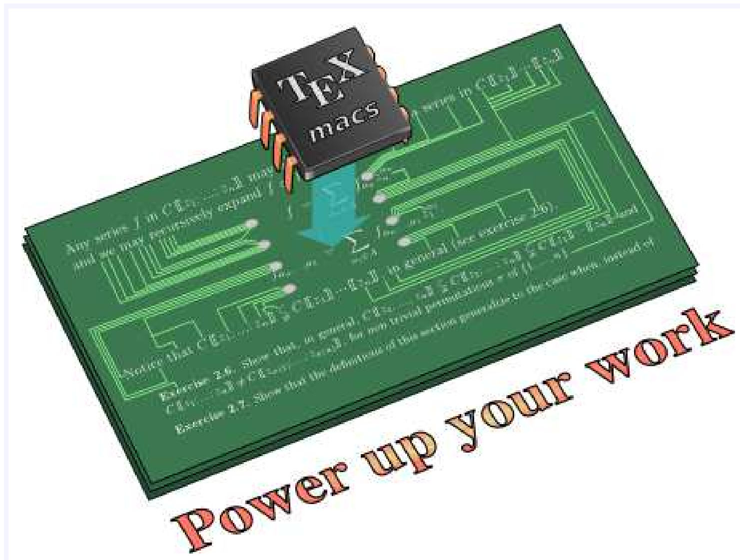
Faster FFTs in medium precision

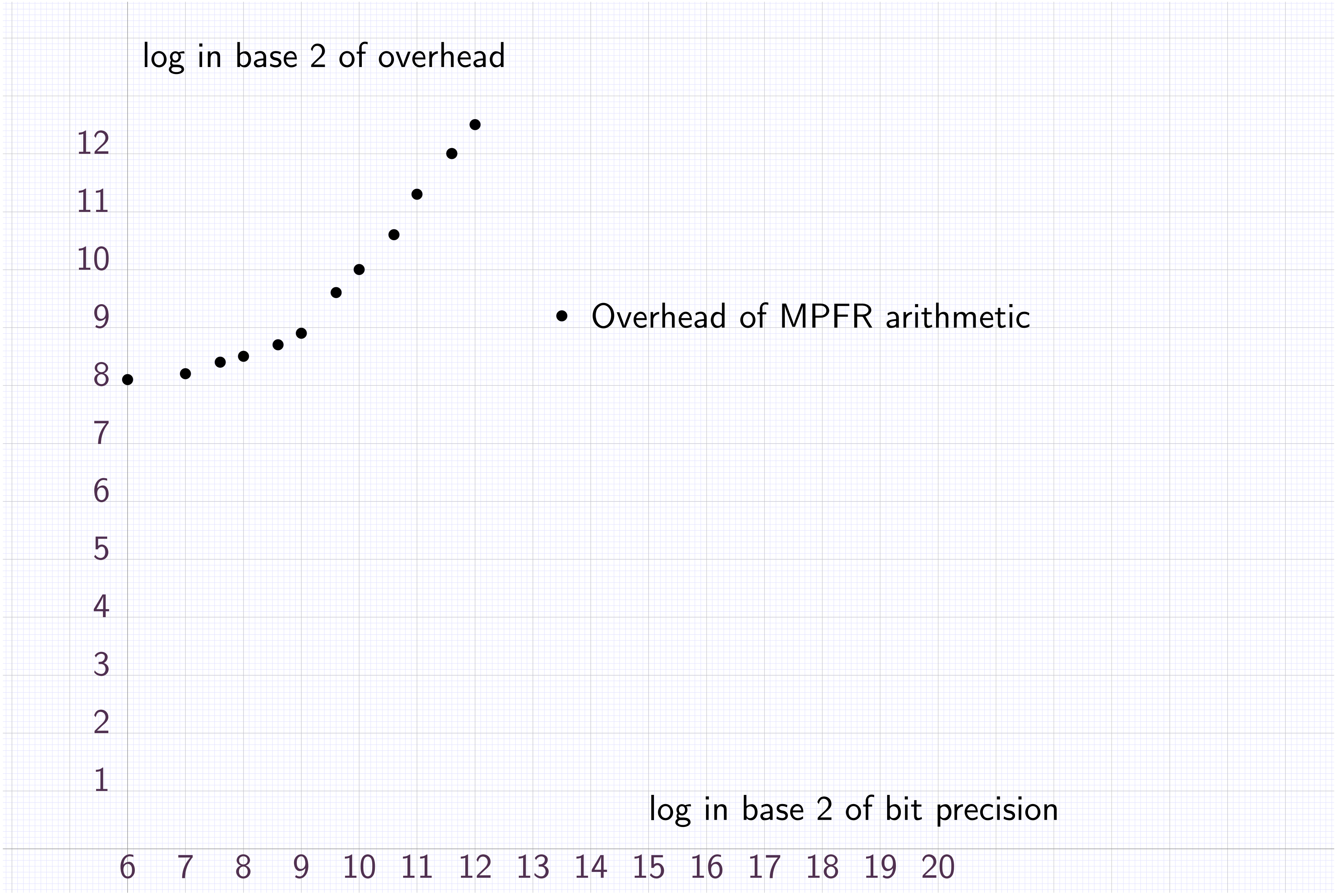
Joris van der Hoeven, Grégoire Lecerf

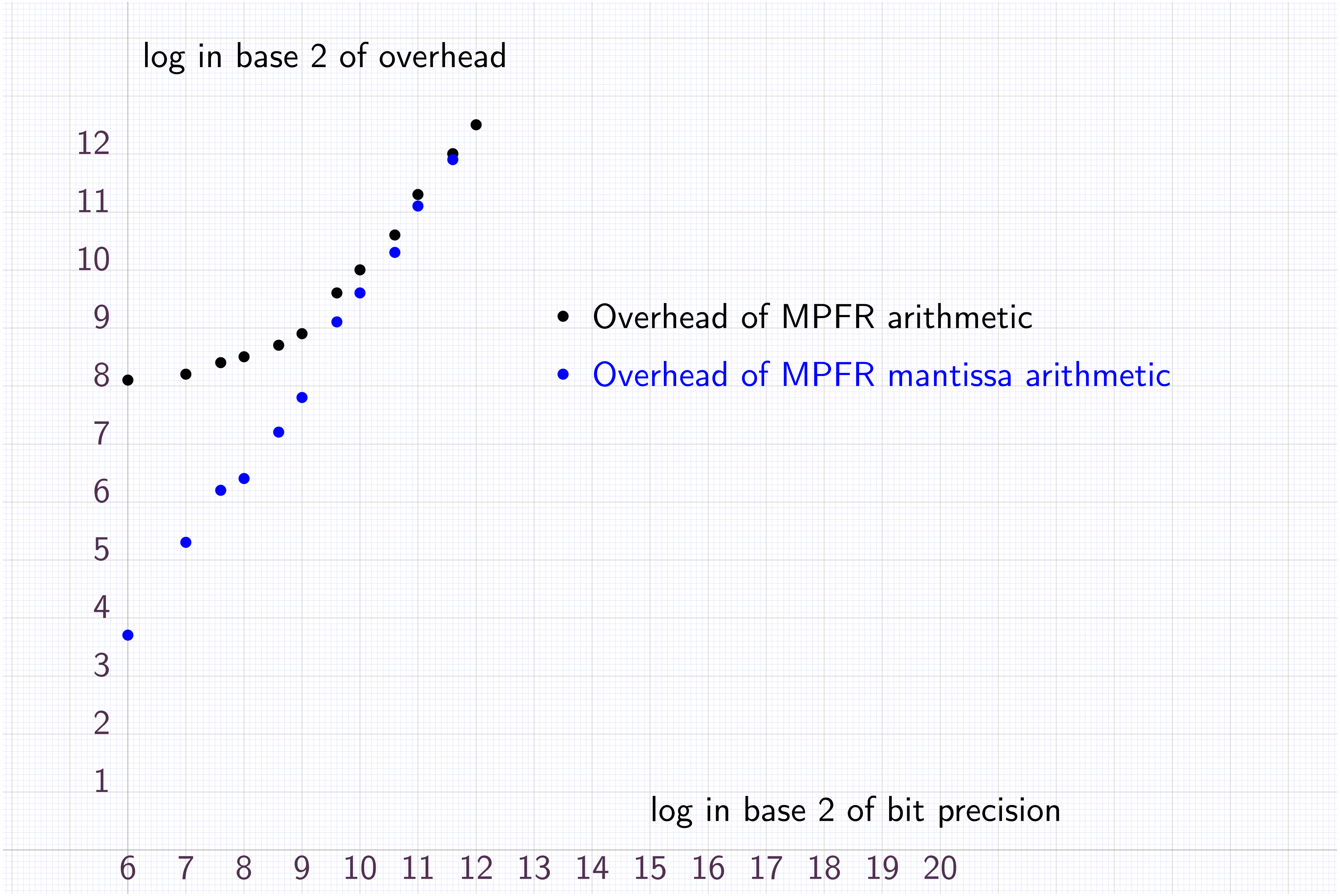
CNRS, École polytechnique

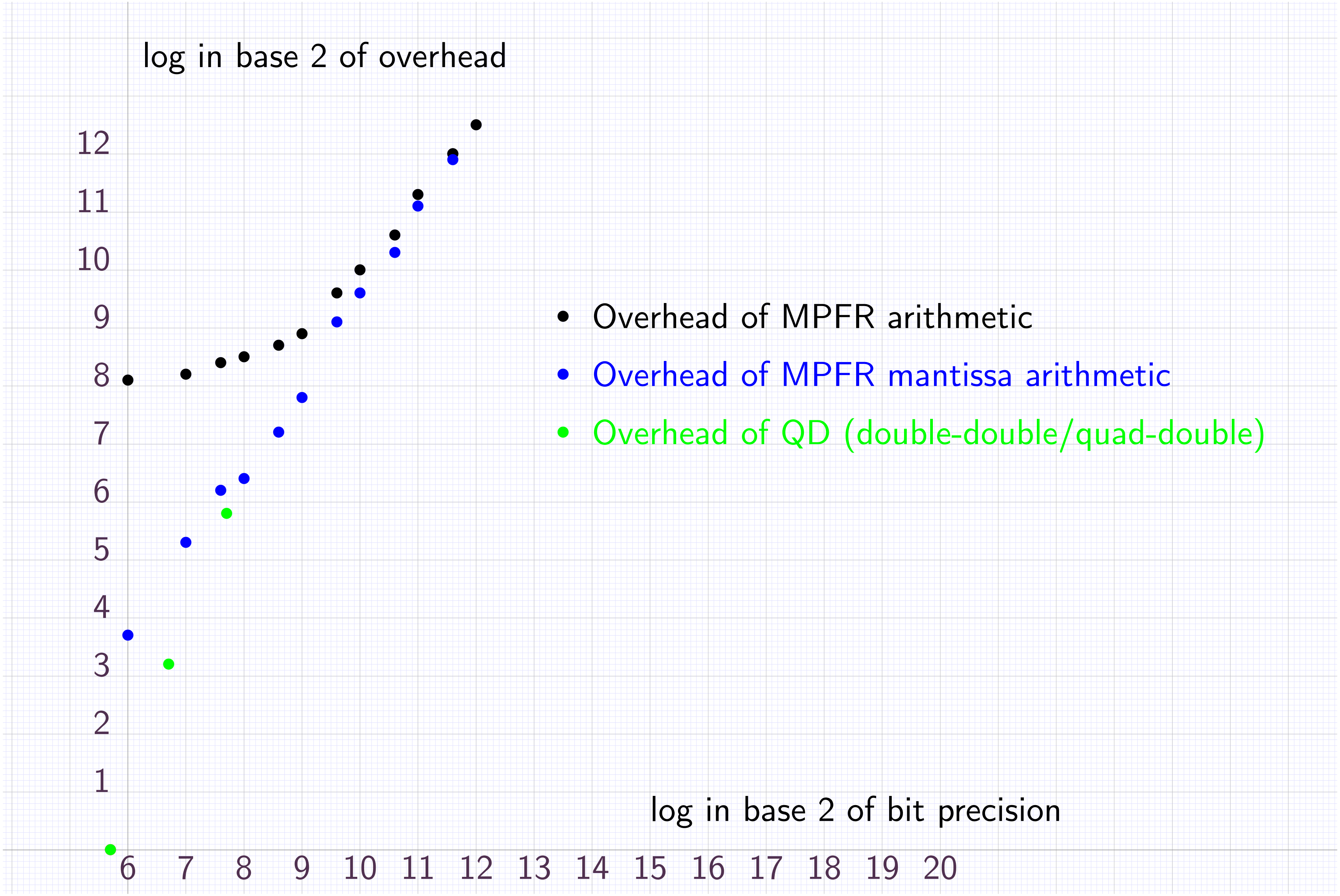


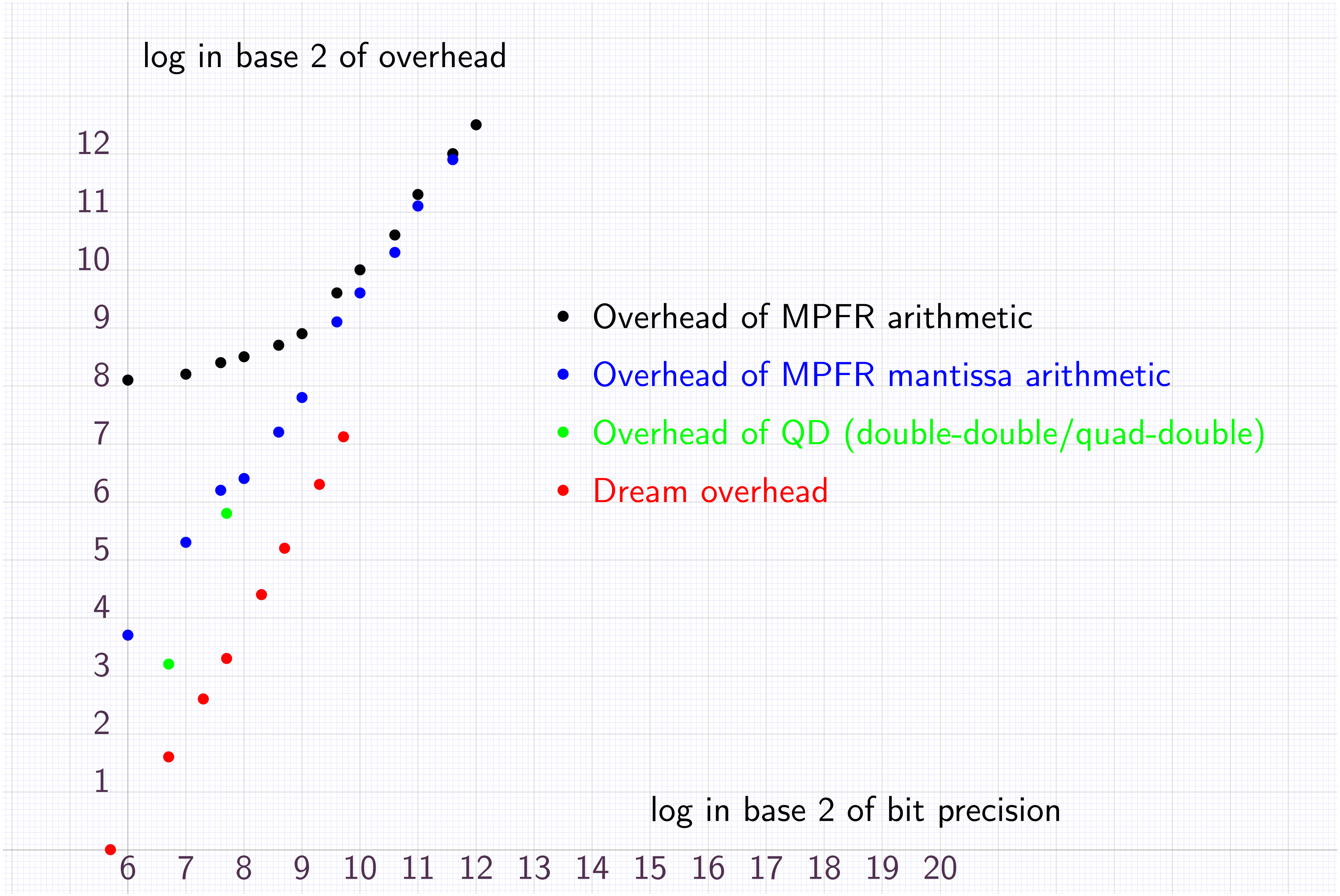
Lyon, June 23, 2015

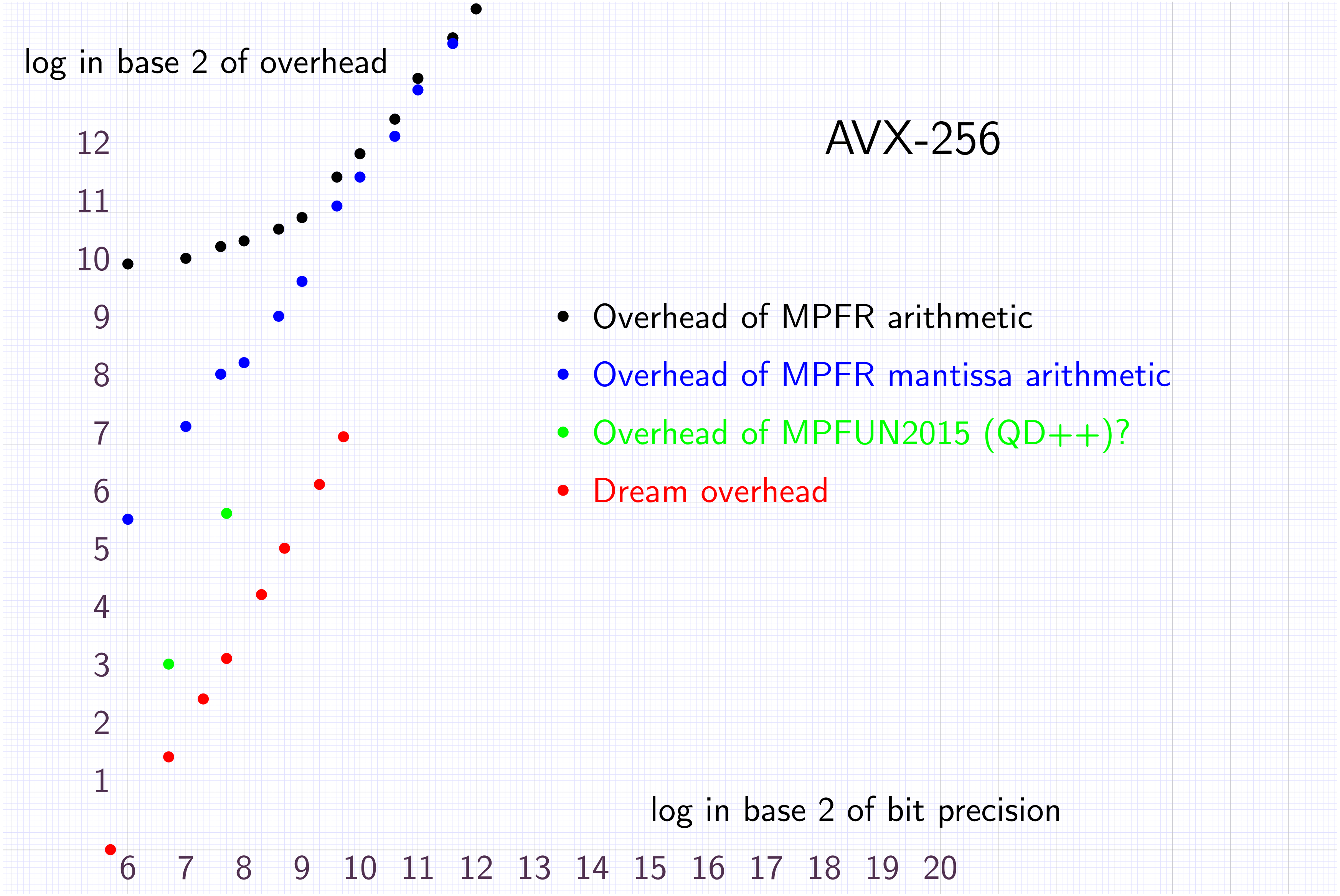
http://www.TeXmacs.org


log in base 2 of overhead
12
11
10
9
8
7
6
5
4
3
2
1
Overhead of MPFR arithmetic
log in base 2 of bit precision
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20


log in base 2 of overhead
12
11
10
9
8
7
6
5
4
3
2
1
Overhead of MPFR arithmetic
