



C S C I 5304

Fall 2021

COMPUTATIONAL ASPECTS OF MATRIX THEORY

Class time : MW 4:00 – 5:15 pm
Room : Keller 3-230 or Online
Instructor : Daniel Boley

Lecture notes: http://www-users.cselabs.umn.edu/classes/Fall-2021/csci5304/

August 27, 2021

# FLOATING POINT ARITHMETIC - ERROR ANALYSIS

---

- Brief review of floating point arithmetic
- Model of floating point arithmetic
- Notation, backward and forward errors


## *Roundoff errors and floating-point arithmetic*

- The basic problem: The set  $A$  of all possible representable numbers on a given machine is finite - but we would like to use this set to perform standard arithmetic operations ( $+$ ,  $*$ ,  $-$ ,  $/$ ) on an infinite set. The usual algebra rules are no longer satisfied since results of operations are rounded.

- Basic algebra breaks down in floating point arithmetic.

**Example:** In floating point arithmetic.

$$a + (b + c) \neq (a + b) + c$$

 Matlab experiment: For 10,000 random numbers find number of instances when the above is true. Same thing for the multiplication..

## Floating point representation:

Real numbers are represented in two parts: A mantissa (significand) and an exponent. If the representation is in the base  $\beta$  then:

$$x = \pm (.d_1 d_2 \cdots d_t) \beta^e$$

- $.d_1 d_2 \cdots d_t$  is a fraction in the base- $\beta$  representation (Generally the form is normalized in that  $d_1 \neq 0$ ), and  $e$  is an integer
- Often, more convenient to rewrite the above as:

$$x = \pm (m / \beta^t) \times \beta^e \equiv \pm m \times \beta^{e-t}$$

- Mantissa  $m$  is an integer with  $0 \leq m \leq \beta^t - 1$ .

## Machine precision - machine epsilon

- Notation :  $fl(x)$  = closest floating point representation of real number  $x$  ('rounding')
- When a number  $x$  is very small, there is a point when  $1 + x == 1$  in a machine sense. The computer no longer makes a difference between 1 and  $1 + x$ .

**Machine epsilon:** The smallest number  $\epsilon$  such that  $1 + \epsilon$  is a float that is different from one, is called machine epsilon. Denoted by `macheps` or `eps`, it represents the distance from 1 to the next larger floating point number.

- With previous representation, `eps` is equal to  $\beta^{-(t-1)}$ .

**Example:** In IEEE standard double precision,  $\beta = 2$ , and  $t = 53$  (includes 'hidden bit'). Therefore  $\text{eps} = 2^{-52}$ .

**Unit Round-off** A real number  $x$  can be approximated by a floating number  $fl(x)$  with relative error no larger than  $\underline{u} = \frac{1}{2}\beta^{-(t-1)}$ .

- $\underline{u}$  is called Unit Round-off.
- In fact can easily show:

$$fl(x) = x(1 + \delta) \text{ with } |\delta| < \underline{u}$$



Matlab experiment: find the machine epsilon on your computer.

➤ Many discussions on what conditions/ rules should be satisfied by floating point arithmetic. The IEEE standard is a set of standards adopted by many CPU manufacturers.



**Rule 1.**


$$fl(x) = x(1 + \epsilon), \quad \text{where } |\epsilon| \leq \underline{u}$$

**Rule 2.** For all operations  $\odot$  (one of  $+$ ,  $-$ ,  $*$ ,  $/$ )

$$fl(x \odot y) = (x \odot y)(1 + \epsilon_{\odot}), \quad \text{where } |\epsilon_{\odot}| \leq \underline{u}$$

**Rule 3.** For  $+$ ,  $*$  operations

$$fl(a \odot b) = fl(b \odot a)$$

 **3** Matlab experiment: Verify experimentally Rule 3 with 10,000 randomly generated numbers  $a_i$ ,  $b_i$ .

**Example:** Consider the sum of 3 numbers:  $y = a + b + c$ .

➤ Done as  $fl(a + b + c) = fl(fl(a + b) + c)$

$$fl(a + b) = (a + b)(1 + \epsilon_1)$$

$$\begin{aligned} fl(a + b + c) &= [(a + b)(1 + \epsilon_1) + c](1 + \epsilon_2) \\ &= a(1 + \epsilon_1)(1 + \epsilon_2) + b(1 + \epsilon_1)(1 + \epsilon_2) \\ &\quad + c(1 + \epsilon_2) \\ &= a(1 + \theta_1) + b(1 + \theta_2) + c(1 + \theta_3) \end{aligned}$$

with  $1 + \theta_1 = 1 + \theta_2 = (1 + \epsilon_1)(1 + \epsilon_2)$  and  $1 + \theta_3 = (1 + \epsilon_2)$

