Reproducible Science and Modern Scientific Software Development

13th eVITA Winter School in eScience sponsored by Norges forskningsråd
Dr. Holms Hotel, Geilo, Norway
January 20-25, 2013

Advanced Topics

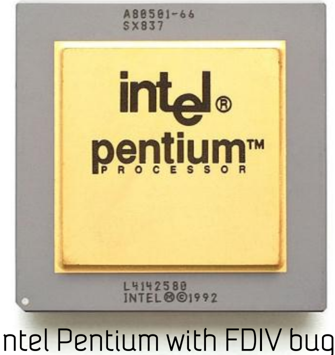
Dr. André R. Brodtkorb,
Research Scientist
SINTEF ICT, Dept. of Appl. Math.

Outline

- Floating point: It's fun!
- Parallel computing: It's n times as fun!
- Reporting performance

Floating point [1]

[1] IEEE Computer Society (August 29, 2008), *IEEE Standard for Floating-Point Arithmetic*



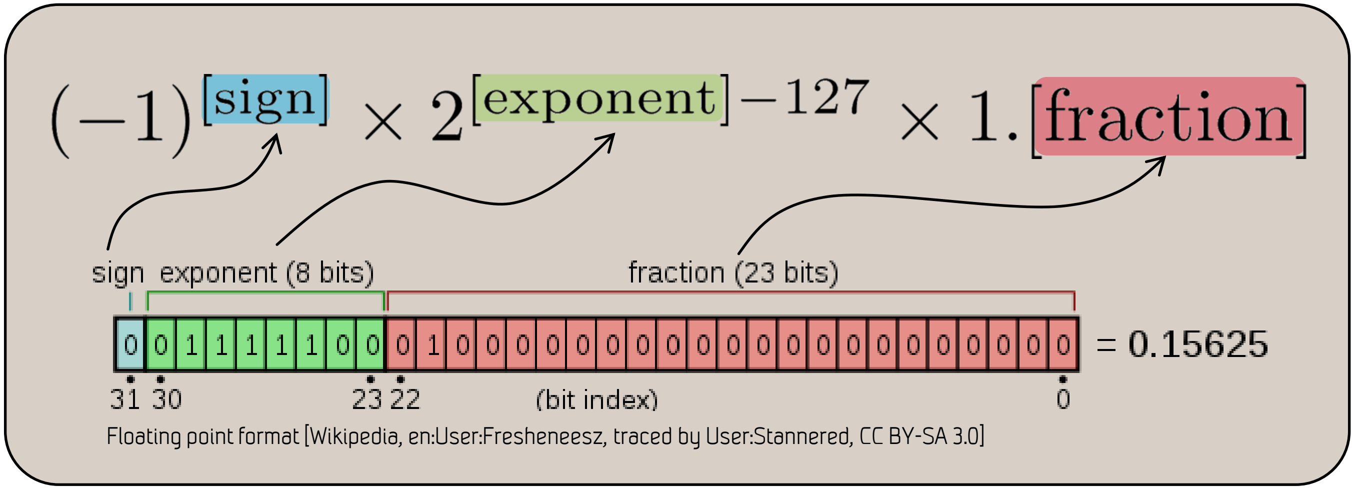
Intel Pentium with FDIV bug,
Wikipedia, user Appaloosa,
CC-BY-SA 3.0

"update [...] to address the hang that occurs when parsing strings like "2.2250738585072012e-308" to a binary floating point number" [1]

[1] http://www.oracle.com/technetwork/java/javase/fpupdater-tool-readme-305936.html

A floating point number on a binary computer

- Floating point numbers are represented using a binary format:



Floating point format [Wikipedia, en:User:Fresheneesz, traced by User:Stannered, CC BY-SA 3.0]

- Defined in the IEEE-754-1985, 2008 standards
  - 1985 standard mostly used up until the last couple of years

Rounding errors

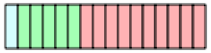
- Floating point has limited precision
- All intermediate results are rounded
- Even worse, not all numbers are representable in floating point
- Demo: 0.1 in IPython

Python:

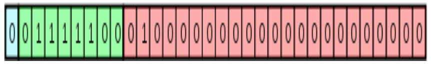
```
› print 0.1
0.1
› print "%.10f" % 0.1
0.1000000000
› print "%.20f" % 0.1
0.10000000000000000555
› print "%.30f" % 0.1
0.100000000000000005551115123126
```

Floating point variations (IEEE-754 2008)

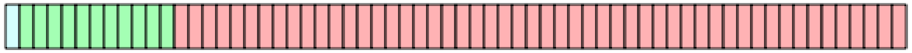
- Half: 16-bit float: Roughly 3-4 correct digits



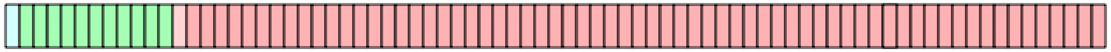
- Float / REAL*4: 32-bit float: Roughly 6-7 correct digits



