Jiman Hwang,
Svetlana Selivanova,
Martin Ziegler

Happy Birthday, iRRAM!

Considerations for the Future of *Exact Real Computation*

Exact Real Computation:

KAIST
Complexity and
Real Computation

Continuous Abstract Data Types

- real numbers,
  vectors/matrices,
  polynomials,
  subspaces (*Seokbin Lee*)
- sequences,
  power series,
  analytic functions (*Holger Thies*)
- continuous/smooth/integrable functions
- compact Euclidean domains
- bounded operators

to agree with structures from Calculus I~IV

exactly:

no rounding errors!

Modularize
Exact Real Computation

KAIST
Complexity and
Real Computation

`BigInt, bignum, ℤ`

Exact Real Computation
(User Program/Space)

FFT/
Strassen-
Schönhage

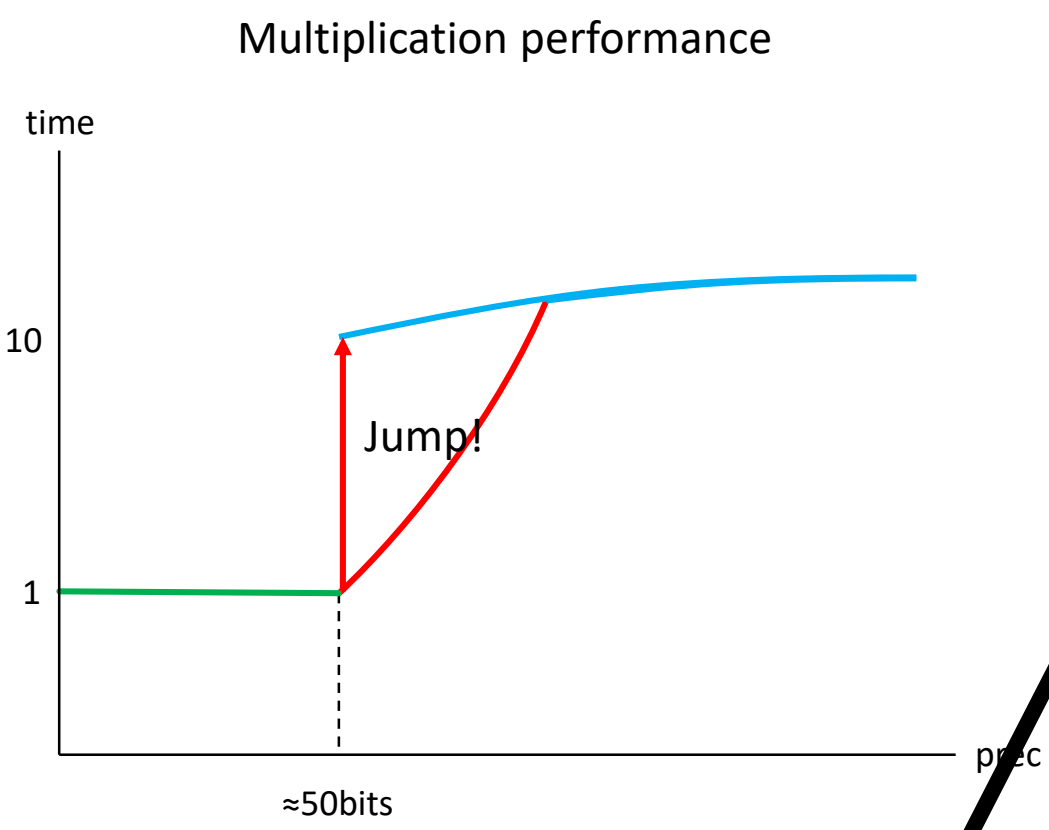
Engine: execute/simulate user code with finite precision sufficient to "behave" as exact/infinite precision