Overhead of MPFR mantissa arithmetic
log in base 2 of bit precision
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


log in base 2 of overhead
12
11
10
9
8
7
6
5
4
3
2
1
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
log in base 2 of bit precision
Overhead of MPFR arithmetic
Overhead of MPFR mantissa arithmetic
Overhead of QD (double-double/quad-double)


log in base 2 of overhead
12
11
10
9
8
7
6
5
4
3
2
1
Overhead of MPFR arithmetic
Overhead of MPFR mantissa arithmetic
Overhead of QD (double-double/quad-double)
Dream overhead
log in base 2 of bit precision
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20


log in base 2 of overhead
AVX-256
12
11
10
9
8
7
6
5
4
3
2
1
Overhead of MPFR arithmetic
Overhead of MPFR mantissa arithmetic
Overhead of MPFUN2015 (QD++)?
Dream overhead
log in base 2 of bit precision
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

- Long term: faster general purpose medium precision floating point arithmetic.

  (difficulty: high overhead for software implementation of floating point)

- Long term: faster general purpose medium precision floating point arithmetic.

  (difficulty: high overhead for software implementation of floating point)

- This talk: faster special purpose medium precision fixed point arithmetic.

  (difficulty: higher implementation cost)

- Long term: faster general purpose medium precision floating point arithmetic.

  (difficulty: high overhead for software implementation of floating point)

- This talk: faster special purpose medium precision fixed point arithmetic.

  (difficulty: higher implementation cost)

- Easiest test case: FFT.

  (sample application: integration of p.d.e.s from hydrodynamics)

- Long term: faster general purpose medium precision floating point arithmetic.

  (difficulty: high overhead for software implementation of floating point)

- This talk: faster special purpose medium precision fixed point arithmetic.