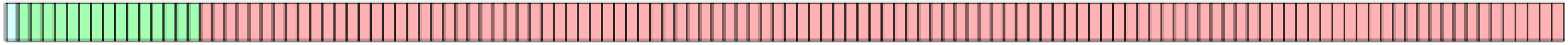
- Double / REAL*8: 64-bit float: Roughly 13-15 correct digits



- Long double / REAL*10: 80-bit float: Roughly 18-21 correct digits



- Quad precision: 128-bit float: Roughly 33 - 36 correct digits



The long double – a real bastard

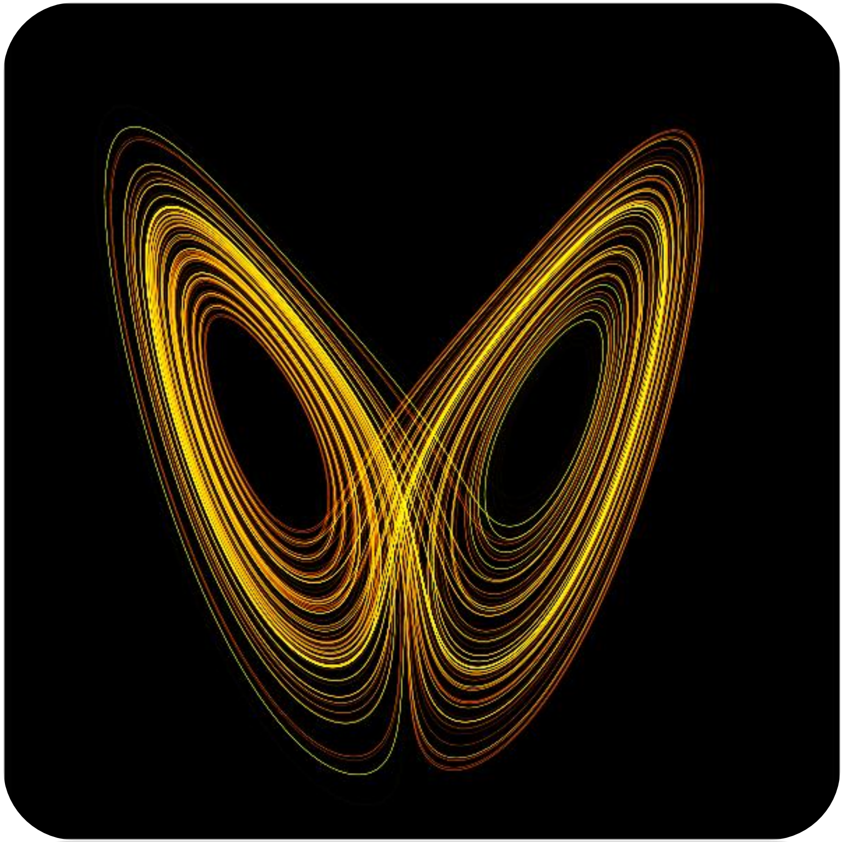
- What is a long double?
  - Defined in C99/C11-standard, its an 80-bit floating point number slightly different than the 32 and 64-bit numbers
  - C99/C11 not implemented in MSVC…
  - Available as __float80 or long double in g++

- Was introduced to give enough accuracy for exponentiation (hardware did not have support for it, and instead computed $x^y = 2^{y \log_2 x}$)

- Extremely unintuitive: when a variable x is in a register, it has 80-bit precision. When it is flushed to the caches or main memory, it can have 128-bit storage.

Floating point and numerical errors

- Some systems are chaotic
  - Is single precision accurate enough for your model?
  - Is double precision --"--?
  - Is quad precision --"--?
  - Is ...
- Put another way:
  - What is the minimum precision required for your model?



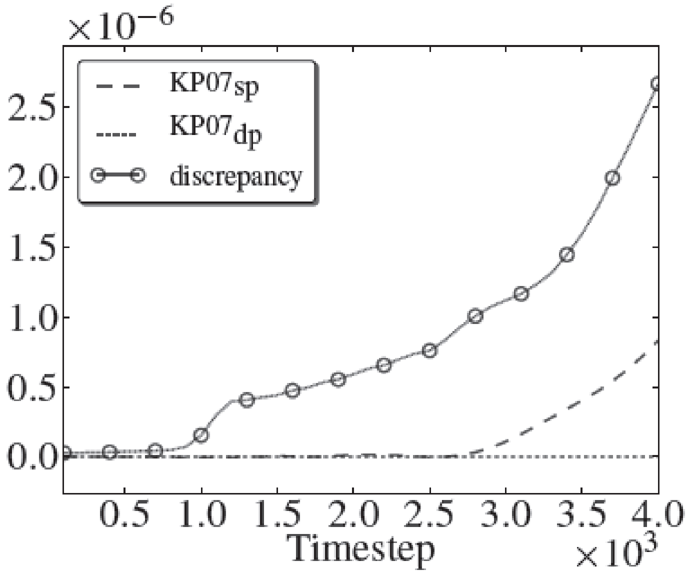
Lorenz strange attractor, Wikimol, wikipedia, CC-BY-SA 3.0

