MANCHESTER 1824

The University of Manchester

M\cr NA
Manchester Numerical Analysis

Multiprecision Algorithms

Nick Higham
School of Mathematics
The University of Manchester

`http://www.maths.manchester.ac.uk/~higham/`
`@nhigham, nickhigham.wordpress.com`

Mathematical Modelling, Numerical Analysis and Scientific Computing, Kácov, Czech Republic May 27-June 1, 2018.

Outline

Multiprecision arithmetic: floating point arithmetic supporting multiple, possibly arbitrary, precisions.

- Applications of & support for low precision.
- Applications of & support for high precision.
- How to exploit different precisions to achieve faster algs with higher accuracy.
- Focus on
  - **iterative refinement** for $Ax = b$,
  - **matrix logarithm**.

Download these slides from `http://bit.ly/kacov18`

Lecture 1

- Floating-point arithmetic.
- Hardware landscape.
- Low precision arithmetic.

siam news

MORE AT SIAM

SIAM NEWS OCTOBER 2017

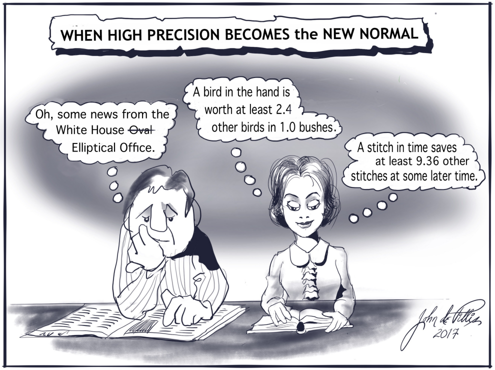
Research | October 02, 2017

Print

A Multiprecision World

By Nicholas Higham

Traditionally, floating-point arithmetic has come in two precisions: single and double. But with the introduction of support for other precisions, thanks in part to the influence of applications, the floating-point landscape has become much richer in recent years.



Floating Point Number System

Floating point number system $F \subset \mathbb{R}$:

$$y = \pm m \times \beta^{e-t}, \qquad 0 \le m \le \beta^t - 1.$$

- *Base* β ($\beta = 2$ in practice),
- *precision* t,
- *exponent range* $e_{\min} \le e \le e_{\max}$.

Assume **normalized**: $m \ge \beta^{t-1}$.

Floating Point Number System

Floating point number system $F \subset \mathbb{R}$:

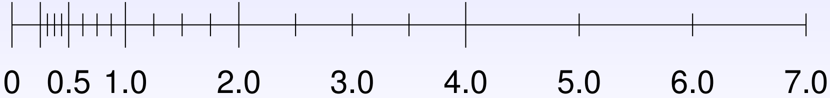
$$y = \pm m \times \beta^{e-t}, \qquad 0 \le m \le \beta^t - 1.$$

- *Base* β ($\beta = 2$ in practice),
- *precision* t,
- *exponent range* $e_{\min} \le e \le e_{\max}$.

Assume **normalized**: $m \ge \beta^{t-1}$.

Floating point numbers are **not equally spaced**.

If $\beta = 2$, $t = 3$, $e_{\min} = -1$, and $e_{\max} = 3$:



Subnormal Numbers

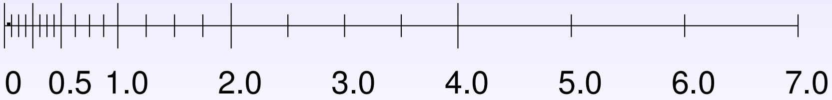
$0 \neq y \in F$ is *normalized* if $m \geq \beta^{t-1}$. Unique representation.

Subnormal numbers have minimum exponent and not normalized:

$$y = \pm m \times \beta^{e_{\min}-t}, \qquad 0 < m < \beta^{t-1},$$

Fewer digits of precision than the normalized numbers.

Subnormal numbers fill the gap between $\beta^{e_{\min}-1}$ and 0 and are *equally spaced*. Including subnormals in our toy system:



IEEE Standard 754-1985 and 2008 Revision

Type	Size	Range	$u = 2^{-t}$
half	16 bits	$10^{\pm 5}$	$2^{-11} \approx 4.9 \times 10^{-4}$
single	32 bits	$10^{\pm 38}$	$2^{-24} \approx 6.0 \times 10^{-8}$
double	64 bits	$10^{\pm 308}$	$2^{-53} \approx 1.1 \times 10^{-16}$
quadruple	128 bits	$10^{\pm 4932}$	$2^{-113} \approx 9.6 \times 10^{-35}$

- Arithmetic ops $(+, -, *, /, \sqrt{})$ performed *as if* first calculated to infinite precision, then rounded.
- Default: round to nearest, round to even in case of tie.
- Half precision is a *storage format only*.

