Subject: Re: Locate an underflow
Posted by George N. White III on Fri, 25 May 2001 12:08:26 GMT
View Forum Message <> Reply to Message

On 23 May 2001, William Thompson wrote:

> Paul van Delst <paul.vandelst@noaa.gov> writes:
>
>> ... If, on running said code, I get a crapload of underflow errors,
>> it's an indication that that either a) the code hasn't been tested
>> very well or b) the programmer didn't really think about the problem
>> enough ...

Sound advice.

> I disagree. It's exceedingly easy to get underflow errors, and extremely
> difficult to program around them. For example, a simple Gaussian
>
> Y = A*exp(-((X-X0)/Sig)^2)
>
> is almost guaranteed to generate underflow errors. At some point this
> is going to be indistinguishable from zero. You'd have to jump
> through hoops to avoid getting the completely useless underfloat
> messages.

I do things like this every day, but only in the privacy of my own computer. I try to avoid it in code that other people will use or even code that I might want to use next year.

In the good old days underflow errors from things like the above example often were harmless, but:

a) you need to be confident that the program doesn't rely on the the assumption that exp(-x) is always positive, and

b) underflow can be very expensive on modern hardware. In cases where underflow doesn't alter the results of the calculation, you are often dealing with something like "a+c*exp(-x)" where "c*exp(-x)" is so small that the f.p. representation of a doesn't change for values of x much small than the values that produce f.p. underflow in exp(-x).

I've seen a factor of 10 speedup for a program running on SGI MIPS by replacing y=a+c*exp(-x) with a version that skips the evaluation of exp(-x) when c*exp(-x) is too small to affect the f.p. representation of a. For IA-64 the effect is likely to be much more severe because the f.p. exception handling loads a software f.p. emulator so the operation producing the exception can be replayed to gather data on

the cause of the exception.

Combined multiply-add instructions (y=a+b*x) aren't uncommon, and in the future you may well see single instructions for things like y=a+b*exp(x). If you get an exception in such an instruction, how do you determine which operation was responsible? Exception handling is an increasingly tricky thing to implement, and isn't much tested by current benchmark codes, so the chances are good that it won't work right on new hardware or after a software upgrade or just because the vendor changed the compiler flags governing exception handling and you didn't take time to study the release notes saying that exception handling only works for certain (low) optimization levels.

In most cases a simple range test will avoid underflow when evaluating a Gaussian. Not only does this mean that the program will have more predictable performance on a variety of hardware, it makes it much easier to examine the other situations where undeflow occurs to ensure that they are harmless.

--
George N. White III <gnw3@acm.org> Bedford Institute of Oceanography