For a longer sum we would have something like:

$$1 + \theta_j = (1 + \epsilon_1)(1 + \epsilon_2)(\cdots)(1 + \epsilon_{n-j})$$

➤ Remark on order of the sum. If  $y_1 = fl(fl(a + b) + c)$ :

$$\begin{aligned} y_1 &= [(a + b + c) + (a + b)\epsilon_1] (1 + \epsilon_2) \\ &= (a + b + c) \left[ 1 + \frac{a + b}{a + b + c} \epsilon_1 (1 + \epsilon_2) + \epsilon_2 \right] \end{aligned}$$

So disregarding the high order term  $\epsilon_1\epsilon_2$

$$\begin{aligned} fl(fl(a + b) + c) &= (a + b + c)(1 + \epsilon_3) \\ \epsilon_3 &\approx \frac{a + b}{a + b + c} \epsilon_1 + \epsilon_2 \end{aligned}$$

- If we redid the computation as  $y_2 = fl(a + fl(b + c))$  we would find

$$fl(a + fl(b + c)) = (a + b + c)(1 + \epsilon_4)$$
$$\epsilon_4 \approx \frac{b + c}{a + b + c} \epsilon_1 + \epsilon_2$$

- The error is amplified by the factor  $(a + b)/y$  in the first case and  $(b + c)/y$  in the second case.
- In order to sum  $n$  numbers accurately, it is better to start with small numbers first. [However, sorting before adding is not worth it.]
- But watch out if the numbers have mixed signs!

## The absolute value notation

- For a given vector  $x$ ,  $|x|$  is the vector with components  $|x_i|$ , i.e.,  $|x|$  is the component-wise absolute value of  $x$ .
- Similarly for matrices:

$$|A| = \{|a_{ij}|\}_{i=1,\dots,m; j=1,\dots,n}$$

- An obvious result: The basic inequality

$$|fl(a_{ij}) - a_{ij}| \leq \underline{u} |a_{ij}|$$

translates into

$$|fl(A) - A| \leq \underline{u} |A|$$

- $A \leq B$  means  $a_{ij} \leq b_{ij}$  for all  $1 \leq i \leq m; 1 \leq j \leq n$

## Backward and forward errors

- Assume the approximation  $\hat{y}$  to  $y = \text{alg}(x)$  is computed by some algorithm with arithmetic precision  $\epsilon$ . Possible analysis: find an upper bound for the **Forward** error

$$|\Delta y| = |y - \hat{y}|$$

- This is not always easy.

**Alternative question:** find equivalent perturbation on initial data ( $x$ ) that produces the result  $\hat{y}$ . In other words, find  $\Delta x$  so that:

$$\text{alg}(x + \Delta x) = \hat{y}$$

- The value of  $|\Delta x|$  is called the backward error. An analysis to find an upper bound for  $|\Delta x|$  is called **Backward error analysis**.

**Example:**

$$A = \begin{pmatrix} a & b \\ 0 & c \end{pmatrix} \quad B = \begin{pmatrix} d & e \\ 0 & f \end{pmatrix}$$

Consider the product:  $fl(A.B) =$

$$\left[ \begin{array}{c|c} ad(1 + \epsilon_1) & [ae(1 + \epsilon_2) + bf(1 + \epsilon_3)](1 + \epsilon_4) \\ \hline 0 & cf(1 + \epsilon_5) \end{array} \right]$$

with  $\epsilon_i \leq \underline{u}$ , for  $i = 1, \dots, 5$ . Result can be written as:

$$\left[ \begin{array}{c|c} a & b(1 + \epsilon_3)(1 + \epsilon_4) \\ \hline 0 & c(1 + \epsilon_5) \end{array} \right] \left[ \begin{array}{c|c} d(1 + \epsilon_1) & e(1 + \epsilon_2)(1 + \epsilon_4) \\ \hline 0 & f \end{array} \right]$$

➤ So  $fl(A.B) = (A + E_A)(B + E_B)$ .

