Subject: Re: Wow. exp() difficulties...
Posted by William Clodius on Fri, 25 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

I have a few comments on Amara Graps post, mostly related to efficiency:

Amara Graps wrote:
>
> <snip>
>
> The primary advantages of using integer and fixed point numbers are that
> they take up less storage disk space and calculations with these numbers
> are fast.

Because of their simplicity it is in principle possible to make integer and fixed point number arithmetic faster than floating point arithmetic. However, in practice nowadays floating point arithmetic is as fast as integer arithmetic for addition and subtraction and often faster than integer arithmetic for multiplication and division. Both integer and floating point arithmetic use a lot of additions and multiplications, so processor designer emphasize making them go as fast as possible, typically a fixed multiple of processor clock speed. Integer arithmetic typically rarely uses multiplication or division so they are not highly optimized, while floating point arithmetic often uses multiplication and division and is often well optimized for those operations.

(A side note: while floating point arithmetic having performance comparable to integer arithmetic is generally a new phenomena, poor performance of integer multiplications and divisions is not new. There was a processor design in the early 60's, I believe by IBM, that had an integer division operation that was something like 1/10,000 the speed of other integer operations. Although integer division operations are rare, they are typically almost exclusively used to convert to and from decimal representations, they are used in more than 1/10,000 operations and the computer spent most of its time inefficiently computing integer divisions.)

> <snip>
>
> Double Precision Floating Numbers
> The advantages of using double precision number types are many. A double
> precision number usually yields fifteen or sixteen decimal places, which
> is more than adequate for most calculations. The largest normalized
> double precision number is about 1.798E+308. The smallest normalized
> number is about 2.225E-308. (Again put a negative sign in front to
> determine the largest and smallest *negative* number.) Even though the
> speed of performing double precision computations is twice as slow as
> performing single precision computations, the computation speed on

> today's computers is still fast.

In recent years this speed ratio is no longer valid. Some floating point processing units perform all their calculations in double or extended precision. As a result single precision can have the additional overhead of conversion to and from the double precision format. Whether single or double precision is more efficient in such cases depends on the magnitude of this overhead vs. increased cache misses due to the larger memory requirements for double.

> <snip>

--

William B. Clodius Phone: (505)-665-9370
Los Alamos Nat. Lab., NIS-2 FAX: (505)-667-3815
PO Box 1663, MS-C323 Group office: (505)-667-5776
Los Alamos, NM 87545 Email: wclodius@lanl.gov

Subject: Re: Wow. exp() difficulties...
Posted by agraps on Fri, 25 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

William Clodius <wclodius@lanl.gov> writes:

> lady of the elves wrote:
>>
>> I've encountered a really interesting--though annoying--problem. I'm
>> trying to use exp(a), where a>88 or so. My calculator can do it...the
>> program can't. It gives me a "floating underflow" error.
>>
>> Can anyone give me more information on how to work this out? I haven't
>> yet figured out long numbers...nor how they differ from floating point.
>> I don't know if these uncertainties relate, but all of my numbers are
>> floating point.

> Ouch!!

> While I know that a lack of knowledge about how computers do arithmetic
> is common, it pains me to see it displayed in a public forum for people
> who do a lot of computer arithmetic. You are not alone in your
> ignorance, and there are others that will benefit from what I am about
> to say, but realize that I am only going to give the most cursory
> overview and you should really read the manual and some texts on
> computer arithmetic.

[nice overview of computer arithmetic deleted]

Here is an article that I wrote 4 years ago for a newsletter for the scientists in my group. This article simplifies a number of issues and is a little bit dated and I think nothing tops reading the first chapter in a numerical analysis text to learn the subject. But maybe this article would be useful anyway. You can also retrieve it from http://www.amara.com/ftpstuff/realnum.txt

Amara

Real Number Computations

Representation of Numbers in a Computer
Numerical methods use numbers to simulate mathematical processes, which, in turn, simulate real world situations. The mathematician's real number system, however, has many peculiar features when implemented in a computer. These peculiar features have led many a researcher astray. In my article this time, I discuss how real numbers are represented in a computer, and calculations for which you should be wary.

Three systems of numbers are represented in the average computer: 1) Integers, 2) Fixed Point Numbers, and 3) Floating Point Numbers. Integers are the counting numbers: -1, 0, 1, 2, etc. Fixed point numbers are the familiar numbers of hand calculators such as 3.14159265, .01234567, etc. Floating numbers are closely related to scientific notation such as .3141592E+01, .1234567E-1, etc.