Single versus double precision in shallow water

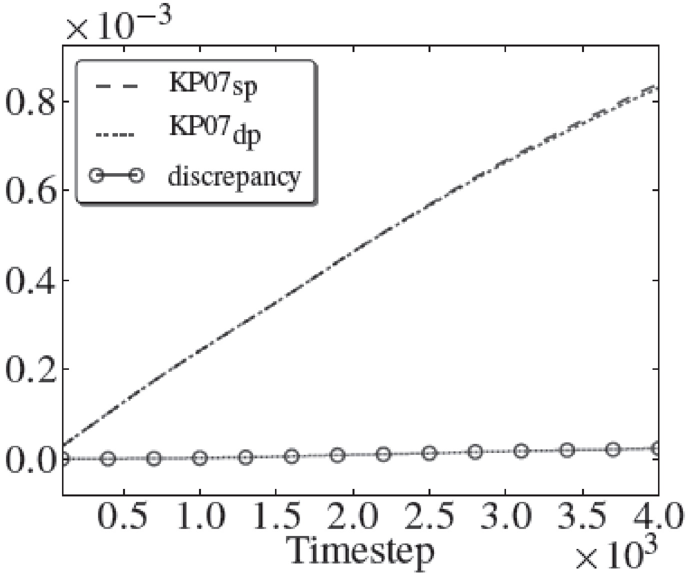
- Shallow water equations: Well studied equations for physical phenomenon
- Difficult to capture wet-dry interfaces accurately
- Let's see the effect of single versus double precision measured as error in conservation of mass

Single versus double precision [1]

- Simple case (analytic-like solution)
  - No wet-dry interfaces
  - Single precision gives growing errors that are "devastating"!



- Realistic case (real-world bathymetry)
  - Single precision errors are drowned by model errors

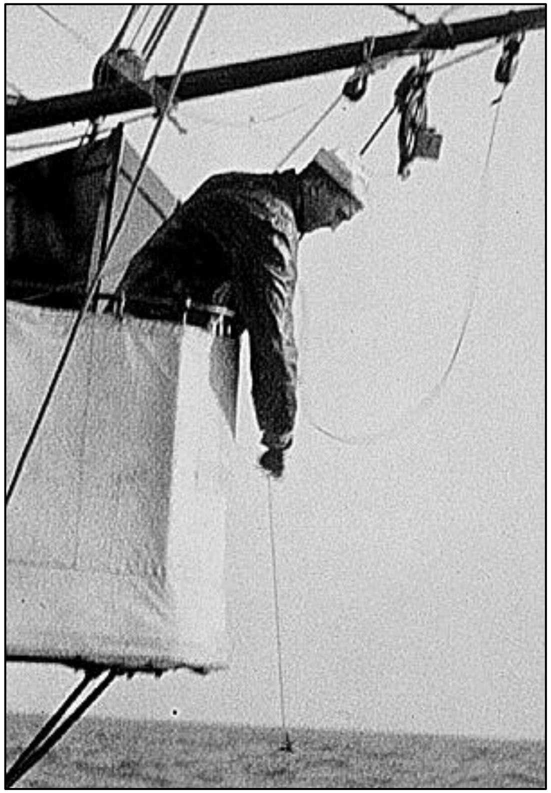


[1] A. R. Brodtkorb, T. R. Hagen, K.-A. Lie and J. R. Natvig, **Simulation and Visualization of the Saint-Venant System using GPUs**, *Computing and Visualization in Science, 2011*

Floating point is often the least problem wrt accuracy

- Garbage in, garbage out
- Many sources for errors
  - Humans!
  - Model and parameters
  - Measurement
  - Storage
  - Gridding
  - Resampling
  - Computer precision
  - …

Recycle image from recyclereminders.com
Cray computer image from Wikipedia, user David.Monniaux



Seaman paying out a sounding line during a hydrographic survey of the East coast of the U.S. in 1916. (NOAA, 2007).

Catastrophic and benign cancellations [1]

- A classical way to introduce a large numerical error is to have a catastrophic cancellation:

$$x^2 - y^2 \Rightarrow (x-y)(x+y)$$

- The first variant above is subject to catastrophic cancellation if x and y are relatively close. The second does not suffer from this catastrophic cancellation!

