A short introduction to the mixed precision paradigm

Matthieu Robeyns

SéPaG 2023, June 7, 2023

Floating Point Arithmetic

- **Floating point arithmetic** is a way to represent real numbers in a computer.
- It is based on the **scientific notation**:

$$\pm 0.d_1 d_2 d_3 \ldots d_p \times \beta^e$$

=> The **IEEE 754** encoding gives:

$$\text{value} = \text{sign} \times \left(\sum_{n=0}^{p-1} \mathbb{1}_n \times 2^{-n} \right) \times 2^{e - biais}$$

- biais : $2^{\text{size}_e - 1} - 1$
- sign : $(-1)^{val_p}$

Exemple of IEEE 754 in Single Precision (FP32)

0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1
0	0	0	0	1	1	1	1	1	1	0	1	1	0	1	1

$$
\begin{aligned}
&(-1)^0(1 + 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} + \cdots + 1 \times 2^{-23}) \times 2^1 \\
\simeq &(1 + 0.5 + 0 + 0 + 0.0675 + \cdots + 1.1920929e{-7}) \times 2 \\
\simeq &1.5707964 \times 2 \\
\simeq &3.1415928
\end{aligned}
$$

Exemple of IEEE 754 in Single Precision (FP16)

0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	0

$$
\begin{aligned}
&(-1)^0(1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} + \dots \\
&+ 0 \times 2^{-10}) \times 2^1 \\
\simeq&(1 + 0.5 + 0 + 0 + 0.0625 + \dots + 0) \times 2 \\
\simeq&1.571 \times 2 \\
\simeq&3.142
\end{aligned}
$$

Goal

- Speed up time computation.

Goal

- Speed up time computation.
- Use low precision floating point arithmetic.

Goal

- Speed up time computation.
- Use low precision floating point arithmetic.
- Use MPFR library to simulate low precision.

Goal

- Speed up time computation.
- Use low precision floating point arithmetic.
- Use MPFR library to simulate low precision.
- Use high precision for fast computation to stay accurate.

Algorithm with high precision

Algorithm 1 Algorithm with high precision

Require: X : A target object to approximate.
Ensure: Y : An approximation of X.
1: Compute $F = \textsc{LongComputation}(X)$ in u_{high}.
2: Compute $Y = \textsc{ShortComputation}(F)$ in u_{high}.

- + **Accuracy** : $o(u_{high})$.
- - **Speed** : $flops(\textsc{LongComputation}_{u_{high}})$.

Algorithm with low precision

Algorithm 2 Algorithm with low precision

Require: X : A target object to approximate.
Ensure: Y : An approximation of X.
1: Compute $F = \textsc{LongComputation}(X)$ in u_{low}.
2: Compute $Y = \textsc{ShortComputation}(F)$ in u_{low}.

- **Accuracy** : $o(u_{low})$.
+ **Speed** : $flops(\textsc{LongComputation}_{u_{low}})$.

Algorithm with mixed precision

Algorithm 3 Algorithm with mixed precision

Require: X : A target object to approximate.

Ensure: Y : An approximation of X.

1: Compute $F = \textsc{LongComputation}(X)$ in u_{low}.

2: Compute $Y = \textsc{ShortComputation}(F)$ in u_{high}.

- **Accuracy** : $o(u_{high})$.
- **Speed** : $flops(\textsc{LongComputation}_{u_{low}})$.

MPFR issues

- *mpfr_set_default_prec*(p) : set all following object at the precision p.
- *mpfr_set_prec*(X, p) : set the precision of X at p.

MPFR issues

- *mpfr_set_default_prec*(p) : set all following object at the precision p.
- *mpfr_set_prec*(X, p) : set the precision of X at p.

Algorithm 5 Algorithm with mixed precision with MPFR

Require: X : A target object to approximate.

Ensure: Y : An approximation of X.

1: *mpfr_set_default_prec*(p_h).

2: Compute F = LongComputation(*mpfr_set_prec*(X, p_l)) in u_{low}.

3: Compute Y = ShortComputation(F) in u_{high}.

Conclusion

- Change the view of the problem.
- Use low precision floating point arithmetic.
- Don't stay at the same precision.
- Be careful with acuracy loss.

Thank you for your attention.