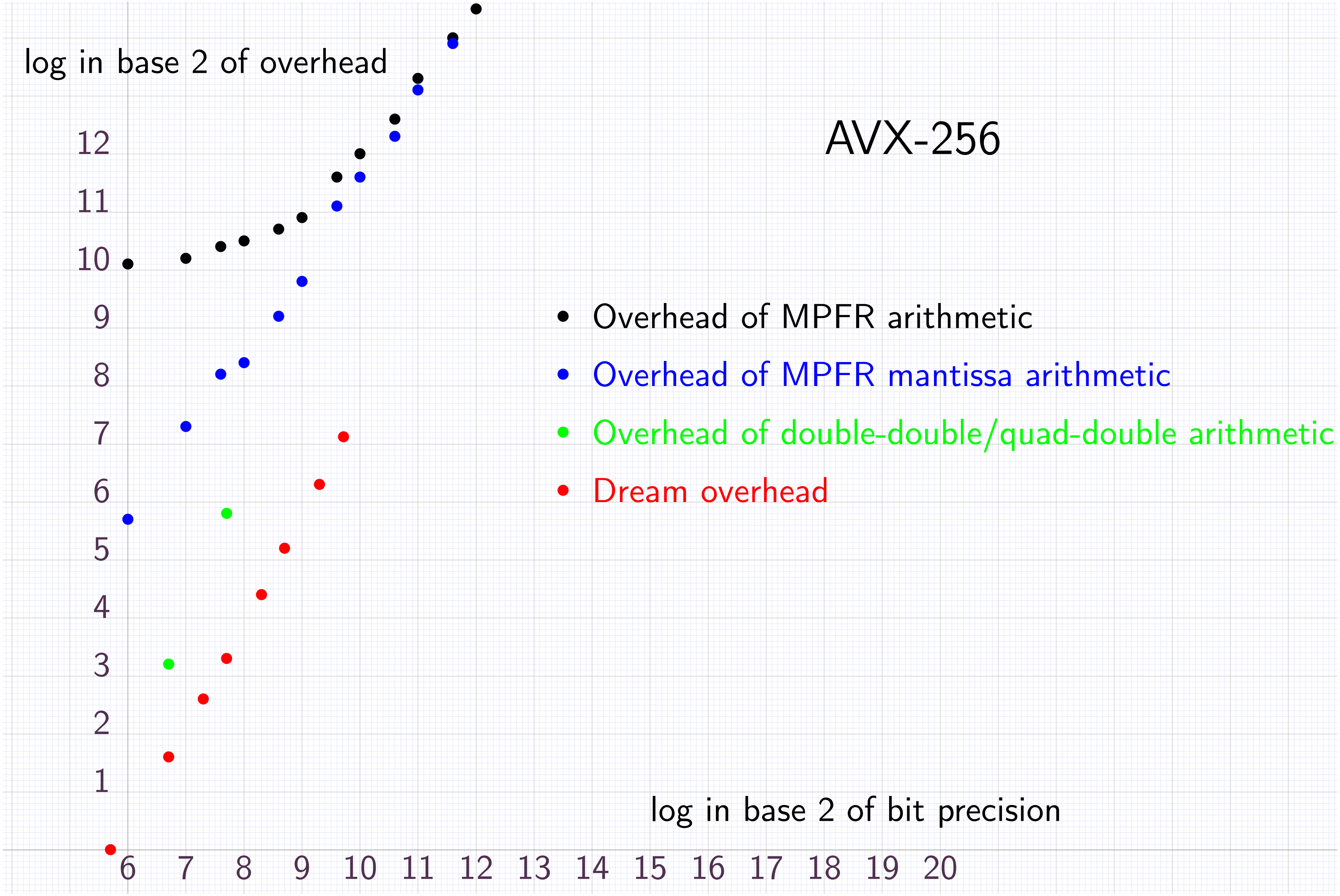
  (difficulty: higher implementation cost)

- Easiest test case: FFT.

  (sample application: integration of p.d.e.s from hydrodynamics)

- General philosophy:

  1. Optimized implementations of core routines (FFT, BLAS, ...).
  2. Slower general purpose routines for remaining tasks.


log in base 2 of overhead
AVX-256
Overhead of MPFR arithmetic
Overhead of MPFR mantissa arithmetic
Overhead of double-double/quad-double arithmetic
Dream overhead
log in base 2 of bit precision
1 2 3 4 5 6 7 8 9 10 11 12
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

- Redundant “carry-save representation” using “nail bits”.
  - Classical in hardware.
  - Experimental support in GMP library, but disabled by default.

- Redundant “carry-save representation” using “nail bits”.
  - Classical in hardware.
  - Experimental support in GMP library, but disabled by default.
- Use extra nail bits to avoid some normalizations during butterflies in the FFT.

- Redundant "carry-save representation" using "nail bits".
  - Classical in hardware.
  - Experimental support in GMP library, but disabled by default.
- Use extra nail bits to avoid some normalizations during butterflies in the FFT.
- Suitable replacements for TwoSum and TwoProduct algorithms.

Strategy

- Redundant “carry-save representation” using “nail bits”.
  - Classical in hardware.
  - Experimental support in GMP library, but disabled by default.
- Use extra nail bits to avoid some normalizations during butterflies in the FFT.
- Suitable replacements for TwoSum and TwoProduct algorithms.
- Careful implementation compatible with SIMD technology,

  with special data type for SIMD multiple precision numbers.

- $\mu = 52$ (machine precision), $E_{\min} = -1022$, $E_{\max} = 1023$ (minimal/maximal exponents).