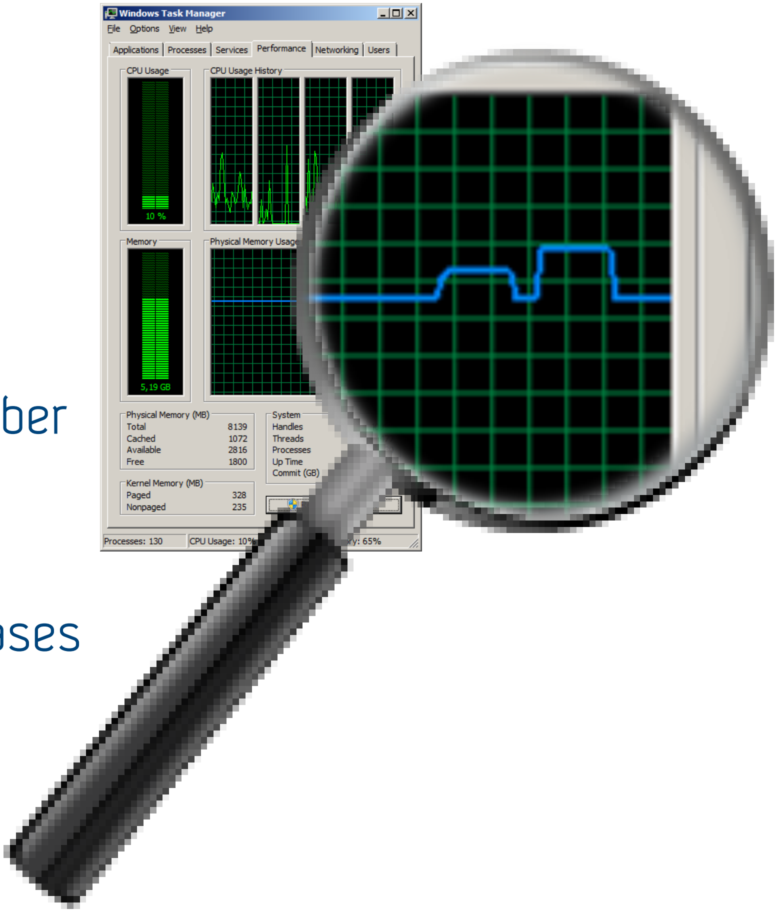
$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad \text{vs} \qquad r = \frac{2c}{-b \pm \sqrt{b^2 - 4ac}}$$

[1] *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, David Goldberg, Computing Surveys, 1991

So what should I use?

- Single precision
  - Single precision uses **half** the memory of double precision
  - Single precision executes **twice** as fast for certain situations (SSE & AVX instructions)
  - Single precision gives you **half** the number of correct digits

- Double precision is not enough in certain cases
  - Quad precision? Arbitrary precision?
  - Extremely expensive operations (100x+++ time usage)



Physical Memory (MB)	
Total	8139
Cached	1072
Available	2816
Free	1800

Kernel Memory (MB)	
Paged	328
Nonpaged	235

System
Handles
Threads
Processes
Up Time
Commit (GB)

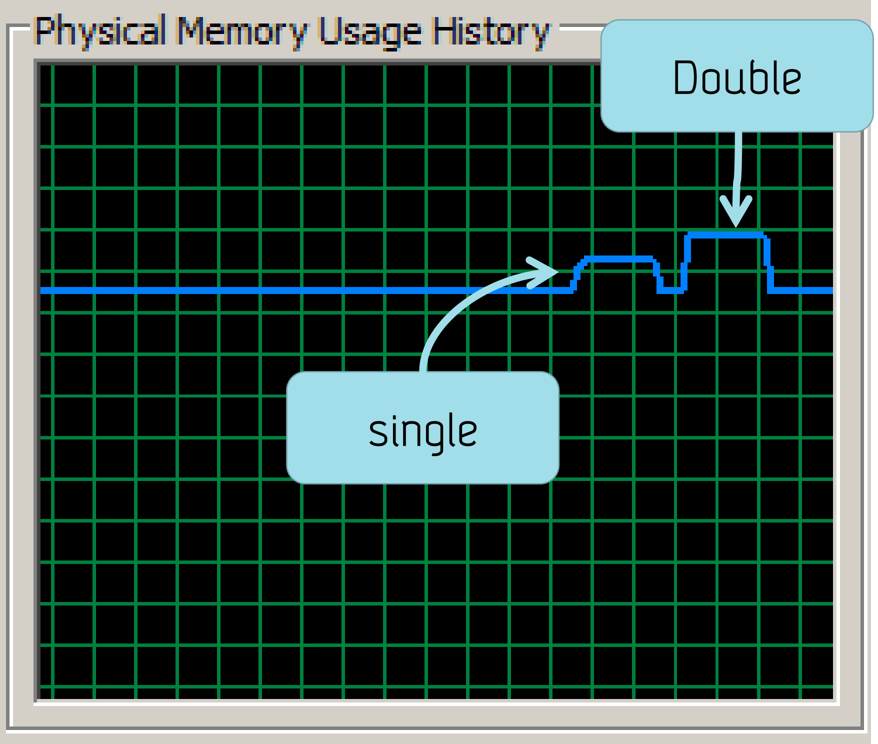
Processes: 130 CPU Usage: 10% ry: 65%

Demo time

- Memory allocation example
  - How much memory does the computer need if I'm allocating 100.000.000 floating point values in a) single precision, and b) double precision?

```
Allocating float:
Address of first element: 00DC0040
Address of last element: 18B38440
Bytes allocated: 400000000

Allocating double:
Address of first element: 00DC0040
Address of last element: 308B0840
Bytes allocated: 800000000
```



Demo time rev 2

Floating point example

- What is the result of the following computation?

```
val = 0.1;
for (i=0  to 10.000.000) {
   result = result + val
}
```

```
Float:
Floating point bits=32
1087937.00000000000000000000000000000000000000000000000000
Completed in 0.01859299999999999841726605609437683597207069396973 s.

Double:
Floating point bits=64
999999.99983897537458688020706176757812500000000000000000
Completed in 0.02386800000000000032684965844964608550071716308594 s.

Long double (__float80):
Floating point bits=128
1000000.00000000871274323799298144876956939697265625000000
Completed in 0.02043599999999999930477834197972697438672184944153 s.

Quad (__float128):
Floating point bits=128
1000000.00000000000000000000000000000000000000000000000000
Completed in 1.39770400000000000574686964682769030332565307617187500 s.
```

The patriot missile…

- Designed by the Raytheon (US) as an air defense system.
- Designed for time-limited use (up-to 8 hours) in mobile locations.
- Heavily used as static defenses using the Gulf war.
- Failed to intercept an incoming Iraqi Scud missile in 1991.
- 28 killed, 98 injured.

