Floating-Point Math and Accuracy

Dr. Axel Kohlmeyer

Senior Scientific Computing Expert

Information and Telecommunication Section
The Abdus Salam International Centre
for Theoretical Physics

http://sites.google.com/site/akohlmey/

akohlmey@ictp.it

Errors in Scientific Computing

- Before computations:
  - Modeling: neglecting certain properties
  - Empirical data: not every input is known perfectly
  - Previous computations: data may be taken from other (error-prone) numerical methods
  - Sloppy programming (e.g. inconsistent conversions)
- During computations:
  - Truncation: a numerical method approximates a continuous solution
  - Rounding: computers offer only finite precision in representing real numbers

Example

- Computing the surface of the earth using

$$A = 4\pi r^2$$

- This involves several approximations:
  - Modeling: the earth is not exactly a sphere
  - Measurement: earth's radius is an empirical number
  - Truncation: the value of π is truncated
  - Rounding: all numbers used are rounded due to arithmetic operations in the computer
- Total error is the sum of all, but one dominates

Floating Point Math in HPC

- Understanding floating point math is at the core of many (traditional) HPC applications (physics, chemistry, applied math)
- Real numbers have unlimited accuracy
- Computers “think” digital (i.e. in integer math) => using integer fractions already has “holes”
- Approximation: use scientific notation and truncate the mantissa

$$\pm\text{mantissa} * 10^{\pm\text{exponent}}$$

IEEE 754 Floating-point Numbers

- Internal representation:

$$\pm\text{mantissa} * 2^{\pm\text{exponent}}$$

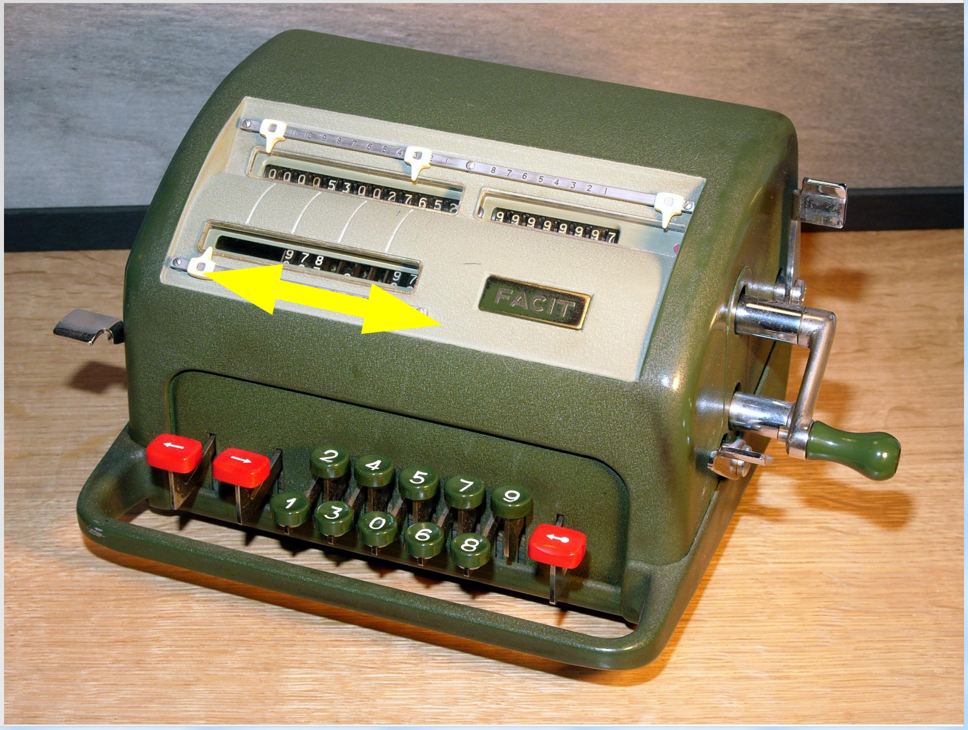
- The standard defines as bit patterns with a sign bit, an exponential field, and a fraction field
  - Single precision: 8-bit exponent, 23-bit fraction
  - Double precision: 11-bit exponent, 52-bit fraction
- The standard defines: storage format, result of operations, special values (infinity, overflow,...) => portability of compute kernels ensured

Range of Single-precision Numbers

- Largest possible “normal” number is ≈$3.4*10^{38}$
- Smallest positive number is ≈$1.8*10^{-38}$
- In comparison: signed 32-bit integer numbers range only from -214783648 to 214783647 and the smallest positive number is 1
- How can we represent so many more numbers in floating point than in integer?
- We don't: the number of unique bit patterns has to be the <u>same</u> => truncation

Density of Floating-point Numbers

- Since the same number of bits are used for the fraction part of the FP number, the exponent determines the representable number density
- E.g.: there are 8,388,607 numbers between 1.0 and 2.0, but only 8191 between 1023.0 and 1024.0
- => accuracy depends on the magnitude



Floating-Point Math Properties

- Many FP math operations do not result in a representable floating point number => rounding

  Examples: 1.0/3.0, 1.0/100.0

- IEEE-754 defines rounding rules
- FP math is commutative, but not associative!
  $1.0 + (1.5*10^{38} + (- 1.5*10^{38})) = 1.0$
  $(1.0 + 1.5*10^{38}) + (- 1.5*10^{38}) = 0.0$
- => results may change, if a compiler changes code to run more efficient => compiler flags

How To Reduce Errors

- Avoid summing numbers of different magnitude
  - Sort first and sum in ascending order
  - Sum in blocks (pairs) and then sum the sums
  - Kahan summation (carry over errors in a compensation variable) -> Wikipedia
    => slower since more operations
    => compilers might optimize it away
  - Use (scaled) integers, if number range allows it
- NOTE: summing numbers in parallel may give different results depending on parallelization

Subtracting FP Numbers

- Subtraction of two floating-point numbers of the same sign and similar magnitude (same exponent) will always be representable
- Leading bits in fraction cancel
  => results have less 'valid' digits
  => (potential) loss of information
- Careful when using the result in multiplication
  => result is 'tainted' by low accuracy

Comparing FP Numbers

- Floating-point results are usually **inexact**
  => comparing for equality is dangerous
  Example: don't use FP as loop index
  => loop.c test code. Check number of iterations.
- It is OK to use exact comparison:
  - When results have to be bitwise identical
  - To prevent division by zero errors
- Best to compare against expected error
  => macheps.c test code -> FPU precision

More Test Examples

- sum_number: compare summing accuracy depending on order 1-N or N-1.
- paranoia: IEEE-754 compliance test
  => Try with different compilers and optimization and FP math-related compiler flags
- mathopt: compute windowed average over two and three number window.
  => compare speed division by 2 vs division by 3
  => impact of compiler flags vs. code rewrite
- inverse: check for not representable numbers