`realLib` **`iRRAM`** **`core2`**

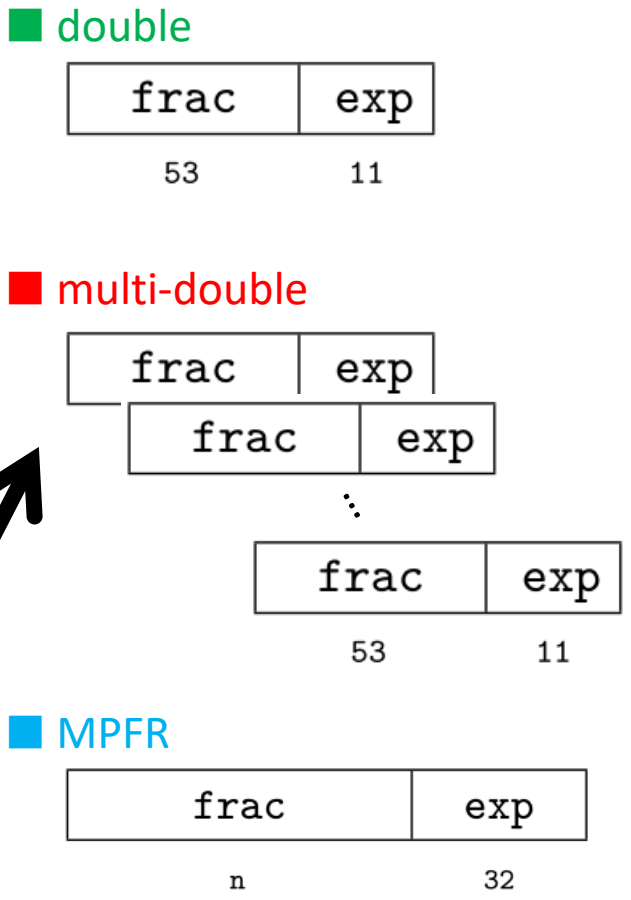
`qwords`
(64bit)

hardware (FPGA/ASIC)

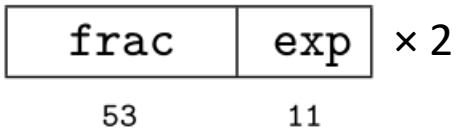
Goal



Goal: Smoothening transition with



Double-double



- Example

$$a = 0.\boxed{10110}\boxed{1010} \quad \text{(assuming 4-bit prec.)}$$

$$= 1.011 \times 2^{-1} + 1.010 \times 2^{-6}$$

- All operations are component-wise

  ex) multiplication

$$\begin{array}{rrrrr} & & (\ 1.010 \times 2^{0} & + & 1.000 \times 2^{-4}\ ) \\ \times & & (\ 1.000 \times 2^{1} & + & 1.000 \times 2^{-4}\ ) \\ \hline & & 1.010 \times 2^{-4} & + & 1.000 \times 2^{-8} \\ 1.010 \times 2^{1} & + & 1.000 \times 2^{-3} & & \\ \hline 1.010 \times 2^{1} & + & 1.101 \times 2^{-3} & + & \cancel{1.000 \times 2^{-8}} \end{array}$$

- Advantage: hardware support → fast at low precision
- Disadvantage: lack of algorithms → slow at high precision

Experiments

1. Matrix multiplication AB
and
Polynomial multiplication $P(x)Q(x)$

with each element/coefficient
double-double vs MPFR
(103-bit) (≈103-bit)