➤ Backward errors  $E_A, E_B$  satisfy:

$$|E_A| \leq 2\underline{u} |A| + O(\underline{u}^2) ; \quad |E_B| \leq 2\underline{u} |B| + O(\underline{u}^2)$$

➤ When solving  $Ax = b$  by Gaussian Elimination, we will see that a bound on  $\|e_x\|$  such that this holds exactly:

$$A(x_{\text{computed}} + e_x) = b$$

is much harder to find than bounds on  $\|E_A\|$ ,  $\|e_b\|$  such that this holds exactly:

$$(A + E_A)x_{\text{computed}} = (b + e_b).$$

**Note:** In many instances backward errors are more meaningful than forward errors: if initial data is accurate only to 4 digits say, then my algorithm for computing  $x$  need not **guarantee a backward error of less than  $10^{-10}$  for example.** A backward error of order  $10^{-4}$  is acceptable.



## *Error Analysis: Inner product*

- Inner products are in the innermost parts of many calculations. Their analysis is important.

*Lemma:* If  $|\delta_i| \leq \underline{u}$  and  $n\underline{u} < 1$  then

$$\prod_{i=1}^n (1 + \delta_i) = 1 + \theta_n \quad \text{where} \quad |\theta_n| \leq \frac{n\underline{u}}{1 - n\underline{u}}$$

- Common notation  $\gamma_n \equiv \frac{n\underline{u}}{1 - n\underline{u}}$

 Prove the lemma [Hint: use induction]

➤ Can use the following simpler result:

*Lemma:* If  $|\delta_i| \leq \underline{u}$  and  $n\underline{u} < .01$  then

$$\prod_{i=1}^n (1 + \delta_i) = 1 + \theta_n \quad \text{where} \quad |\theta_n| \leq 1.01n\underline{u}$$

**Example:**

Previous sum of numbers can be written

$$\begin{aligned} fl(a + b + c) &= fl(fl(a + b) + c) \\ &= [(a + b)(1 + \epsilon_1) + c](1 + \epsilon_2) \\ &= a(1 + \epsilon_1)(1 + \epsilon_2) + b(1 + \epsilon_1)(1 + \epsilon_2) + \\ &\quad c(1 + \epsilon_2) \\ &= a(1 + \theta_1) + b(1 + \theta_2) + c(1 + \theta_3) \\ &= \text{exact sum of slightly perturbed inputs,} \end{aligned}$$

where all  $\theta_i$ 's satisfy  $|\theta_i| \leq 1.01n\underline{u}$  (here  $n = 2$ ).

➤ **Backward** error result (output is exact sum of perturbed input)

➤ Alternatively, can write '**forward**' bound:

$$|fl(a + b + c) - (a + b + c)| \leq |a\theta_1| + |b\theta_2| + |c\theta_3|.$$

(bound on | output - exact sum | )

## Analysis of inner products (cont.)

Consider

$$s_n = fl(x_1 * y_1 + x_2 * y_2 + \cdots + x_n * y_n)$$

- In what follows  $\eta_i$ 's come from  $*$ ,  $\epsilon_i$ 's come from  $+$
- They satisfy:  $|\eta_i| \leq \underline{u}$  and  $|\epsilon_i| \leq \underline{u}$ .
- The inner product  $s_n$  is computed as:
  1.  $s_1 = fl(x_1 y_1) = (x_1 y_1)(1 + \eta_1)$

$$\begin{aligned} 2. \quad s_2 &= fl(s_1 + fl(x_2y_2)) = fl(s_1 + x_2y_2(1 + \eta_2)) \\ &= (x_1y_1(1 + \eta_1) + x_2y_2(1 + \eta_2))(1 + \epsilon_2) \\ &= x_1y_1(1 + \eta_1)(1 + \epsilon_2) + x_2y_2(1 + \eta_2)(1 + \epsilon_2) \end{aligned}$$

$$\begin{aligned} 3. \quad s_3 &= fl(s_2 + fl(x_3y_3)) = fl(s_2 + x_3y_3(1 + \eta_3)) \\ &= (s_2 + x_3y_3(1 + \eta_3))(1 + \epsilon_3) \end{aligned}$$