- $\mu = 52$ (machine precision), $E_{\min} = -1022$, $E_{\max} = 1023$ (minimal/maximal exponents).
- Redundant fixed point representation $x = [\![x_0, \ldots, x_{k-1}]\!]$ at precision $k\,p$

$$x = x_0 + x_1 + \cdots + x_{k-1}$$

$$x = x_0 + x_1\, 2^{-p} + \cdots + x_{k-1}\, 2^{-(k-1)p}$$

where $x_0, \ldots, x_{k-1}$ are integer multiples of $2^{-p}, \ldots, 2^{-kp}$ (resp. $2^{-p}, \ldots, 2^{-p}$).

$x_0, \ldots, x_{k-1}$ are machine IEEE double precision numbers (so that $p \leqslant \mu$).

May release integer multiple condition for least significant number x_{k-1}.

- $\mu = 52$ (machine precision), $E_{\min} = -1022$, $E_{\max} = 1023$ (minimal/maximal exponents).
- Redundant fixed point representation $x = [\![x_0, \ldots, x_{k-1}]\!]$ at precision $k\,p$

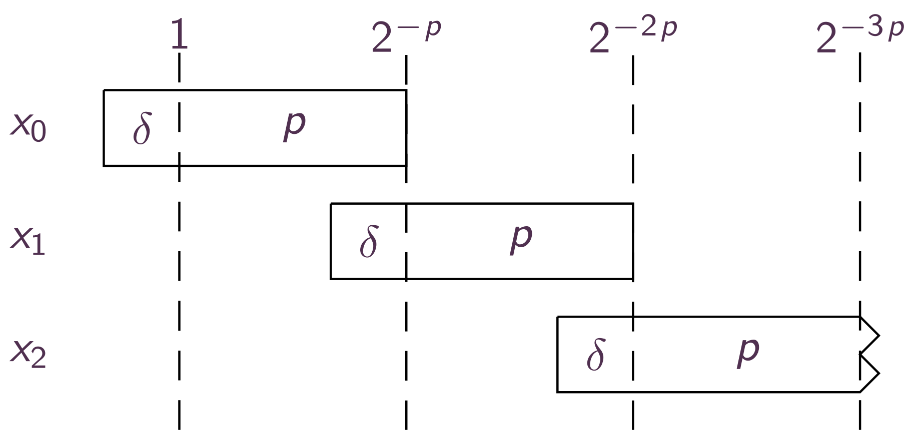
$$x = x_0 + x_1 + \cdots + x_{k-1}$$
$$x = x_0 + x_1\,2^{-p} + \cdots + x_{k-1}\,2^{-(k-1)p}$$

where $x_0, \ldots, x_{k-1}$ are integer multiples of $2^{-p}, \ldots, 2^{-kp}$ (resp. $2^{-p}, \ldots, 2^{-p}$).

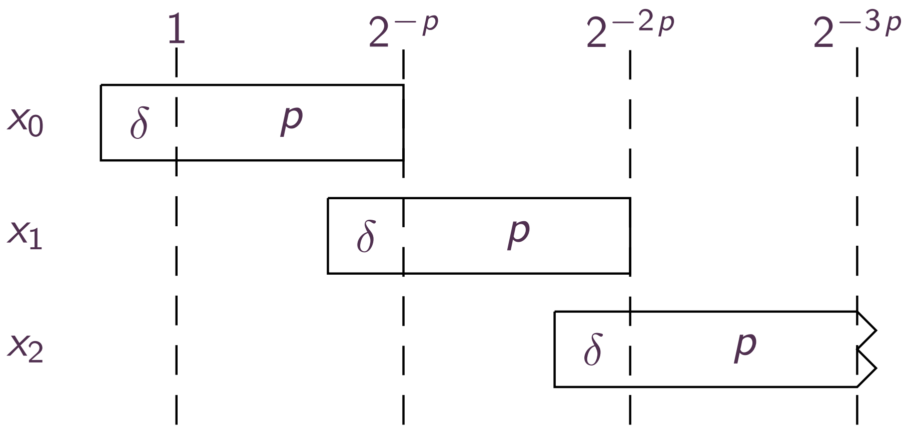
$x_0, \ldots, x_{k-1}$ are machine IEEE double precision numbers (so that $p \leqslant \mu$).

May release integer multiple condition for least significant number x_{k-1}.

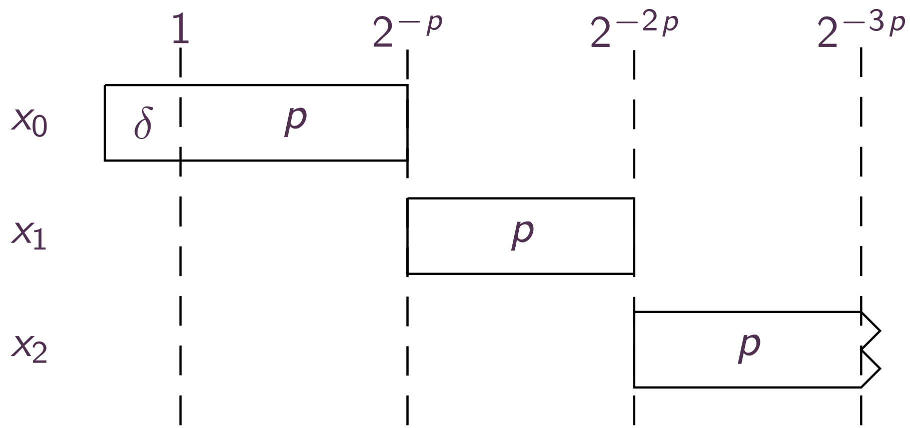
- We take $p = \mu - \delta$, where δ is a suitable number of nail bits, e.g. $\delta = 4$