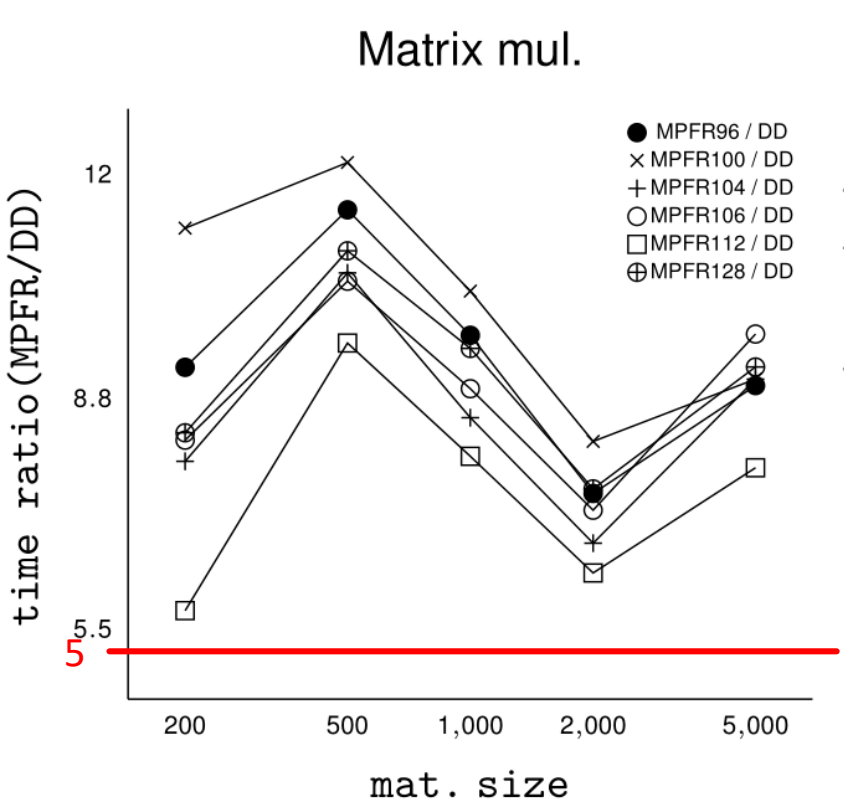
2. Matrix multiplication AB
and
Polynomial multiplication $P(x)Q(x)$

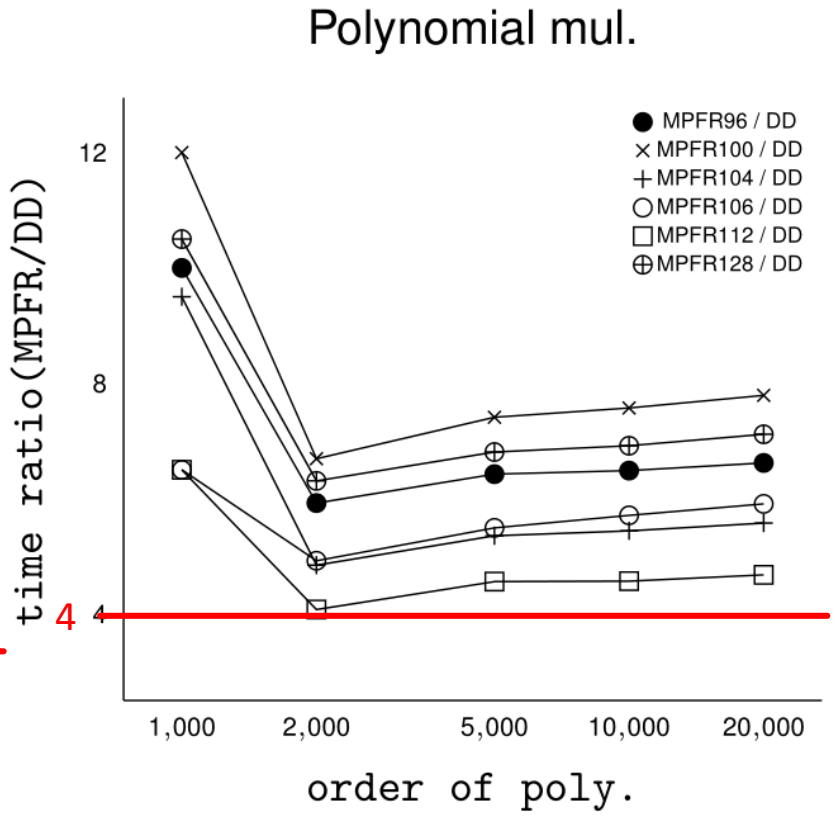
with each element/coefficient
quad-double vs MPFR
(209-bit) (≈209-bit)