Expand:

$$\begin{aligned} s_3 &= x_1y_1(1 + \eta_1)(1 + \epsilon_2)(1 + \epsilon_3) \\ &\quad + x_2y_2(1 + \eta_2)(1 + \epsilon_2)(1 + \epsilon_3) \\ &\quad + x_3y_3(1 + \eta_3)(1 + \epsilon_3) \end{aligned}$$

➤ Induction would show that [with convention that  $\epsilon_1 \equiv 0$ ]

$$s_n = \sum_{i=1}^n x_i y_i (1 + \eta_i) \prod_{j=i}^n (1 + \epsilon_j)$$

**Q:** How many terms in the coefficient of  $x_i y_i$  do we have?

- A:**
- When  $i > 1$  :  $1 + (n - i + 1) = n - i + 2$
  - When  $i = 1$  :  $n$  (since  $\epsilon_1 = 0$  does not count)

➤ Bottom line: always  $\leq n$ .

➤ For each of these products

$$(1 + \eta_i) \prod_{j=i}^n (1 + \epsilon_j) = 1 + \theta_i, \quad \text{with } |\theta_i| \leq \gamma_n \quad \text{so:}$$

$$s_n = \sum_{i=1}^n x_i y_i (1 + \theta_i) \quad \text{with } |\theta_i| \leq \gamma_n \quad \text{or:}$$

$$\boxed{fl \left( \sum_{i=1}^n x_i y_i \right) = \sum_{i=1}^n x_i y_i + \sum_{i=1}^n x_i y_i \theta_i \quad \text{with } |\theta_i| \leq \gamma_n}$$

➤ This leads to the final result (forward form)

$$\left| fl \left( \sum_{i=1}^n x_i y_i \right) - \sum_{i=1}^n x_i y_i \right| \leq \gamma_n \sum_{i=1}^n |x_i| |y_i|$$

➤ or (backward form)

$$fl \left( \sum_{i=1}^n x_i y_i \right) = \sum_{i=1}^n x_i y_i (1 + \theta_i) \quad \text{with } |\theta_i| \leq \gamma_n$$



## Main result on inner products:

- Backward error expression:

$$fl(x^T y) = [x .* (1 + d_x)]^T [y .* (1 + d_y)]$$

where  $\|d_{\square}\|_{\infty} \leq 1.01n\underline{u}$ ,  $\square = x, y$ .

- Can show equality valid even if one of the  $d_x, d_y$  absent.

- Forward error expression:  $|fl(x^T y) - x^T y| \leq \gamma_n |x|^T |y|$

with  $0 \leq \gamma_n \leq 1.01n\underline{u}$ .

- Elementwise absolute value  $|x|$  and multiply  $.*$  notation.

- Above assumes  $n\underline{u} \leq .01$ .


For  $\underline{u} = 2.0 \times 10^{-16}$ , this holds for  $n \leq 4.5 \times 10^{13}$ .


- Consequence for matrix products: ( $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times p}$ )

$$|fl(AB) - AB| \leq \gamma_n |A||B|$$

- Another way to write the result (less precise) is


$$|fl(x^T y) - x^T y| \leq n \underline{u} |x|^T |y| + O(\underline{u}^2)$$

5 Assume you use single precision for which you have  $\underline{u} = 2. \times 10^{-6}$ . What is the largest  $n$  for which  $n\underline{u} \leq 0.01$  holds? Any conclusions for the use of single precision arithmetic?


6 What does the main result on inner products imply for the case when  $y = x$ ? [Contrast the relative accuracy you get in this case vs. the general case when  $y \neq x$ ]

7 Show for any  $x, y$ , there exist  $\Delta x, \Delta y$  such that

$$\begin{aligned} fl(x^T y) &= (x + \Delta x)^T y, & \text{with } |\Delta x| &\leq \gamma_n |x| \\ fl(x^T y) &= x^T (y + \Delta y), & \text{with } |\Delta y| &\leq \gamma_n |y| \end{aligned}$$

8 (Continuation) Let  $A$  an  $m \times n$  matrix,  $x$  an  $n$ -vector, and  $y = Ax$ . Show that there exist a matrix  $\Delta A$  such

$$fl(y) = (A + \Delta A)x, \quad \text{with } |\Delta A| \leq \gamma_n |A|$$

9 (Continuation) From the above derive a result about a column of the product of two matrices  $A$  and  $B$ . Does a similar result hold for the product  $AB$  as a whole?