Relative Error

If $\widehat{x} \approx x \in \mathbb{R}^n$ the **relative error** is

$$\frac{\|x - \widehat{x}\|}{\|x\|}.$$

The **absolute error** $\|x - \widehat{x}\|$ is scale dependent.

Common error not to normalize errors and residuals.

Rounding

For $x \in \mathbb{R}$, $fl(x)$ is an element of F nearest to x, and the transformation $x \to fl(x)$ is called **rounding** (to nearest).

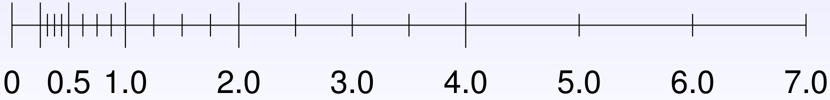
Theorem

If $x \in \mathbb{R}$ lies in the range of F then

$$fl(x) = x(1+\delta), \qquad |\delta| \le u.$$

$u := \frac{1}{2}\beta^{1-t}$ is the **unit roundoff**, or machine precision.

The **machine epsilon**, $\epsilon_M = \beta^{1-t}$ is the spacing between 1 and the next larger floating point number (`eps` in MATLAB).



Model vs Correctly Rounded Result

$y = x(1+\delta)$, **with** $|\delta| \le u$ **does not imply** $y = fl(x)$**.**

$\beta = 10$,
$t = 2$

x	y	$\lvert x-y\rvert/x$	$u = \frac{1}{2}10^{1-t}$
9.185	8.7	5.28e-2	5.00e-2
9.185	8.8	4.19e-2	5.00e-2
9.185	8.9	3.10e-2	5.00e-2
9.185	9.0	2.01e-2	5.00e-2
9.185	9.1	9.25e-3	5.00e-2
9.185	9.2	1.63e-3	5.00e-2
9.185	9.3	1.25e-2	5.00e-2
9.185	9.4	2.34e-2	5.00e-2
9.185	9.5	3.43e-2	5.00e-2
9.185	9.6	4.52e-2	5.00e-2
9.185	9.7	5.61e-2	5.00e-2

Model for Rounding Error Analysis

For $x, y \in F$

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \qquad |\delta| \le u, \quad \text{op} = +, -, *, /.$$

Also for $\text{op} = \sqrt{\ }$.

Model for Rounding Error Analysis

For $x, y \in F$

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \qquad |\delta| \le u, \quad \text{op} = +, -, *, /.$$

Also for $\text{op} = \sqrt{}$.

Sometimes more convenient to use

$$fl(x \text{ op } y) = \frac{x \text{ op } y}{1+\delta}, \qquad |\delta| \le u, \quad \text{op} = +, -, *, /.$$

Model is weaker than $fl(x \text{ op } y)$ being correctly rounded.

Precision versus Accuracy

$$\begin{aligned} fl(abc) &= ab(1+\delta_1)\cdot c(1+\delta_2) \qquad |\delta_i| \le u, \\ &= abc(1+\delta_1)(1+\delta_2) \\ &\approx abc(1+\delta_1+\delta_2). \end{aligned}$$

- Precision $= u$.
- Accuracy $\approx 2u$.

Precision versus Accuracy

$$\begin{aligned} fl(abc) &= ab(1+\delta_1)\cdot c(1+\delta_2) \qquad |\delta_i| \le u, \\ &= abc(1+\delta_1)(1+\delta_2) \\ &\approx abc(1+\delta_1+\delta_2). \end{aligned}$$

- Precision $= u$.
- Accuracy $\approx 2u$.

Accuracy is not limited by precision


65
220

Fused Multiply-Add Instruction

A multiply-add instruction with just one rounding error:

$$fl(x + y * z) = (x + y * z)(1 + \delta), \qquad |\delta| \le u.$$

With an FMA:

- Inner product $x^T y$ can be computed with half the rounding errors.
- In the IEEE 2008 standard.
- Supported by much hardware, including NVIDIA Volta architecture (P100, V100) at FP16.

Fused Multiply-Add Instruction (cont.)