The patriot missile…

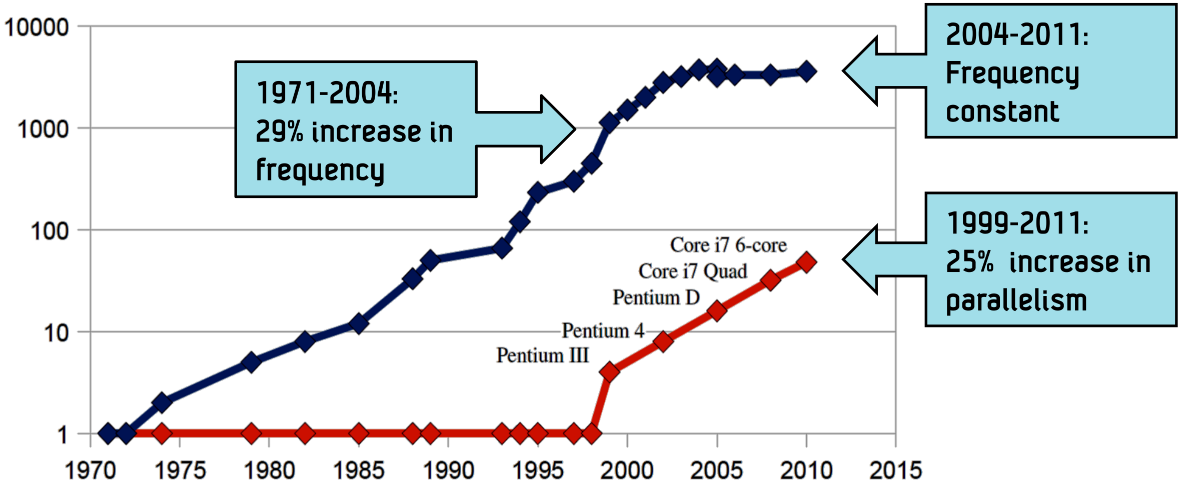
- It appears, that 0.1 seconds is not really 0.1 seconds…
  - Especially if you add a large amount of them

Hours	Inaccuracy (sec)	Approx. shift in Range Gate (meters)
0	0	0
1	.0034	7
8	.0025	55
20	.0687	137
48	.1648	330
72	.2472	494
100	.3433	687

http://sydney.edu.au/engineering/it/~alum/patriot_bug.html

Floating point and parallelism

Should I care about parallel computing?



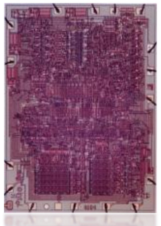
1971-2004: 29% increase in frequency

2004-2011: Frequency constant

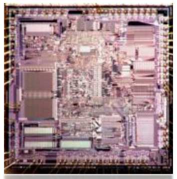
1999-2011: 25% increase in parallelism

A serial program uses 2% of available resources!

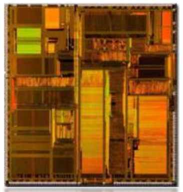
Parallelism technologies:
- Multi-core (8x)
- Hyper threading (2x)
- AVX/SSE/MMX/etc (8x)



1971: Intel 4004,
2300 trans, 740 KHz

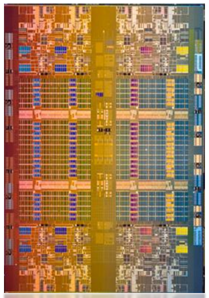


1982: Intel 80286,
134 thousand trans, 8 MHz



1993: Intel Pentium P5,
1.18 mill. trans, 66 MHz

2000: Intel Pentium 4,
42 mill. trans, 1.5 GHz



2010: Intel Nehalem,
2.3 bill. trans, 8 X 2.66 GHz

Floating point and parallelism

- Fact 1: Floating point is non-associative:
  - a*(b*c) != (a*b)*c
  - a+(b+c) != (a+b)+c
  - …

Floating point and parallelism

- Fact 2: Parallel execution is non-deterministic
  - Reduction operations (sum of elements, maximum value, minimum value, average value, etc.)

- Combine fact 1 and fact 2 for great joys!