- Why mat. and poly.? Frequent use in science and engineering
- Both used long multiplication and *QD library*[1]

[1] David H. Bailey, https://www.davidhbailey.com/dhbsoftware/

Result 1: Double-double(DD) vs MPFR

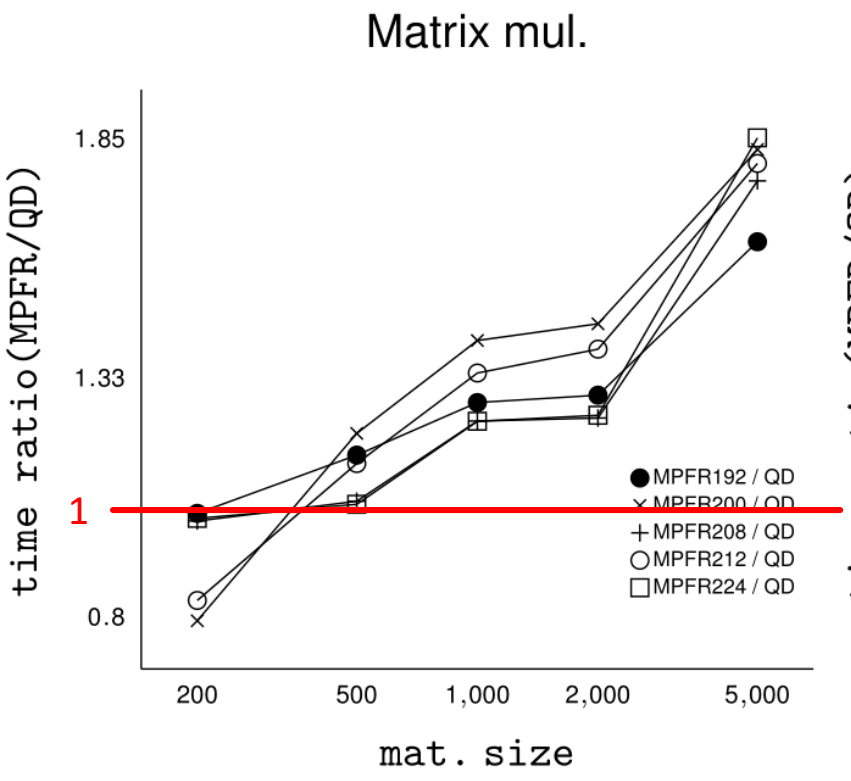


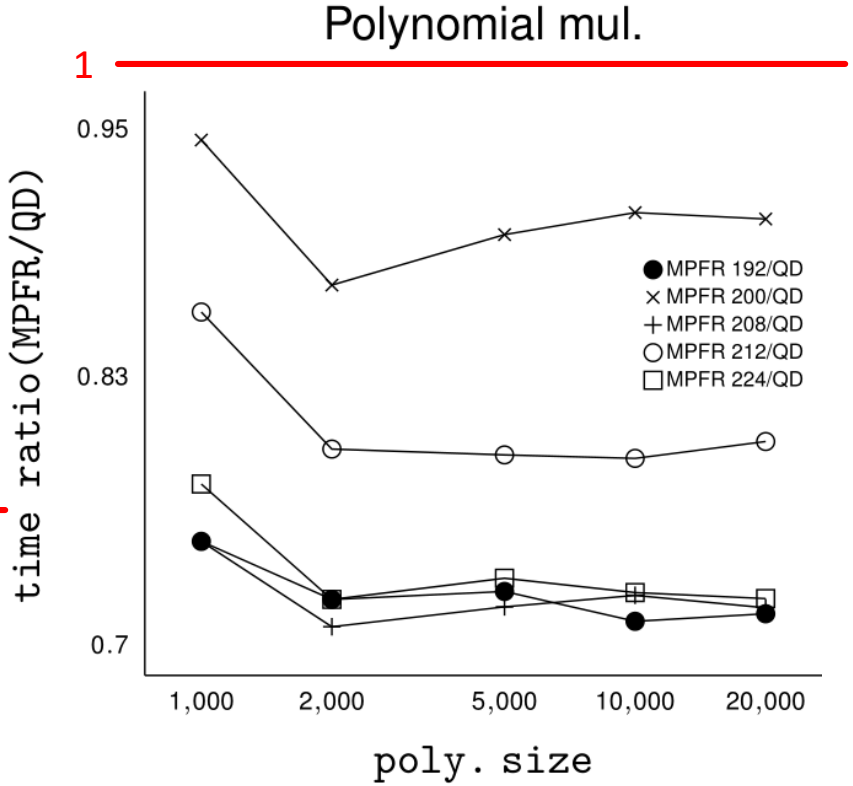


- $(\text{ratio}) = \dfrac{(\text{calc. time by MPFR})}{(\text{calc. time by DD})}$ → the greater is, the faster double-double is

- Double-double runs at least 4 times faster.