The primary advantages of using integer and fixed point numbers are that they take up less storage disk space and calculations with these numbers are fast. However their disadvantages are that their maximum data range (2-byte integers can usually be only between -32768 and +32767) is easily exceeded and computations with them (especially integers) can introduce significant errors. These errors, called truncation errors, are significant if your computation involves many iterations. Another caveat of working with integers is that byte-order (i.e. when the most significant and the least significant bytes are ordered differently) often makes it difficult to transfer integer files between computers. And, for fixed numbers and integers both, you must always be aware of their internal computer representation of the number offsets amd multipliers.

Because of the above disadvantages of working with integers and fixed point numbers, floating numbers are the preferred choice of scientific programmers for their computations.

Floating Numbers

Floating numbers are designed to handle very large and very small numbers. In the base 10 number: 9.10956E-28, the first block of seven digits is called the mantissa, and the last digits are called the exponent (for single precision, this value often ranges from -38 to +38). The sign of this number is +1, which is the third piece of information stored for this number. However, this number is stored in the computer as a *binary* number, which is equal to 1.1277089*2^(-90), and each of the mantissa, exponent, and sign are stored in their binary forms.

Let's consider the fractional number 4.537. It can be represented exactly as a decimal fractional number. However, it cannot be represented exactly in binary form, which is how your computer will store it. You may get 4.53699 instead.

Most computers have at least two types of floating numbers: single precision and double precision. Double precision numbers have approximately twice as many digits for the mantissa, and store several more bits for the exponent. The chart below illustrates the number of bits for the IEEE Standard of single and double precision numbers:

	Sign	Exponent	Mantissa
Single Precision	1	8	23
Double Precision	1	11	52

Single Precision Floating Numbers

Scientific programmers normally never need to care about how floating numbers are stored. What they should be concerned with, however, is the *precision* and *range* of their floating point numbers. For a single precision floating number, the largest possible value is 3.403E+38 and the smallest positive number is 1.175E-38. (These values are the result of considering the largest/smallest exponent and the largest/smallest mantissa. Put a negative sign in front of these values to determine the largest and smallest *negative* numbers.)

This range is sometimes not adequate for the scientific programmer. Our galaxy contains about 1E+44 grams of material. The electron charge to the fourth power is a part of some formulas and is 5.32E-38 esu. The Planck constant is 6.63E-34 J-s, so most single precision manipulations with these numbers will exceed the -1E+38 to 1E+38 range. Think carefully about the range of possible numbers in your computer calculations when choosing single precision or double precision floating types!

Double Precision Floating Numbers

The advantages of using double precision number types are many. A double precision number usually yields fifteen or sixteen decimal places, which is more than adequate for most calculations. The largest normalized double precision number is about 1.798E+308. The smallest normalized number is about 2.225E-308. (Again put a negative sign in front to determine the largest and smallest *negative* number.) Even though the speed of performing double precision computations is twice as slow as performing single precision computations, the computation speed on today's computers is still fast.

Aside: It is best to perform your calculations in the native mode for the computer you are using because of the time and precision involved when the computer converts your number to one with more or fewer bits. For example, the native mode for floating operations on a Macintosh is "extended mode", which is a ten byte, ~20 significant digit number.

The only drawback to consider for double precision numbers is that they will use twice as much disk space as that of single precision numbers.

A trivia question proposed by Brand Fortner: If you were sending a rocket to Pluto (5.91E+09 kilometers away) and your orbital calculations were perfect except that your value of pi was off in the sixteenth digit, by how much would you miss the planet? Answer: 1.3 millimeters.

Beware of These Operations

Calculations involving floating point numbers can introduce some major errors for the unaware programmer. Before I describe some "gotchas" to be aware of in your computing, let me introduce "round-off error." Round-off error in any floating point calculation arises due to machine accuracy. The round-off error corresponds to a change in the least significant bit of the mantissa. It is a characteristic of the computer hardware.

Subtracting Almost-Equal Numbers

One of the most common error-producing calculations involves the cancellation of significant digits due to subtraction of nearly equal numbers. Round-off error in this case dominates the result. The only significant digits remaining are those few low-order ones in which the operands differ, which may be beyond the precision of the floating-point format chosen. The solution for avoiding this problem is to rewrite the computation so that this situation is less likely to occur.