- The algorithm of **Kahan**

  1 $w = b * c$

  2 $e = w - b * c$

  3 $x = (a * d - w) + e$

  computes $x = \det(\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right])$ with high relative accuracy

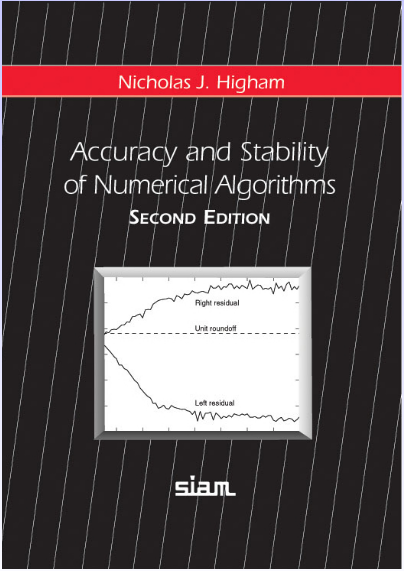
But

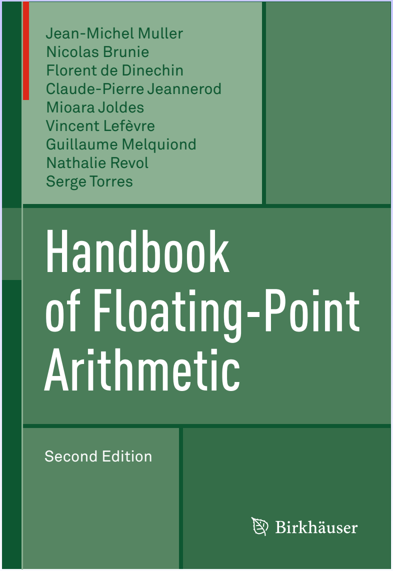
- What does `a*d + c*b` mean?
- The product

$$(x + iy)^*(x + iy) = x^2 + y^2 + i(xy - yx)$$

  may evaluate to non-real with an FMA.
- $b^2 - 4ac$ can evaluate negative even when $b^2 \geq 4ac$.

References for Floating-Point





ARM NEON

ARMDeveloper ▾ Products Markets Technologies Support

Overview big.LITTLE NEON DSP ▾ DynamIQ Graphics Technologies ▾ TrustZone ▾ Compute Library Floating Point

Overview

The NEON technology is a packed SIMD architecture. NEON registers are considered as vectors of elements of the same data type. Multiple data types are supported by the technology. The following table describes data types as supported by the architecture version.

	ARMv7-A/R	ARMv8-A/R	ARMv8-A
		AArch32	AArch64
Floating-point	32-bit	16-bit*/32-bit	16-bit*/32-bit/64-bit
Integer	8-bit/16-bit/32-bit	8-bit/16-bit/32-bit/64-bit	8-bit/16-bit/32-bit/64-bit

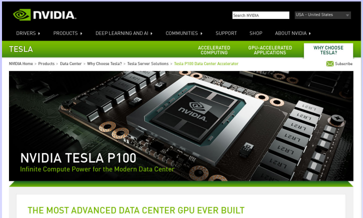
SIMD Architecture

The NEON instructions perform the same operations in all lanes of the vectors. The number of operations performed depends on the data types. NEON instructions allow up to:

- 16x8-bit, 8x16-bit, 4x32-bit, 2x64-bit integer operations
- 8x16-bit*, 4x32-bit, 2x64-bit** floating-point operations

The implementation on NEON technology can also support issue of multiple instructions in parallel.

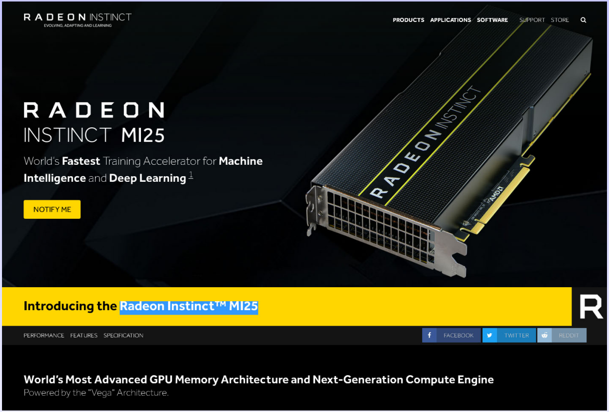
NVIDIA Tesla P100 (2016), V100 (2017)



- "The Tesla P100 is the world's first accelerator built for deep learning, and has native hardware ISA support for **FP16** arithmetic"
- V100 tensor cores do 4×4 mat mult in one clock cycle.