Result 2: Quad-double(QD) vs MPFR



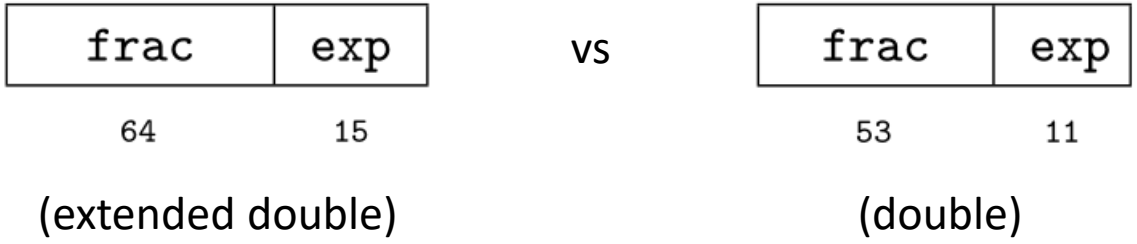


- $(\text{ratio}) = \dfrac{(\text{calc. time by MPFR})}{(\text{calc. time by QD})}$ → the greater is, the faster quad-double is

- No feasible gain

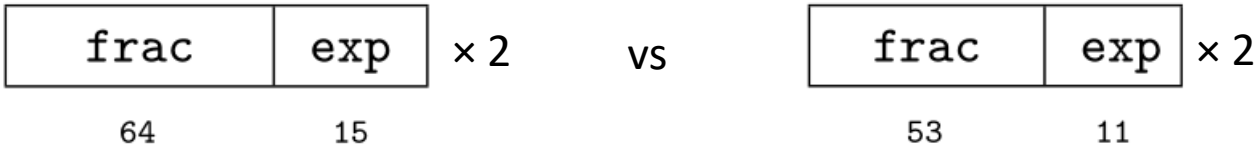
Extended Double-Double(EDD)

- x87 chipset provides extended double.



(extended double) (double)

- Extended double-double vs Double-double?



(125-bit) (103-bit)