## *Error Analysis for linear systems: Triangular case*

### ➤ Recall

### ALGORITHM : 1. *Back-Substitution algorithm*

*For*  $i = n : -1 : 1$  *do*:

$t := b_i$

*For*  $j = i + 1 : n$  *do*

$t := t - a_{ij}x_j$

*End*

$x_i = t/a_{ii}$

*End*

}  $t := t - (a_{i,i+1:n}, x_{i+1:n})$   
=  $t - \text{an inner product}$

➤ We must require that each  $a_{ii} \neq 0$

➤ Round-off error (use previous results for  $(\cdot, \cdot)$ )?

The computed solution  $\hat{x}$  of the triangular system  $Ux = b$  computed by the back-substitution algorithm satisfies:

$$(U + E)\hat{x} = b$$

with

$$|E| \leq n \underline{u} |U| + O(\underline{u}^2)$$

- Backward error analysis. Computed  $x$  solves a slightly perturbed system.
- Backward error not large in general. It is said that triangular solve is “backward stable”.

## *Error Analysis for Gaussian Elimination*

If no zero pivots are encountered during Gaussian elimination (no pivoting) then the computed factors  $\hat{L}$  and  $\hat{U}$  satisfy

$$\hat{L}\hat{U} = A + H$$

with

$$|H| \leq 3(n-1) \times \underline{u} (|A| + |\hat{L}| |\hat{U}|) + O(\underline{u}^2)$$

Solution  $\hat{x}$  computed via  $\hat{L}\hat{y} = b$  and  $\hat{U}\hat{x} = \hat{y}$  is s. t.

$$(A + E)\hat{x} = b \text{ with}$$

$$|E| \leq n\underline{u} (3|A| + 5|\hat{L}| |\hat{U}|) + O(\underline{u}^2)$$

- “Backward” error estimate.
- $|\hat{L}|$  and  $|\hat{U}|$  are not known in advance – they can be large.
- What if partial pivoting is used?
- Permutations introduce no errors. Equivalent to standard LU factorization on matrix  $PA$ .
- $|\hat{L}|$  is small since  $l_{ij} \leq 1$ . Therefore, only  $U$  is “uncertain”
- In practice partial pivoting is “stable” – i.e., it is highly unlikely to have a very large  $U$ .



## Supplemental notes: Floating Point Arithmetic

In most computing systems, real numbers are represented in two parts: A mantissa and an exponent. If the representation is in the base  $\beta$  then:

$$x = \pm (.d_1 d_2 \cdots d_m)_{\beta} \beta^e$$

- $.d_1 d_2 \cdots d_m$  is a fraction in the base- $\beta$  representation
- $e$  is an integer - can be negative, positive or zero.
- Generally the form is normalized in that  $d_1 \neq 0$ .

**Example:** In base 10 (for illustration)

1. 1000.12345 can be written as

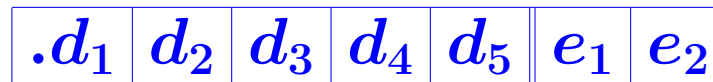
$$0.100012345_{10} \times 10^4$$

2. 0.000812345 can be written as

$$0.812345_{10} \times 10^{-3}$$

➤ Problem with floating point arithmetic: we have to live with limited precision.

**Example:** Assume that we have only 5 digits of accuracy in the mantissa and 2 digits for the exponent (excluding sign).



Try to add  $1000.2 = .10002e+03$  and  $1.07 = .10700e+01$ :

$$1000.2 = \boxed{.1} \boxed{0} \boxed{0} \boxed{0} \boxed{2} \boxed{0} \boxed{4} ; \quad 1.07 = \boxed{.1} \boxed{0} \boxed{7} \boxed{0} \boxed{0} \boxed{0} \boxed{1}$$

**First task:** align decimal points. The one with smallest exponent will be (internally) rewritten so its exponent matches the largest one:

$$1.07 = 0.000107 \times 10^4$$

**Second task:** add mantissas:

$$\begin{array}{r} 0.10002 \\ + 0.000107 \\ \hline = 0.100127 \end{array}$$

### *Third task:*

round result. Result has 6 digits - can use only 5 so we can

➤ Chop result: 

