Why Aren't Computers Better?

Hardware Keeps Getting Faster, But Software Limits Progress

Computers have certainly come a long way during the past twenty years. Microprocessors have become thousands of times faster, memories have become thousands of times larger, and programs have become vastly more complex. Yet applications aren't getting the traction that they should; very little of the improvements in speed and capacity typically translate into comparable improvements in usefulness.

There is an enormous gap between the potential of computing and what it delivers in practice. Computers are far too often annoying and aggravating and too rarely do everything we want them to do—and that they should, in theory, be able to do. The spectacular, continuing increases in semiconductor technology have given us a tool that is far outpacing our ability to use it optimally.

At the heart of computing's difficulties is the radically different nature of hardware and software. Hardware is based on underlying technologies that improve at impressive and consistent rates. In addition, most hardware is independent of a particular application: a microprocessor fetches and executes instructions, a memory stores data, a display presents colored dots. The pace of hardware improvements creates an expectation among computer users (and in the industry itself) that the value of the systems is increasing as rapidly. Unfortunately, this is far from true.

Software gains great benefit, to be sure, from the advances in hardware technology. But unlike hardware, there is nothing in software that gets better at a geometric rate. Programming tools improve, and faster computers make programmers more productive, but the rate of improvement is slow. Yet at the same time, the complexity of the tasks being performed increases boundlessly.

Software is so much more difficult than hardware because of its intimate connection with the application and the user. Writing great software requires a deep understanding of the tasks to be performed and of effective ways to interface with the user. It is not a technological problem, which makes it far less amenable to technology-powered boosts.

We may have reached the end of the line in terms of computer engineers designing great computer applications. Any good programmer, for example, can figure out how to write a spelling checker. To create a useful grammar checker, however, requires a deep understanding of natural language semantics—knowledge that programmers don't have. (What we have today in PC grammar checkers is an example of what you get with a programmer's view of grammar.) There are probably enough CPU cycles available to run a truly useful writing assistant, given the right software—but no one today seems able to create that software.

Software must also take the blame for the poor reliability of most computers. Software crashes are far more frequent than hardware failures. Ultimately, every crash has the same cause: a programmer made a mistake. As software becomes more and more complex, ferreting out all the mistakes becomes daunting. This is one fundamental weakness of computers that is hard to see a way past. As reliable and predictable as the hardware may be, the software will always wear the marks of creative human endeavor, which is often wonderful but never perfectly reliable.

Better-designed software should be able to reduce the waiting that is a big factor in making computers frustrating to use. Faster processors certainly help, but it is amazing how slow many functions still are. Any time the computer makes you wait more than a fraction of a second, it slows you down and makes the experience less productive and enjoyable.

Disk access is, in many cases, a bigger cause of delays than actual processing. To some degree, these delays are inherent in the disks, whose access times have not decreased dramatically. All too often, however, starting a program or loading a document takes far longer than the raw transfer time from the disk. Today's PC software systems just aren't very intelligent when it comes to handling storage and interleaving multiple tasks. As DRAM capacities of hundreds of megabytes become economical, it should be possible to hide most of the disk delays by preloading programs and data into DRAM and performing saves in the background—but more sophisticated software is needed.

For computing to approach its potential, we need software that is smarter, more automated, and less failure prone. The challenge is to bridge the gap between the powerful but general-purpose machines and the very specific things that each of us wants them to do. The solutions will come not from programmers but from experts in the application domains, armed with tools that enable them to create programs. What is needed, more than any technology development, is huge amounts of human creativity—and that will occur on a human time scale, not on technology's time scale.

The microprocessor industry, and Intel in particular, is thus held back by forces over which it has little control. Unfortunately, there is nothing on the horizon that will change the fact that software is improving at a much slower rate than is hardware. M

See www.MDRonline.com/slater/better *for more on this subject. I welcome your feedback at* mslater@mdr.zd.com.