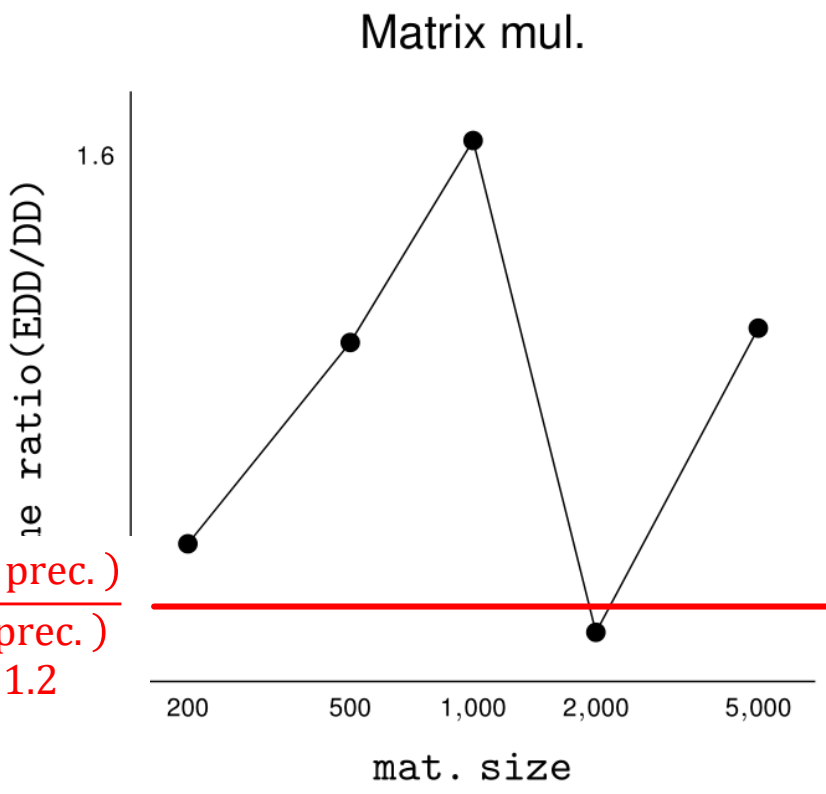
Double-Double(DD) vs Extended Double-Double(EDD)

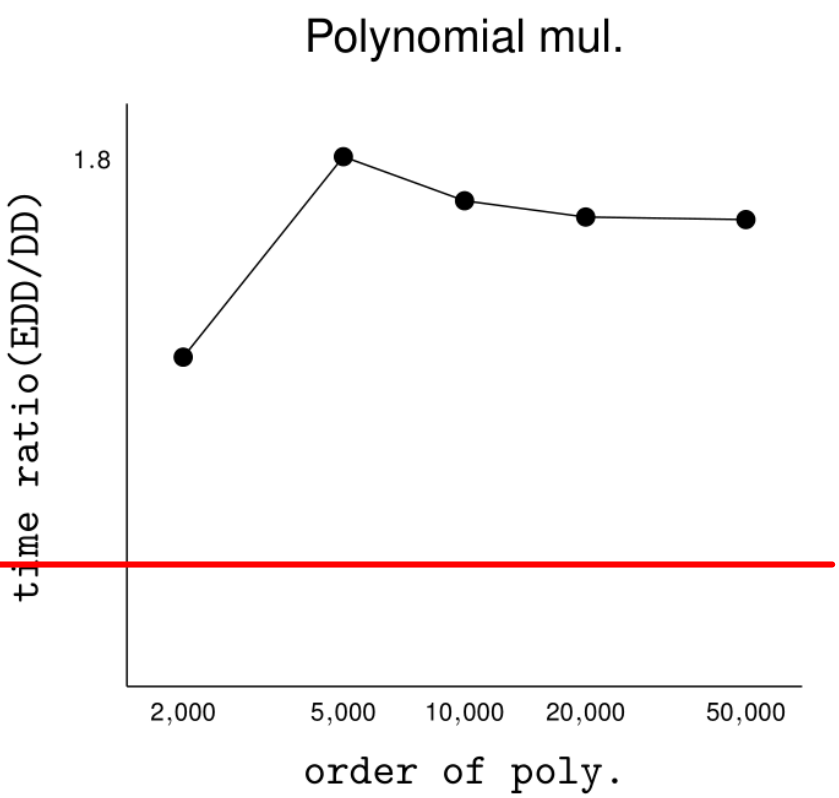
$$\left(\frac{\text{EDD time}}{\text{DD time}}\right) > \left(\frac{\text{EDD prec.}}{\text{DD prec.}}\right)$$