Demo time ver 3

- Openmp summation of 10.000.000 numbers using 10 threads

```
val = 0.1;
#omp parallel for
for (i=0  to 10.000.000) {
   result = result + val
}
```

```
OpenMP float test using 10 threads
Float:
Floating point bits=32
Run 0: 976668.75000000000000000000000000000000000000000000000000
Run 1: 976759.37500000000000000000000000000000000000000000000000
Run 2: 976424.87500000000000000000000000000000000000000000000000
Run 3: 977388.37500000000000000000000000000000000000000000000000
Run 4: 981089.06250000000000000000000000000000000000000000000000
Run 5: 976620.25000000000000000000000000000000000000000000000000

Double:
Floating point bits=64
Run 0: 1000000.00003875180000000000000000000000000000000000000000
Run 1: 1000000.00003898310000000000000000000000000000000000000000
Run 2: 1000000.00003432810000000000000000000000000000000000000000
Run 3: 1000000.00003912390000000000000000000000000000000000000000
Run 4: 1000000.00003827200000000000000000000000000000000000000000
Run 5: 1000000.00003756480000000000000000000000000000000000000000
```

Kahan summation [1]

- It appears that naïve summation works really poorly for floating point, especially with parallelism
- We can try to use algorithms that take floating point into account

```
 function KahanSum(input)
    var sum = 0.0
    var c = 0.0          //A running compensation for lost low-order bits.
    for i = 1 to input.length {
        y = input[i] - c    //So far, so good: c is zero.
        t = sum + y         //Alas, sum is big, y small,
                            //so low-order digits of y are lost.
        c = (t - sum) - y   //(t - sum) recovers the high-order part of y;
                            //subtracting y recovers -(low part of y)
                            //Algebraically, c should always be zero.
                            //Beware eagerly optimising compilers!
        sum = t
    }
return sum
```

[1] Inspired by Bob Robey, EPSum, ICERM 2012 talk, http://faculty.washington.edu/rjl/icerm2012/Lightning/Robey.pdf

Demo time ver 4

- Kahan summation in parallel!

```
Float:
Floating point bits=32
   Traditional sum,   Kahan sum
Run 0: 499677.062500, 4996754.500
Run 1: 499679.250000, 4996754.500
Run 2: 499677.468750, 4996754.500
Run 3: 499676.312500, 4996754.500
Run 4: 499676.687500, 4996754.500
Run 5: 499679.937500, 4996754.500

Double:
Floating point bits=64
       Traditional sum,         Kahan sum
Run 0: 500136.4879299310900, 5001364.87929929420
Run 1: 500136.4879299307400, 5001364.87929929420
Run 2: 500136.4879299291600, 5001364.87929929420
Run 3: 500136.4879299313800, 5001364.87929929420
Run 4: 500136.4879299254400, 5001364.87929929420
Run 5: 500136.4879299341700, 5001364.87929929420
```

Advanced floating point

Rounding modes

- Round towards +infinity (ceil)
- Round towards –infinity (floor)
- Round to nearest (and up for 0.5)
- Round to nearest (and towards zero for 0.5)
- Round towards zero
- **Can be used for interval arithmetics!**

Special floating point numbers

- Signed zeros -0 != +0

- Signed not-a-numbers:
  quiet NaN, and signaling NaN (gives exception)
  examples: 0/0, sqrt(-1), …
  (x == x) is false if x is a NaN

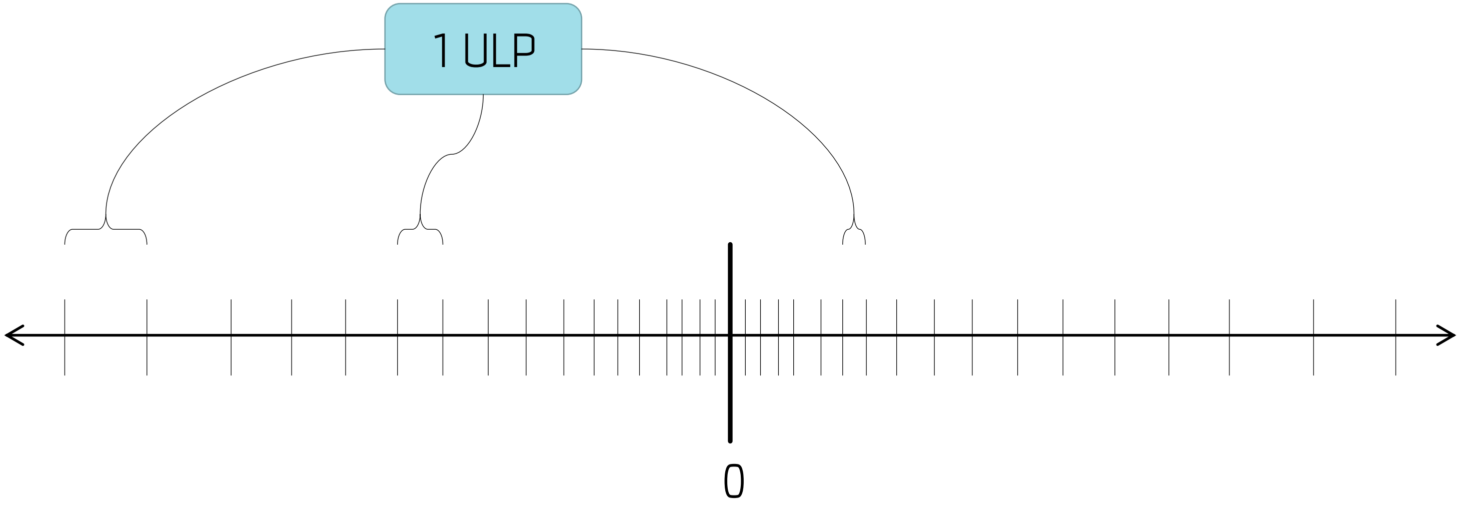
Special floating point numbers

- Signed infinity
  - Numbers that are too large to represent
    5/0 = +infty, -8/0 = -infty