	TFLOPS		
	double	single	half/ tensor
P100	4.7	9.3	18.7
V100	7	14	112

AMD Radeon Instinct MI25 GPU (2017)



"24.6 TFLOPS FP16 or 12.3 TFLOPS FP32 peak GPU compute performance on a single board . . . Up to 82 GFLOPS/watt FP16 or 41 GFLOPS/watt FP32 peak GPU compute performance"

Low Precision in Machine Learning

Widespread use of low precision, for training *and* inference:

single precision (fp32)	32 bits
half precision (fp16)	16 bits
integer (INT8)	8 bits
ternary	$\{-1, 0, 1\}$
binary	$\{0, 1\}$

plus other newly-proposed floating-point formats.

- "We find that very low precision is sufficient not just for running trained networks but also for training them."
  Courbariaux, Benji & David (2015)
- No rigorous rounding error analysis exists (yet).
- Papers usually experimental, using particular data sets.

Why Does Low Precision Work in ML?

- We're solving the wrong problem (**Scheinberg**, 2016), so don't need an accurate solution.
- Low precision provides regularization.
- Low precision encourages flat minima to be found.

Deep Learning for Java

DL4J

Deep Learning Textbook

Download SKIL Community Edition

Request Corporate Training

Getting Started ›

Tutorials ›

Introduction to Deep Learning ›

Neural Networks ›

Data & ETL ›

HALF Datatype

If your app can afford using half-precision math (typically neural nets can afford this), you can enable this as data type for your app, and you'll see following benefits:

- 2x less GPU ram used
- up to 200% performance gains on memory-intensive operations, though the actual performance boost depends on the task and hardware used.

```
DataTypeUtil.setDTypeForContext(DataBuffer.Type.HALF);
```

Place this call as the first line of your app, so that all subsequent allocations/calculations will be done using the HALF data type.

However you should be aware: HALF data type offers way smaller precision then FLOAT or DOUBLE, thus neural net tuning might become way harder.

On top of that, at this moment we don't offer full LAPACK support for HALF data type.

Climate Modelling

- **T. Palmer**, **More reliable forecasts with less precise computations: a fast-track route to cloud-resolved weather and climate simulators?**, Phil. Trans. R. Soc. A, 2014:

  > "Is there merit in representing variables at sufficiently high wavenumbers using half or even quarter precision floating-point numbers?"

- **T. Palmer**, **Build imprecise supercomputers**, Nature, 2015.

Fp16 for Communication Reduction

ResNet-50 training on ImageNet.

- Solved in **60 mins on 256 TESLA P100s** at Facebook (2017).
- Solved in **15 mins on 1024 TESLA P100s** at Preferred Networks, Inc. (2017) using ChainerMN (**Takuya Akiba**, SIAM PP18):

Fp16 for Communication Reduction

ResNet-50 training on ImageNet.

- Solved in **60 mins on 256 TESLA P100s** at Facebook (2017).
- Solved in **15 mins on 1024 TESLA P100s** at Preferred Networks, Inc. (2017) using ChainerMN (**Takuya Akiba**, SIAM PP18):
  "While **computation was generally done in single precision, in order to reduce the communication overhead** during all-reduce operations, **we used half-precision floats** … In our preliminary experiments, we observed that the effect from using half-precision in communication on the final model accuracy was relatively small."

Preconditioning with Adaptive Precision

Anzt, Dongarra, Flegar, H & Quintana-Ortí (2018):

- For sparse A and iterative $Ax = b$ solver, execution time and energy dominated by **data movement**.
- Block Jacobi preconditioning: $D = \mathrm{diag}(D_i)$, where $D_i = A_{ii}$. Solve $D^{-1}Ax = D^{-1}b$.
- All computations are at **fp64**.
- Compute D^{-1} and store D_i^{-1} in **fp16, fp32** or **fp64**, depending on $\kappa(D_i)$.
- Simulations and energy modelling show can outperform fixed precision preconditioner.

Range Parameters

$r_{\min}^{(s)}$ = *smallest* positive (subnormal) number,
$r_{\min}$ = *smallest normalized* positive number,
$r_{\max}$ = *largest* finite number.

	$r_{\min}^{(s)}$	$r_{\min}$	$r_{\max}$
fp16	5.96×10^{-8}	6.10×10^{-5}	65504
fp32	1.40×10^{-45}	1.18×10^{-38}	3.4×10^{38}
fp64	4.94×10^{-324}	2.22×10^{-308}	1.80×10^{308}

Example: Vector 2-Norm in fp16

Evaluate $\|x\|_2$ for

$$x = \begin{bmatrix} \alpha \\ \alpha \end{bmatrix}$$

as $\sqrt{x_1^2 + x_2^2}$ in fp16.