Cost Gain

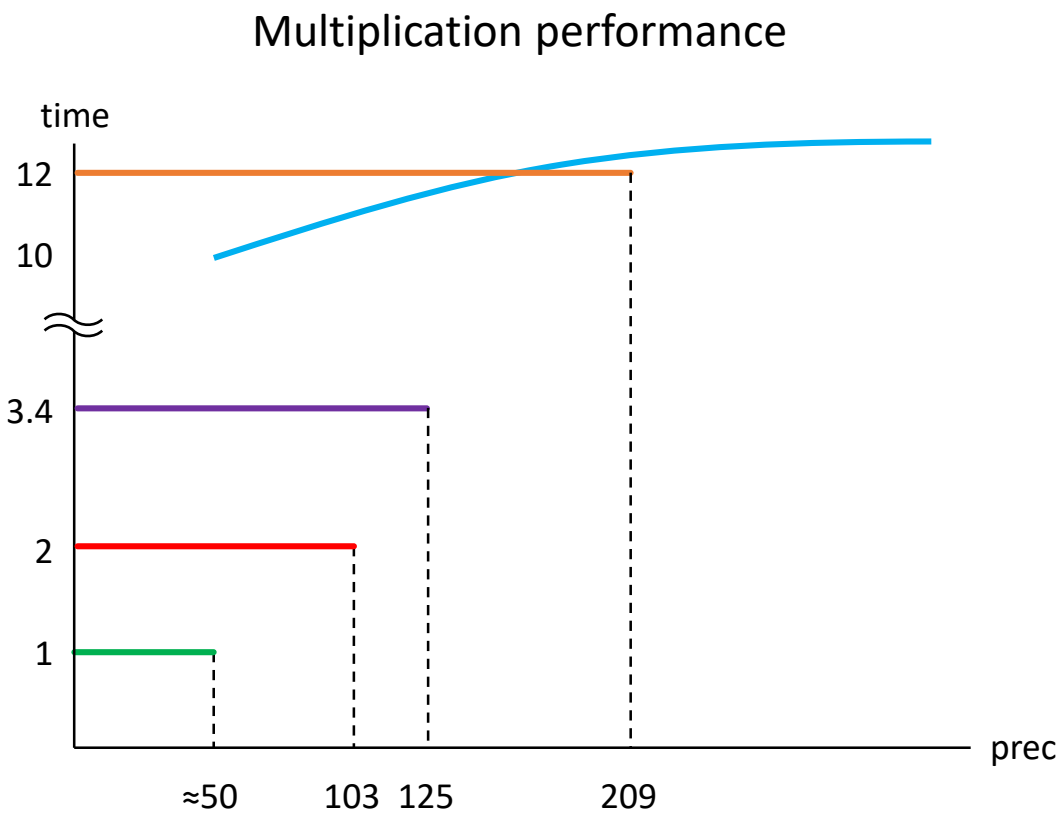
→ EDD is not efficient than double-double.