- Subnormal or denormal numbers
  - Numbers that are too small to represent

Units in the last place [1]

- Unit in the last place or unit of least precision (ULP) is the spacing between floating point numbers

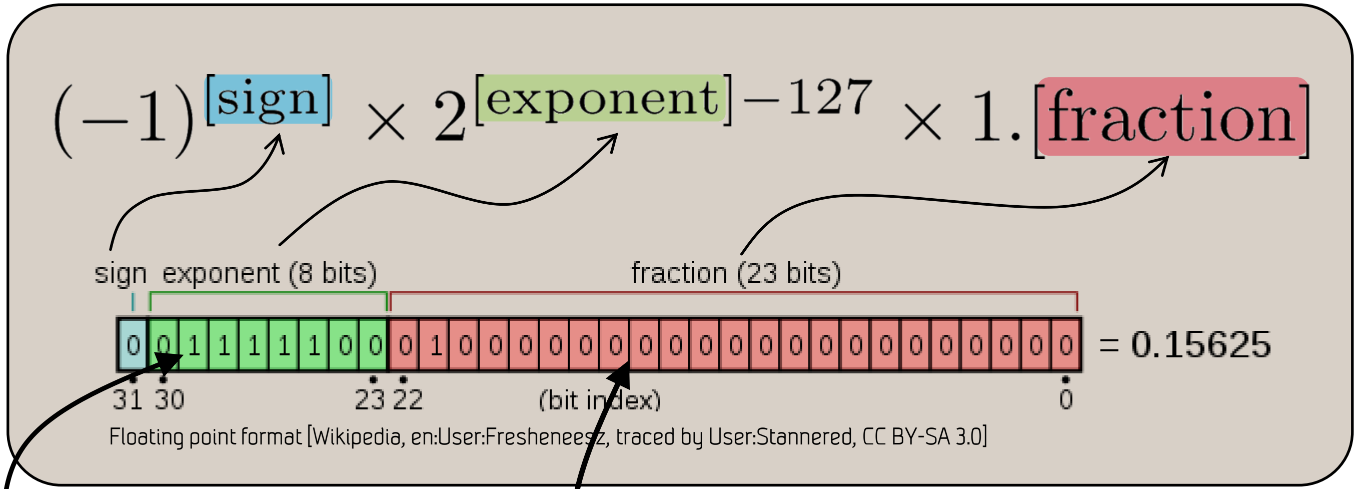


- "The most natural way to measure floating point errors"
- Number of contaminated digits: $\log_2 n$ when the error is n ulps

[1] What every computer scientist should know about floating-point arithmetic, David Goldberg, Computing Surveys , 1991

Subnormals

- Subnormals / denormals are gradual underflows
  - Graceful loss of precision instead of flush to zero
  - Can be really, really, expensive



Floating point format [Wikipedia, en:User:Fresheneesz, traced by User:Stannered, CC BY-SA 3.0]

Exponent zero when subnormal

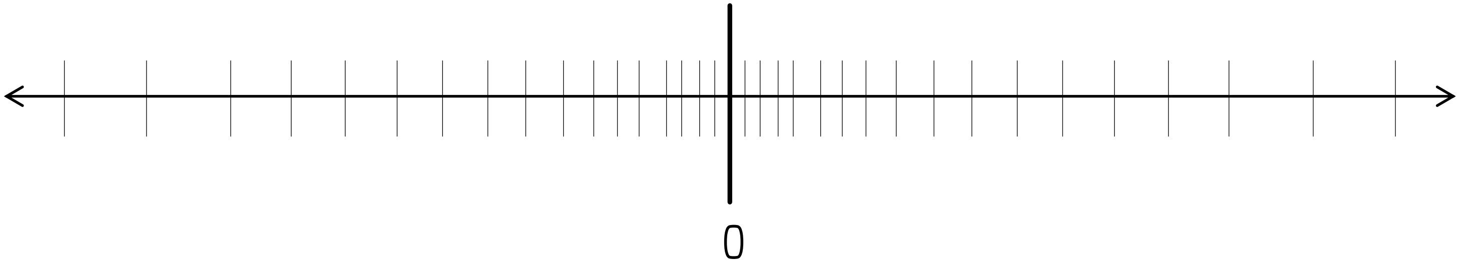
Leading zeros appear in significand / fraction / mantissa when subnormal

Some differences between 1985 and 2008

- Floating point multiply-add as a fused operation
  - a = b*c+d with only **one** round-off error
  - GPUs implement this already
- This is basically the same deal as the extended precision.
  - It's a good idea to use this instruction, but it gives "unpredictable" results
  - Users need to be aware that computers are not exact, and that two computers will not always give the same answer

Floating point best practices

- Floating point has the highest resolution around 0:
  - Lattice Bolzmann intermediate results: subtract 1 when storing to keep resolution
  - Store water elevations in shallow water as depths, or as deviations from mean sea level, not elevations.