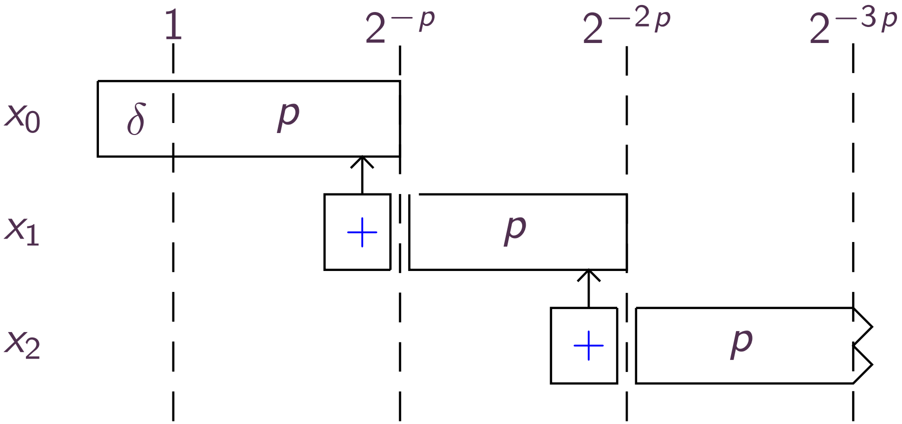
$[\![x_0, \ldots, x_{k-1}]\!]$ normal if $|x_1| < 2^{-p}, \ldots, |x_{k-1}| < 2^{-(k-1)p}$



$[\![x_0, \ldots, x_{k-1}]\!]$ normal if $|x_1| < 2^{-p}, \ldots, |x_{k-1}| < 2^{-(k-1)p}$



$[\![x_0, \ldots, x_{k-1}]\!]$ normal if $|x_1| < 2^{-p}, \ldots, |x_{k-1}| < 2^{-(k-1)p}$



Input: IEEE number $x \in \mathbb{F}$ and exponent $e \in \{E_{\min}, \dots, E_{\max} - \mu\}$ with $|x| < 2^{e+\mu-2}$.

Output: IEEE number $\tilde{x} \in \mathbb{F}$ with $\tilde{x} \in \mathbb{Z}\, 2^e$ and $|\tilde{x} - x| < 2^e$.

Algorithm Split$_e$(x)

$a := \circ(x + {}^3/_2 \cdot 2^{e+\mu})$
return $\circ(a - {}^3/_2 \cdot 2^{e+\mu})$

Input: IEEE number $x \in \mathbb{F}$ and exponent $e \in \{E_{\min}, \dots, E_{\max} - \mu\}$ with $|x| < 2^{e+\mu-2}$.

Output: IEEE number $\tilde{x} \in \mathbb{F}$ with $\tilde{x} \in \mathbb{Z}\, 2^e$ and $|\tilde{x} - x| < 2^e$.

Algorithm Split$_e$(x)

$a := \circ(x + 3/2 \cdot 2^{e+\mu})$
return $\circ(a - 3/2 \cdot 2^{e+\mu})$

Normalization for $k = 2$:

Algorithm Normalize(x)

$c := \text{Split}_{-p}(x_1)$
return $[\![\circ(x_0 + c), \circ(x_1 - c)]\!]$

Input: IEEE number $x \in \mathbb{F}$ and exponent $e \in \{E_{\min}, \ldots, E_{\max} - \mu\}$ with $|x| < 2^{e+\mu-2}$.

Output: IEEE number $\tilde{x} \in \mathbb{F}$ with $\tilde{x} \in \mathbb{Z}\, 2^e$ and $|\tilde{x} - x| < 2^e$.

Algorithm Split$_e$(x)

$a := \circ(x + 3/2 \cdot 2^{e+\mu})$
return $\circ(a - 3/2 \cdot 2^{e+\mu})$

Normalization for general k:

Algorithm Normalize(x)

$r_{k-1} := x_{k-1}$
for i **from** $k-1$ **down to** 1 **do**
    $c_i := \mathrm{Split}_{-ip}(r_i)$
    $\tilde{x}_i := \circ(r_i - c_i)$
    $r_{i-1} := \circ(x_{i-1} + c_i)$
$\tilde{x}_0 := r_0$
return $[\![\tilde{x}_0, \ldots, \tilde{x}_{k-1}]\!]$

Algorithm Add(x, y)

return $[\![\circ(x_0 + y_0), \circ(x_1 + y_1)]\!]$

Algorithm Add(x, y)

return $[\![\circ(x_0+y_0), \circ(x_1+y_1)]\!]$

Algorithm LongMul$_e(x, y)$

$a := \circ(xy + {}^3\!/_2 \cdot 2^{e+\mu})$
$h := \circ(a - {}^3\!/_2 \cdot 2^{e+\mu})$
$l := \circ(xy - h)$
return (h, l)

Algorithm Add(x, y)

return $[\![\circ(x_0 + y_0), \circ(x_1 + y_1)]\!]$

Algorithm LongMul$_e$(x, y)

$a := \circ(x\,y + 3/2 \cdot 2^{e+\mu})$
$h := \circ(a - 3/2 \cdot 2^{e+\mu})$
$l := \circ(x\,y - h)$
return (h, l)

Algorithm Multiply(x, y)