.1	0	0	1	2
----	---	---	---	---

 ;

➤ Round result: 

.1	0	0	1	3
----	---	---	---	---

 ;

### *Fourth task:*

Normalize result if needed (not needed here)

result with rounding: 

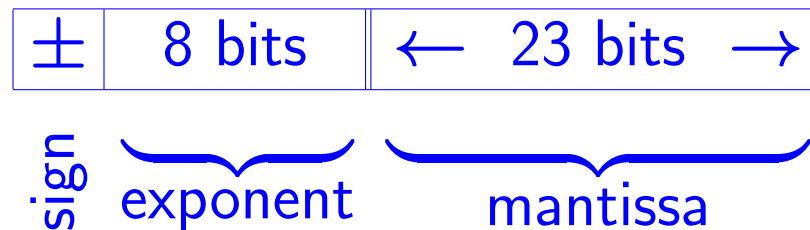
.1	0	0	1	3	0	4
----	---	---	---	---	---	---

 ;

10 Redo the same thing with  $7000.2 + 4000.3$  or  $6999.2 + 4000.3$ .

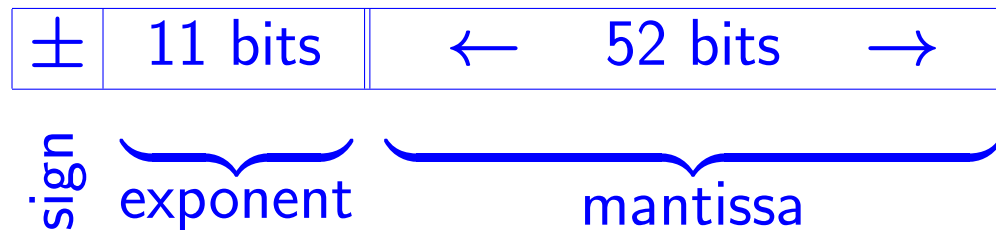
## The IEEE standard

**32 bit** (Single precision) :




- Number is scaled so it is in the form  $1.d_1d_2\dots d_{23} \times 2^e$  - but leading one is not represented.
- $e$  is between -126 and 127.
- [Here is why: Internally, exponent  $e$  is represented in “biased” form: what is stored is actually  $c = e + 127$  - so the value  $c$  of exponent field is between 1 and 254. The values  $c = 0$  and  $c = 255$  are for special cases (0 and  $\infty$ )]

**64 bit** (Double precision) :



- Bias of 1023 so if  $e$  is the actual exponent the content of the exponent field is  $c = e + 1023$
- Largest exponent: **1023**; Smallest = -1022.
- $c = 0$  and  $c = 2047$  (all ones) are again for 0 and  $\infty$
- Including the hidden bit, mantissa has total of 53 bits (52 bits represented, one hidden).
- In single precision, mantissa has total of 24 bits (23 bits represented, one hidden).

 Take the number 1.0 and see what will happen if you add  $1/2, 1/4, \dots, 2^{-i}$ . Do not forget the hidden bit!

Hidden bit (Not represented)

Expon. ↓ ← 52 bits →

e	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
e	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
e	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
.....															
e	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
e	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(Note: The 'e' part has 12 bits and includes the sign)

➤ Conclusion

$$fl(1 + 2^{-52}) \neq 1 \text{ but: } fl(1 + 2^{-53}) == 1 !!$$

## *Special Values*

- Exponent field = 0000000000 (smallest possible value)  
No hidden bit. All bits == 0 means exactly zero.
- Allow for unnormalized numbers,  
leading to gradual underflow.
- Exponent field = 1111111111 (largest possible value)  
Number represented is "Inf" "-Inf" or "NaN".



## *Recent trend: GPUs*

- Graphics Processor Units: Very fast boards attached to CPUs for heavy-duty computing
- e.g., NVIDIA V100 can deliver 112 Teraflops (1 Teraflops =  $10^{12}$  operations per second) for certain types of computations.
- Single precision much faster than double ...
- ... and there is also “half-precision” which is  $\approx 16$  times faster than standard 64bit arithmetic
- Used primarily for Deep-learning