Wandering Precision

If you call the same numerical library routine twice with exactly the same input data, you expect to get identical results - don't you?

Well, it all depends. If you're calling a routine that somehow uses random numbers, for example a solver based on a monte carlo method, then the results might be different - but then you'd probably accept that (assuming that you'd been warned in the documentation).

But what if you're calling a numerical routine that in theory contains no randomness, but you still get slightly different results? And what if those slightly different results cause you problems - for example if you depend on repeatability for testing purposes? Then you might waste a lot of time hunting for the cause.

Well, this problem has turned up more than once recently for us at NAG, and in the cases we've looked at it has turned out to be a problem with the way the machine does its arithmetic - or rather, in the way that the compiler tells the machine to do its arithmetic.

Let me give you an example. Suppose you want to compute the inner product of two vectors x and y, each of length n. Simple - you just multiply each element of x by the corresponding element of y, then add all the products together to get the answer. No apparent randomness there. What could go wrong?

Well, if the hardware you are running on is equipped with Streaming SIMD Extension (SSE) instructions - and almost all modern hardware is - here's what.

SSE instructions enable low-level parallelism of floating-point arithmetic operations. For example, you can hold four single precision numbers at the same time in a 128-bit register, and operate on them all at the same time. This leads to massive time savings when working on large amounts of data.

But this may come at a price. Efficient use of SSE instructions can sometimes depend on exactly how the memory used to store vectors x and y is aligned. If it's aligned nicely - by which I mean, in the inner product example, that the addresses of the first elements of the arrays x and y are multiples of 16 bytes - then that's good. The hardware can efficiently move numbers from memory into registers to work on them, using instructions that depend on that alignment. So for our inner product, with a good optimizing compiler, we'd load numbers four at a time, multiply them together four at a time, and accumulate the results as we go along into our final result.

But if the memory is not nicely aligned - and there's a good chance it may not be - the compiler needs to generate a different code path to deal with the situation. Here the result will take longer to get because the numbers have to be accumulated one at a time. At run time, the code checks whether it can take the fast path or not, and works appropriately.

The problem is that by messing with the order of the accumulations, you are quite possibly changing the final result, simply due to rounding differences when working with finite precision computer arithmetic.

Instead of getting

```
s = x1*y1 + x2*y2 + x3*y3 + ...
```

you get

```
s = (x1*y1 + x5*y5 + x9*y9 + x13*y13) +
    (x2*y2 + x6*y6 + x10*y10 + x14*y14) + ...
```

Chances are that the result will be just as accurate either way - but it's different by a tiny amount. And if that tiny difference leads to a different decision made by the code that called the inner product routine, the difference can be magnified.

In some cases, for example when solving a mathematical optimization problem using a local minimizer, the final result can be completely different - and yet still valid - because a local minimizer might converge to any one of several different minimum points, if the function does not have a unique minimum.

And all because of how a piece of memory happens to be aligned. If you allocated the memory yourself you might be able to do something about it, but on the other hand the alignment might be completely outside your control. NAG users have reported cases where running a program from the command line would give consistently different results from running the same program by clicking on its name in Windows Explorer.

Is there anything we can do about problems like this? The fact is that NAG users want their code to run fast, so using SSE instructions makes sense. We can lobby the compiler writers to ask them to be careful how they optimize the code they produce, but they won't necessarily take notice. Compiler writers could also help by making their memory allocation routines only return 16-byte aligned memory, specifically to help with the use of SSE instructions. In the past, though, I had no success trying to convince the gfortran compiler folks to do that. In any case, even if they did, it wouldn't always help - if the first element of a single or double precision array is aligned on a 16 byte boundary, the second element will definitely not be, so if you want to operate on a vector starting at that location you've no chance.

We could choose not to use SSE instructions at all. But, apart from efficiency reasons, the "legacy" extended precision 80-bit register arithmetic which started in the 1980s with the 8087 numeric co-processor had its own "wobbling precision" problems for numerical code.

As new hardware with AVX instructions and 256 bit registers comes along, even more numerical work can be done in parallel. So - it seems that for the foreseeable future we're just going to have to live with this, and try to minimize the effects on NAG users by means of documentation.

LEARN MORE ABOUT THE NAG LIBRARY →

Author

Mick Pont (/people/mick-pont)

Leave a Comment

Your name

Your Comment

CAPTCHA

This form has an automated anti-spam system running (Recaptcha). If it suspects you are not a valid visitor a backup challenge will appear here.

SUBMIT →

Sign up for the NAG newsletter

SUBMIT →

WORLDWIDE LOCATIONS

(/CONTENT/WORLDW
CONTACT-
INFORMATION-0)

(https://github.com/numericalalgorithmsgroup)

ABOUT NAG (/CONTENT/ABOUT-NAG-0)

Blog (/content/nag-blog)
NAGnews (/content/nagnews-0)
Case Studies (/content/case-studies-0)
Contact us (/content/worldwide-contact-information)

SUPPORT (/CONTENT/TECHNICAL-SUPPORT-SERVICE-OVERVIEW)

Contact support (/content/technical-support-service-overview#contact)
Documentation (/content/software-documentation)
Installer's & Users' Notes (/content/installers-and-users-notes-nag-products)
Downloads (/content/software-downloads)
Technical Reports (/content/technical-report-repository)