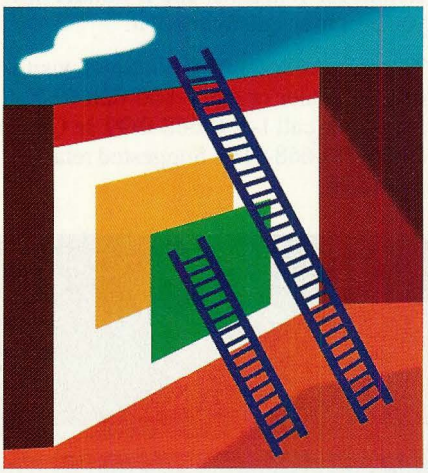
By Douglas A. Hamilton

Is 32-bits Really Twice as Good?



SCOTT BALDWIN

To listen to the marketing razzmatazz these days, you'd certainly think 32-bits is twice as fast as 16-bits. Personally, I believe that the world is moving to 32-bits and that we will see benefits emerging, but I'd also like to keep things in perspective.

My sense is that 32-bit code should produce performance benefits of about 30% or so in typical mainstream applications. But my rule of thumb is that in most interactive, single-user applications, you need roughly a 2:1 performance improvement before you can really count on most of your customers noticing.

An improvement of 30% is still worth doing. It's rare to find any single improvement that can yield 2:1, instead, you have to get there little by little, and every few percent helps.

In addition, there are clearly some problems where one would naturally like to work with very large objects. Trying to cram them into 64 kilobyte (KB) segments is a major pain. And there are specific recurring situations like sorting a list that's greater than 64KB that can turn up in almost any application area.

But these problems are not as pervasive as you might think. Most applications tend (not because of segmentation problems but because of other issues) to manage data in more "bushy" structures of smaller objects.

The argument that not having to deal with segment registers and so on simplifies application design even when large objects aren't involved is also a bit oversold. Modern compilers hide most of that nuisance.

In my own experience with the 85,000 lines in my C shell product, my estimate is that there are maybe a few hundred lines that I'd have saved (and now am able to save under NT and OS/2 2.0) with flat rather than segmented memory.

More to the point, who cares how difficult it is for the application vendor to program his or her product? Speaking as one of those vendors, even I have to admit: That's what he's paid for.

The question is not, is 32-bits better than 16 (only a moron would answer no), but one of perspective. How much better is it, and is it more important than other characteristics that mark our most modern platforms?

There's a myth in the industry that if you want things to run fast, what you need is small, tight assembly. Grotesquely, some vendors even feature that in their ads reminiscent of the old "hand crafted quality" campaign Zenith used 25 years ago after every other television set manufacturer went to printed

circuit boards and only Zenith was left still using a breadboarded chassis. Thankfully, the myth seems to be dying out, finally. Not because people have stopped believing it, but because vendors have found they can't control the quality of enormous assembly code projects.

If you want things to run really fast, trying to write everything in assembly is not usually the way to do it. In most projects, once you've reached somewhere around 1% to 3% assembly, you have all the benefit you're ever going to. Beyond that, the key is to re-examine the problem, looking for ways to avoid doing calculations for which you could already know the answer and to also be sure that the answers you need are right there at your fingertips when you need them.

My sense is that in most horizontal application areas, the more critical issue is not whether memory is flat vs. segmented, but whether there is lots of it. Computer scientists routinely talk about the time-space trade, the idea being that if you have more memory, you can use a larger, more complex algorithm or data structure and end up with a system that runs faster. Quite commonly as one is walking down through a logical path in an application, there may be occasions where it'd be inexpensive and convenient to calculate a value that might be useful later. If you wait until you know for sure you need that value, it'll be much more expensive. So, in a constrained memory situation, you shrug and admit, well, too bad, there just isn't room.

But memory is cheap, for example, because it's all virtual anyway, not only is the door open to higher performance designs, it's also open to better functionality. A designer doesn't have to ask which is better, working right or fitting in the space available. Under DOS, one always has to choose. Under OS/2 or NT or (to a lesser extent) under Windows, there's no need to choose, you can always have both.

The other architectural feature that I believe dwarfs flat memory in importance is concurrency. Most applications spend most of their time waiting. They wait for the disk, they wait for a key to be pressed, they wait for a printer. In short, they wait for everything and even worse, they serialize all these waits. The key idea behind concurrency features like multiple processes or multithreading is that while the machine is waiting for one "blocking" event, maybe there's something else it could be doing. That's important because most blocking events relate to things that happen at mechanical, not electronic speeds. Lots of machine cycles end up dribbling on the floor when you start waiting for mechanical events. So if an application that wastes 90% of the processor cycles waiting can be redesigned using threads to break up the problem into separable asynchronous activities, perhaps that waste could be reduced to, say, 10%. Suddenly, you have an application that runs nine times as fast.

Now, I admit, that's a contrived sort of example. Most real applications won't show that kind of improvement, but certainly you can achieve a doubling or more in performance.

Concurrency offers advantages even when the applications aren't so well-designed. Pre-emptive multitasking in OS/2 and NT allows background processes to run on spare cycles. Again, the point is to use resources that otherwise would go to waste. This is basic, good design. And between tight, even 32-bit code and good design, you can guess what I'd pick.

Again, just in case anyone missed it the first few times: Of course I think 32-bits is better than 16. Of course I think that's where we're going. Of course I, like everyone else, will be shipping 32-bit code. I think it'll lead to new functionality and better performance. But it pains me to see this one feature being latched onto as the only or even the most important issue in systems design.

So why is it the marketers are making such a big deal out of 32-bits particularly to the exclusion of anything else? I think it's really quite simple: 32-bits sounds twice as good as 16 and it's a heck of lot easier to explain. It's marketing, just like corn flakes. ■

Douglas Hamilton is president of Hamilton Laboratories in Wayland, Mass., and author of the Hamilton C Shell, an advanced interactive command processor and tools package for OS/2. Reach him on BIX as hamilton, on MCI Mail at 389-0321 or care of Editor at the address on page 10.