Ratio of Two Sets of Similar Single Precision Numbers

The ratio of similar single precision numbers results in values that are discretized because relatively few single precision exist between the minimum and maximum values of the resulting quotient.

The solution for this case is to use double precision number types
for this calculation.

Division by a Very Small Number
We all have encountered a division-by-zero error. Mathematically the
ratio is infinite, but your computer will respond in a way that is
likely to halt your program. Now suppose that your denominator is a
non-zero small number, and your numerator is many orders of magnitude
larger. A similar situation to the divide-by-zero problem will arise,
especially if you are using single-precision data types. One solution to
this problem is to normalize your ratio so that such a large magnitude
discrepancy doesn't exist. The easier solution is to rewrite your
calculation using double-precision so that you have a large range of
numbers to work from.

Floating Numbers for Logical Control
Equating two floating numbers to control a loop is inviting trouble. Say
a=1.417 and b=1.413. You wish to add .001 to b inside a loop and to stop
when b equals a. You may find your program in an infinite loop! Remember
that a computer stores your numbers in a binary form which gets
evaluated when your conditional statement is executed. So b could easily
surpass a without it ever equalling it. The solution is either to
convert a and b to integers which *can* be exactly compared, or else to
use "greater/less-than" for your logical control statement instead.

A computer is a tool, and a very useful one for simulating real-world
situations. But the computer's number manipulation uses a logic
different from what you are accustomed to in your paper calculations. Be
aware of the computer's limitations (and exploit its power and speed)!

References

Burden, Richard L., Numerical Analysis, PWS-Kent Publishing Co, 1989, pg 10-18.

Fortner, Brand, The Data Handbook, Spyglass, 1992, pg 35-58.

Hamming, R. W., Numerical Methods for Scientists and Engineers, Dover Press,
1973, pg 19-22.

Press, William, Teukolsky, Saul, Vetterling, Willam, and Flannery, Brian,
Numerical Recipes in Fortran, Cambridge University Press, 1992, pg 19-21.
--

********** ********
Amara Graps email: agraps@netcom.com
Computational Physics vita: finger agraps@shell5.ba.best.com
Multiplex Answers URL: http://www.amara.com/
********** ********

Subject: Re: Wow. exp() difficulties...
Posted by Amara Graps on Fri, 01 Aug 1997 07:00:00 GMT
View Forum Message <> Reply to Message

Thanks for your comments. I realize that my essay may have been a
little bit dated since I wrote it a few years ago.

William Clodius wrote:
>
> I have a few comments on Amara Graps post, mostly related to efficiency:
>
> Amara Graps wrote:
>>
>> <snip>
>>
>> The primary advantages of using integer and fixed point numbers are that
>> they take up less storage disk space and calculations with these numbers
>> are fast.
>
> Because of their simplicity it is in principle possible to make integer
> and fixed point number arithmetic faster than floating point arithmetic.
> However, in practice nowadays floating point arithmetic is as fast as
> integer arithmetic for addition and subtraction and often faster than
> integer arithmetic for multiplication and division.
[...]
>> Double Precision Floating Numbers
>> [...]
>> determine the largest and smallest *negative* number.) Even though the
>> speed of performing double precision computations is twice as slow as
>> performing single precision computations, the computation speed on
>> today's computers is still fast.
>
> In recent years this speed ratio is no longer valid. Some floating point
> processing units perform all their calculations in double or extended
> precision. As a result single precision can have the additional overhead
> of conversion to and from the double precision format. Whether single or
> double precision is more efficient in such cases depends on the
> magnitude of this overhead vs. increased cache misses due to the larger
> memory requirements for double.

Yes.. It's true. I discovered this on my Macintosh 5 years ago
running my thesis simulations for my variables in extended precision.
(I wrote my simulations in Pascal) It was faster to run it that way,
than change the variables to double precision, or even, single
precision.
I didn't know if the concept was true for other platforms though.
Thanks for the update.

Amara

--

 *** **
Amara Graps
amara@quake.stanford.edu
Solar Oscillation Investigations Stanford University
http://quake.stanford.edu/~amara/
 *** **

"Never fight an inanimate object." - P. J. O'Rourke