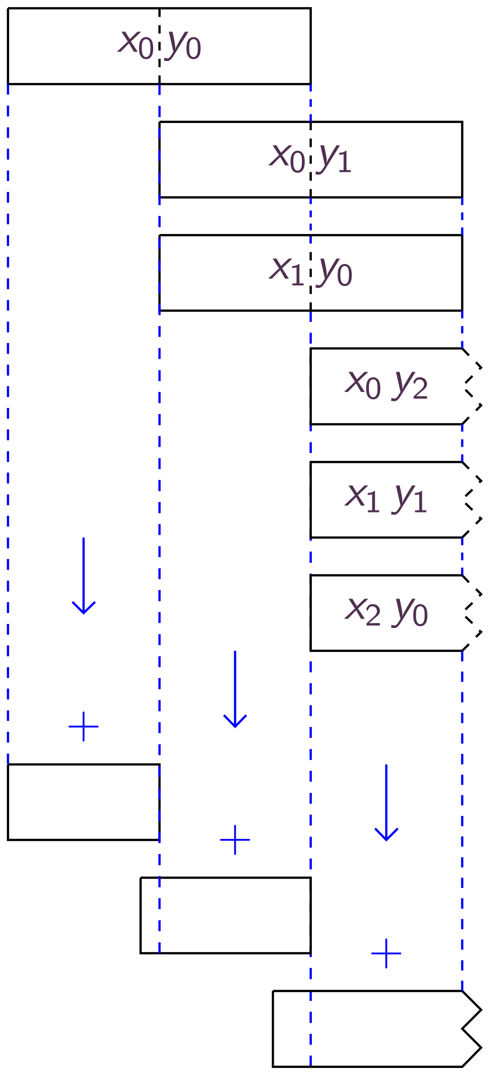
$(h, l) := \mathrm{LongMul}_{-p}(x_0, y_0)$
$l := \circ(x_0\, y_1 + l)$
$l := \circ(x_1\, y_0 + l)$
return $[\![h, l]\!]$

Algorithm Add(x, y)

return $[\![\circ(x_0 + y_0), \ldots, \circ(x_{k-1} + y_{k-1})]\!]$

Algorithm Multiply(x, y)

$(r_0, r_1) := \mathrm{LongMul}_{-p}(x_0, y_0)$
for i **from** 1 **to** $k-2$ **do**
    $(h, r_{i+1}) := \mathrm{LongMul}_{-(i+1)p}(x_0, y_i)$
    $r_i := \circ(r_i + h)$
    for j **from** 1 **to** i **do**
        $(h, l) := \mathrm{LongMul}_{-(i+1)p}(x_j, y_{i-j})$
        $r_i := \circ(r_i + h)$
        $r_{i+1} := \circ(r_{i+1} + l)$
for i **from** 0 **to** $k-1$ **do**
    $r_{k-1} := \circ(x_i y_{k-1-i} + r_{k-1})$
return $[\![r_0, \ldots, r_{k-1}]\!]$


$x_0\, y_0$
$x_0\, y_1$
$x_1\, y_0$
$x_0\, y_2$
$x_1\, y_1$
$x_2\, y_0$
+
+
+

Algorithm ComplexMultiply(u, v)

$a :=$ Multiply($\Re u, \Re v$)
$b :=$ Multiply($\Im u, \Im v$)
$c :=$ Multiply($\Re u, \Im v$)
$d :=$ Multiply($\Im u, \Re v$)
return Subtract(a, b) + Add(c, d) i

$$\tilde{u} = u + \omega v$$
$$\tilde{v} = u - \omega v$$

Algorithm DirectButterfly(u, v, ω)

$z :=$ ComplexMultiply(ω, v)
$u' :=$ ComplexAdd(u, z)
$v' :=$ ComplexSubtract(u, z)
$\tilde{u} :=$ ComplexNormalize(u')
$\tilde{v} :=$ ComplexNormalize(v')
return ($\tilde{u}, \tilde{v}$)

ν	8	9	10	11	12	13	14	15	16
double	0.54	1.1	2.5	6.8	16	37	85	220	450
long double	9.5	21	48	110	230	500	1100	2300	5000
__float128	94	220	490	1100	2400	5300	11000	25000	53000
quadruple	5.0	11	25	55	120	260	570	1200	2600
fixed_quadruple	2.8	6.4	15	33	71	160	380	820	1700
FFTW3 double	0.43	0.94	2.3	5.4	15	34	85	190	400
FFTW3 long double	6.1	14	31	70	153	332	720	1600	3400
FFTW3 __float128	89	205	463	1000	2300	5100	11000	24000	51000
MPFR (113 bits)	270	610	1400	3100	6800	15000	33000	81000	230000
double	0.54	1.1	2.5	6.8	16	37	85	220	450
fixed_quadruple	2.8	6.4	15	33	71	160	380	820	1700
fixed_hexuple	7.6	17	38	84	180	400	870	1800	3900
fixed_octuple	18	42	93	200	450	980	2100	4500	9600

Table 1. FFT timings for size $n = 2^{\nu}$, in micro-seconds.

- Classical double-double arithmetic: hardware FusedAddAdd instruction.

- Classical double-double arithmetic: hardware FusedAddAdd instruction.
- For our approach, faithful rounding more important than correct rounding.

- Classical double-double arithmetic: hardware FusedAddAdd instruction.
- For our approach, faithful rounding more important than correct rounding.
- Instructions for faster splitting-normalization would be nice. E.g.:
  - Built-in Split instruction (RoundScale?).
  - Instruction for $(a, b) \mapsto (a + b, a - b)$.

- Classical double-double arithmetic: hardware FusedAddAdd instruction.
- For our approach, faithful rounding more important than correct rounding.
- Instructions for faster splitting-normalization would be nice. E.g.:
  - Built-in Split instruction (RoundScale?).
  - Instruction for $(a, b) \mapsto (a + b, a - b)$.
- Better support for long multiplication.

- Classical double-double arithmetic: hardware FusedAddAdd instruction.
- For our approach, faithful rounding more important than correct rounding.
- Instructions for faster splitting-normalization would be nice. E.g.:
  - Built-in Split instruction (RoundScale?).
  - Instruction for $(a, b) \mapsto (a + b, a - b)$.
- Better support for long multiplication.
- What about integer instructions for SIMD large integer multiplication?