Where's the Real Bottleneck in Scientific Computing?

Scientists would do well to pick up some tools widely used in the software industry

Gregory V. Wilson

When I first started doing computational science in 1986, a new generation of fast, cheap chips had just ushered in the current era of low-cost supercomputers, in which multiple processors work in parallel on a single problem. Suddenly, it seemed as though everyone who took number crunching seriously was rewriting his or her software to take advantage of these new machines. Sure, it hurt—the compilers that translated programs to run on parallel computers were flaky, debugging tools were nonexistent, and thinking about how to solve problems in parallel was often like trying to solve a thousand crossword puzzles at once—but the potential payoff seemed enormous. Many investigators were positive that within a few years, computer modeling would let scientists investigate a whole range of phenomena that were too big, too small, too fast, too slow, too dangerous or too complicated to examine in the lab or to analyze with pencil and paper.

But by the mid-1990s, I had a nagging feeling that something was wrong. For every successful simulation of global climate, there were a dozen or more groups struggling just to get their program to run. Their work was never quite ready to showcase at conferences or on the cover of their local supercomputing center's newsletter. Many struggled on for months or years, tweaking and tinkering until their code did something more interesting than grinding to a halt or dividing by zero. For some reason, getting to computational heaven was taking a lot longer than expected.

I therefore started asking scientists how they wrote their programs. The answers were sobering. Whereas a few knew more than most of the commercial software developers I'd worked with, the overwhelming majority were still using ancient text editors like Vi and Notepad, sharing files with colleagues by emailing them around and testing by, well, actually, not testing their programs systematically at all.

I finally asked a friend who was pursuing a doctorate in particle physics why he insisted on doing everything the hard way. Why not use an integrated development environment with a symbolic debugger? Why not write unit tests? Why not use a version-control system? His answer was, "What's a version-control system?"

A version-control system, I explained, is a piece of software that monitors changes to files—programs, Web pages, grant proposals and pretty much anything else. It works like the "undo" button on your favorite editor: At any point, you can go back to an older version of the
Of Possible Interest

FEATURE ARTICLE: Is Beauty a Sign of Truth in Scientific Theories?

COLUMN: Black-Light Sandwich
COLUMN: Visualizing Data
COLUMN: Science as Play
COLUMN: Industrial Origami

Related Sigma Xi Links
Distinguished Lecturer: Diana Rhoten
Sigma Xi Expertise Database
Sigma Xi International Newsletter
Forum: Undergraduate Education Reform
International Resources

© Sigma Xi, The Scientific Research Society