$$\frac{(\text{EDD prec.})}{(\text{DD prec.})} \approx 1.2$$

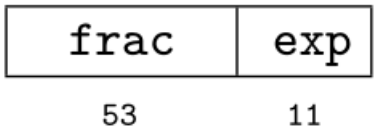




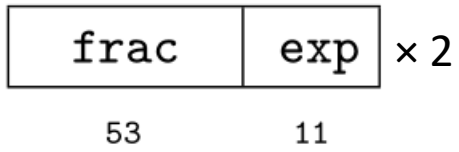
Conclusion



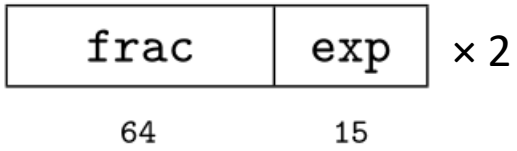
■ double



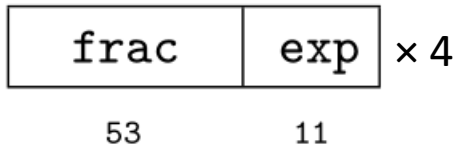
■ double-double



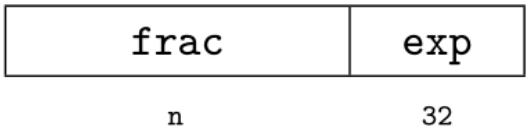
■ extended double-double



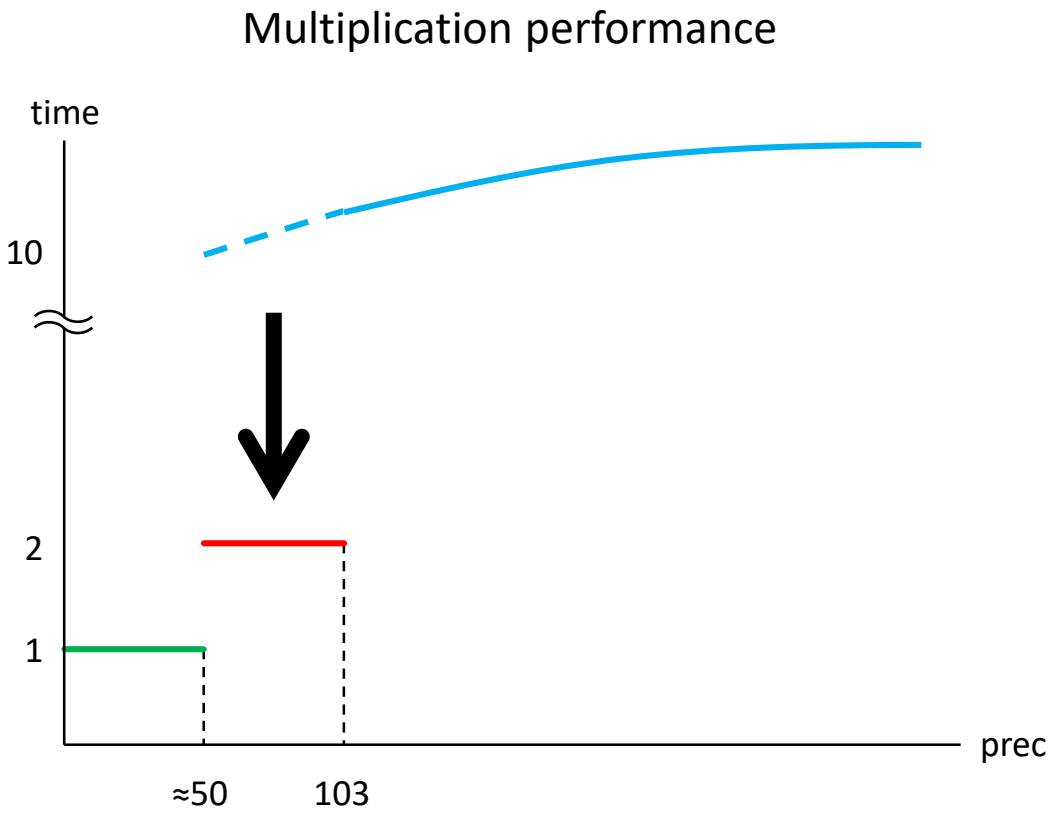
■ quad-double



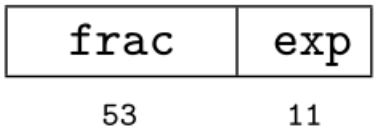
■ MPFR



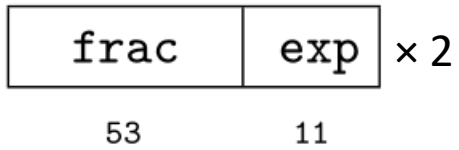
Future work



■ double



■ double-double



■ MPFR