entry is much lower."

Edelman lives a "double life," he says. In addition to helping developing

Julia, writing HPC applications, and teaching MIT students, he's also a theoretical mathematician with a focus on random matrix theory. In this role, Edelman is also a consumer of HPC simulations written in Julia: As he puts it, "I eat my own dog food."

Edelman spends a lot of time running Monte Carlo simulations, in which he generates a lot of random instances, and then tries to "understand collectively what might happen," he explains. "I love using Julia for Monte Carlo because it lends itself to lots of parallelism. I can grab as many processors as I need. I can grab shared or distributed memory from different computers and put them altogether. When you use one processor, it's like having a magnifying glass, but with Julia I feel like I've got an electron microscope. For a little while nobody else had that and it was all mine. I loved that."

Open source helps kickstart global community

The experience of co-developing Julia has deepened Edelman's belief in the power of open-source software. Thanks to Julia's open-source licensing, as well as the enthusiasm it generates among HPC developers, collaboration has been heightened in both the development of the language and in working together on Julia programs.

"We have hundreds of developers all over the world collaborating on Julia," Edelman says. "It's not like in the old days, when I would recruit the best Ph.D. students I could find at MIT and put them on a project. With Julia, people are joining us from around the world, and doing great things."

The open-source licensing has helped to quickly build an "incredible worldwide community," which Edelman says is just as important as the software's technical capabilities. "People are collaborating at so many levels it's amazing," he says. "Julia is out there, so I don't even know

what's going to show up tomorrow morning. People will ask me if there's an optimization package of a certain kind for Julia, and I say, 'I guess not,' and then I wake up the next morning and somebody's just written one."

One key to accelerating the development of Julia was the decision to create a package manager that eases the development of add-ons. These include an IJulia app developed in conjunction with the IPython community that provides a browser-based graphical notebook interface.

As with most other programming languages, Julia lets you split a task up into different chunks. Julia is notable, however, for how easy it is to work on the same piece of software together, Edelman says. In one of his recent HPC classes at MIT, a student developed a project where one programmer could start developing Julia on one terminal, and let others start typing on the same code as well.

"All these students started typing together," Edelman says. "It was an experience I'd never seen before. It was a great party, and a lot of fun. It changes everything about developing software."

This story is republished courtesy of MIT News (web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

Provided by Massachusetts Institute of Technology

Citation: High-performance computing programming with ease (2014, June 17) retrieved 21 September 2024 from https://phys.org/news/2014-06-high-performance-ease.html