Silent Data Corruption [1]

- Silent data corruption happens when a bit is flipped "by itself"…
- Can be handled somewhat with ECC memory (available on servers)
- Can have many causes: Environmental (temperature/voltage fluctuations; particles), manufacturing residues, oxide breakdown, electro-static discharge.
  - Estimate of 1 cosmic-ray-neutron-induced SDC every 1.5 months of operation (RoadRunner)
  - Smaller feature sizes increases frequency of SDC's

[1] Sarah Michalak, Silent Data Corruption and Other Anomalies, ICERM talk, 2012, http://faculty.washington.edu/rjl/icerm2012/Lightning/michalak.pdf

Reporting performance

What do we do in papers we publish?

A. Solve a problem that we previously could not

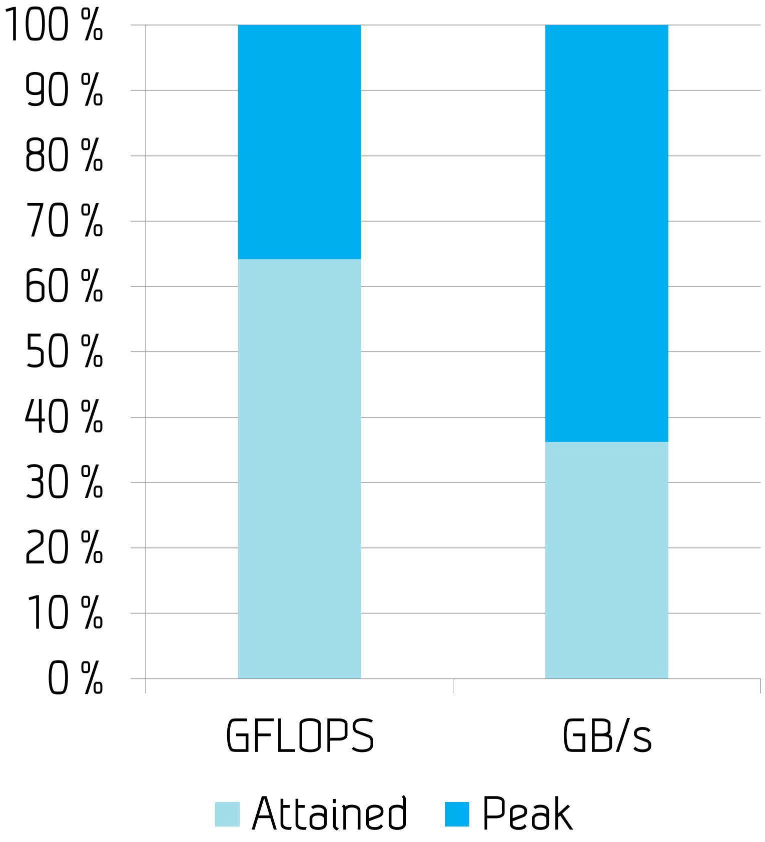
B. Solve an existing problem better than previously
   1. More accurately in the same amount of time
   2. As accurate as before, but faster
   3. A more demanding version of an existing problem

C. Perform a case study / write a survey article / …

Performance reporting is often a key element in B

Assessing performance

- Different ways of assessing performance
    - Algorithmic performance, numerical performance, wall clock time, …
- Speedups can be dishonest
    - Comparison of apples to oranges
- Sanity check for performance: Profile your code, and see what percentage of peak performance you attain
    - The aim should be to approach peak performance



Top Ways of Misleading the Masses [1]

1. Quote only 32-bit performance results, not 64-bit results
2. Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application
3. Quote performance results projected to a full system
4. When direct run time comparisons are required, compare with an old code on an obsolete system
5. If all else fails, show pretty pictures and animated videos, and don't talk about performance

[1] Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers David H. Bailey, 1991

"In established engineering disciplines a 12 % improvement, easily obtained, is never considered marginal and I believe the same viewpoint should prevail in software engineering"

--Donald Knuth

Summary

- Floating point can be devastating when misused
  - But floating point is most often **not** the largest problem
  - Programming errors, model errors, measurement errors…
- Floating point and parallel computing do not work well at all
  - Examine at algorithms that handle summation and parallelism well without affecting performance.
- Tell people that computers are non-deterministic
  Tell people that all results have uncertainties by including error bars
- Be methodical, thorough, and honest; also when reporting performance

Further reading

- Accuracy and Stability of Numerical Algorithms, Nicholas J. Higham
- What every computer scientist should know about floating-point arithmetic, David Goldberg, Computing Surveys , 1991.
- Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers, David H. Bailey, Supercomputing Review, 1991.
- Ten Ways to Fool the Masses When Giving Performance Results on GPUs, Scott Pakin, HPC Wire, 2011