Recall $u_h = 4.88 \times 10^{-4}$, $r_{\min} = 6.10 \times 10^{-5}$.

α	Relative error	Comment
10^{-4}	1	Underflow to 0
3.3×10^{-4}	4.7×10^{-2}	Subnormal range.
5.5×10^{-4}	7.1×10^{-3}	Subnormal range.
1.1×10^{-2}	1.4×10^{-4}	Perfect rel. err.

A Simple Loop

```
x = pi; i = 0;
while x/2 > 0
  x = x/2; i = i+1;
end
for k = 1:i
  x = 2*x;
end
```

Precision	i	$\lvert x - \pi \rvert$
Double	1076	0.858
Single	151	0.858
Half	26	0.858

A Simple Loop

```
x = pi; i = 0;
while x/2 > 0
   x = x/2; i = i+1;
end
for k = 1:i
   x = 2*x;
end
```

Precision	i	$\lvert x - \pi \rvert$
Double	1076	0.858
Single	151	0.858
Half	26	0.858

- Why these large errors?
- Why the same error for each precision?

Error Analysis in Low Precision (1)

For inner product $x^T y$ of n-vectors standard error bound is

$$|\,\mathrm{fl}(x^T y) - x^T y| \le \gamma_n |x|^T |y|, \qquad \gamma_n = \frac{nu}{1 - nu}, \quad nu < 1.$$

Can also be written as

$$|\,\mathrm{fl}(x^T y) - x^T y| \le nu|x|^T |y| + O(u^2).$$

In half precision, $u \approx 4.9 \times 10^{-4}$, so $nu = 1$ for $n = 2048$.

Error Analysis in Low Precision (1)

For inner product $x^T y$ of n-vectors standard error bound is

$$|\,\mathrm{fl}(x^T y) - x^T y| \le \gamma_n |x|^T |y|, \qquad \gamma_n = \frac{nu}{1 - nu}, \quad nu < 1.$$

Can also be written as

$$|\,\mathrm{fl}(x^T y) - x^T y| \le nu|x|^T |y| + O(u^2).$$

In half precision, $u \approx 4.9 \times 10^{-4}$, so $nu = 1$ for $n = 2048$.

What happens when $nu > 1$?

Error Analysis in Low Precision (2)

Rump & Jeannerod (2014) prove that in a number of standard rounding error bounds, $\gamma_n = nu/(1 - nu)$ **can be replaced by** nu **provided that round to nearest is used.**

Error Analysis in Low Precision (2)

Rump & Jeannerod (2014) prove that in a number of standard rounding error bounds, $\gamma_n = nu/(1 - nu)$ **can be replaced by** nu **provided that round to nearest is used.**

- Analysis nontrivial. Only a few core algs have been analyzed.
- Be'rr bound for $Ax = b$ is now $(3n - 2)u + (n^2 - n)u^2$ instead of γ_{3n}.
- Cannot replace γ_n by nu in all algs (e.g., pairwise summation).
- Once $nu \geq 1$ bounds cannot guarantee any accuracy, maybe not even a correct exponent!

Simulating fp16 Arithmetic

- **Simulation 1.**
  Converting operands to fp32 or fp64, carry out the operation in fp32 or fp64, then round the result back to fp16.
- **Simulation 2.**
  Scalar fp16 operations as in Simulation 1. Carry out matrix multiplication and matrix factorization in fp32 or fp64 then round the result back to fp16.

MATLAB fp16 Class (Moler)

Cleve Laboratory fp16 class uses Simulation 2 for `mtimes` (called in `lu`) and `mldivide`.

http://mathworks.com/matlabcentral/fileexchange/59085-cleve-laboratory

```
function z = plus(x,y)
   z = fp16(double(x) + double(y));
end
function z = mtimes(x,y)
    z = fp16(double(x) * double(y));
end
function z = mldivide(x,y)
   z = fp16(double(x) \ double(y));
end
function [L,U,p] = lu(A)
    [L,U,p] = lutx(A);
end
```

Is Simulation 2 Too Accurate?

- For matrix mult, standard error bound is

$$|C - \widehat{C}| \le \gamma_n |A||B|, \qquad \gamma_n = nu/(1 - nu).$$

  Error bound for Simulation 2 has **no n factor**.

- For triangular solves, $Tx = b$, error should be bounded by $\mathrm{cond}(T, x)\gamma_n$, but we actually get **error of order u**.

- Simulation 1 preferable. but too slow unless the problem is fairly small.
- Large operator overloading overheads in any language.