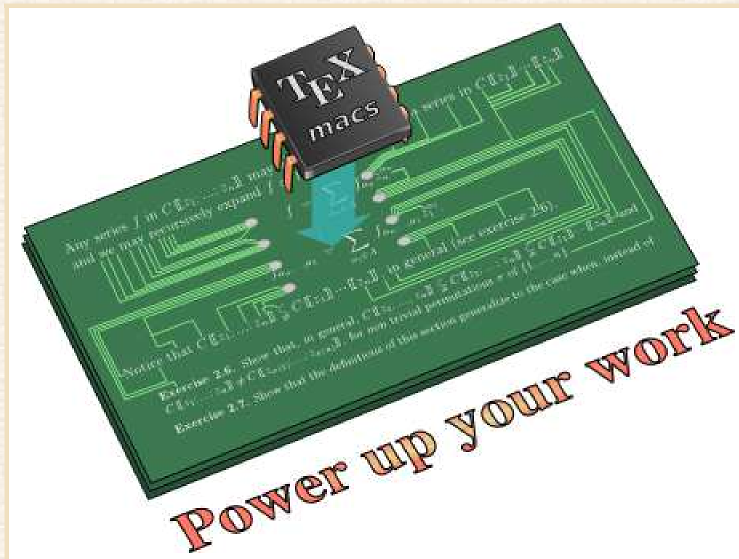
Multi-precision computations & high performance

A delicate marriage

Joris van der Hoeven
CNRS, École polytechnique



Bangalore, 2011

http://www.TEXMACS.org

The silly remark not to make

"We already lack precision in our input data. Why use multiple precision?"

The silly remark not to make

"We already lack precision in our input data. Why use multiple precision?"

"Does the Navier-Stokes equation change when you modify your input data?"

The silly remark not to make

"We already lack precision in our input data. Why use multiple precision?"

"Does the Navier-Stokes equation change when you modify your input data?"

"Does the nature of a solution to NS change if you (slightly) modify your input data?"

The silly remark not to make

"We already lack precision in our input data. Why use multiple precision?"

"Does the Navier-Stokes equation change when you modify your input data?"

"Does the nature of a solution to NS change if you (slightly) modify your input data?"

"Why not perform all computations using 8 bits of precision?"

When do we need multiple precision arithmetic?

Large condition numbers

Condition number $\kappa \geqslant 2^{52}$ implies double precision arithmetic makes no sense

When do we need multiple precision arithmetic?

Large condition numbers

Condition number $\kappa \geqslant 2^{52}$ implies double precision arithmetic makes no sense

Example: Inversion of a matrix M with

$$\kappa(M) = \|M\| \, \|M^{-1}\| \geqslant 2^{52}$$

When do we need multiple precision arithmetic?

Large condition numbers

Condition number $\kappa \geqslant 2^{52}$ implies double precision arithmetic makes no sense

Example: Inversion of a matrix M with

$$\kappa(M) = \|M\| \, \|M^{-1}\| \geqslant 2^{52}$$

Example: integration of a dynamical system $Y' = \Phi(Y), Y(0) = C$ near a singularity σ

$$\kappa\left(\frac{\partial Y(\sigma - \varepsilon)}{\partial C}\right) \geqslant 2^{52}$$

When do we need multiple precision arithmetic?

Beyond condition numbers

Problem: compute some real number with a relative error $\varepsilon > 0$

When do we need multiple precision arithmetic?

Beyond condition numbers

Problem: compute some real number with a relative error $\varepsilon > 0$

Classical: pick precision p with $\kappa\, 2^{-p} \leqslant \varepsilon$, that is $p \geqslant \phi(p_\varepsilon) = p_\varepsilon + \log_2 \kappa$ with $p_\varepsilon = \log_2 \frac{1}{\varepsilon}$

When do we need multiple precision arithmetic?

Beyond condition numbers

Problem: compute some real number with a relative error $\varepsilon > 0$

Classical: pick precision p with $\kappa\, 2^{-p} \leqslant \varepsilon$, that is $p \geqslant \phi(p_\varepsilon) = p_\varepsilon + \log_2 \kappa$ with $p_\varepsilon = \log_2 \frac{1}{\varepsilon}$

Non-classical: we need $p \geqslant c\, p_\varepsilon$ with $c > 1$ or even $p \geqslant p_\varepsilon^c$ with $c > 1$

When do we need multiple precision arithmetic?

Beyond condition numbers

Problem: compute some real number with a relative error $\varepsilon > 0$

Classical: pick precision p with $\kappa\, 2^{-p} \leqslant \varepsilon$, that is $p \geqslant \phi(p_\varepsilon) = p_\varepsilon + \log_2 \kappa$ with $p_\varepsilon = \log_2 \frac{1}{\varepsilon}$

Non-classical: we need $p \geqslant c\, p_\varepsilon$ with $c > 1$ or even $p \geqslant p_\varepsilon^c$ with $c > 1$

Asymptotic extrapolation for a favourable sequence f_n with

$$f_n \approx \alpha^n \left(\frac{a_0 \log n + b_0}{n^0} + \frac{a_1 \log n + b_1}{n^1} + \frac{a_2 \log n + b_2}{n^2} + \cdots \right)$$

Problem: cost to determine α with relative error $\varepsilon > 0$?

Compute $f_0, \ldots, f_N$

When do we need multiple precision arithmetic?

Beyond condition numbers

Problem: compute some real number with a relative error $\varepsilon > 0$

Classical: pick precision p with $\kappa\, 2^{-p} \leqslant \varepsilon$, that is $p \geqslant \phi(p_\varepsilon) = p_\varepsilon + \log_2 \kappa$ with $p_\varepsilon = \log_2 \frac{1}{\varepsilon}$

Non-classical: we need $p \geqslant c\, p_\varepsilon$ with $c > 1$ or even $p \geqslant p_\varepsilon^c$ with $c > 1$

Asymptotic extrapolation for a favourable sequence f_n with

$$f_n \approx \alpha^n \left(\frac{a_0 \log n + b_0}{n^0} + \frac{a_1 \log n + b_1}{n^1} + \frac{a_2 \log n + b_2}{n^2} + \cdots \right)$$

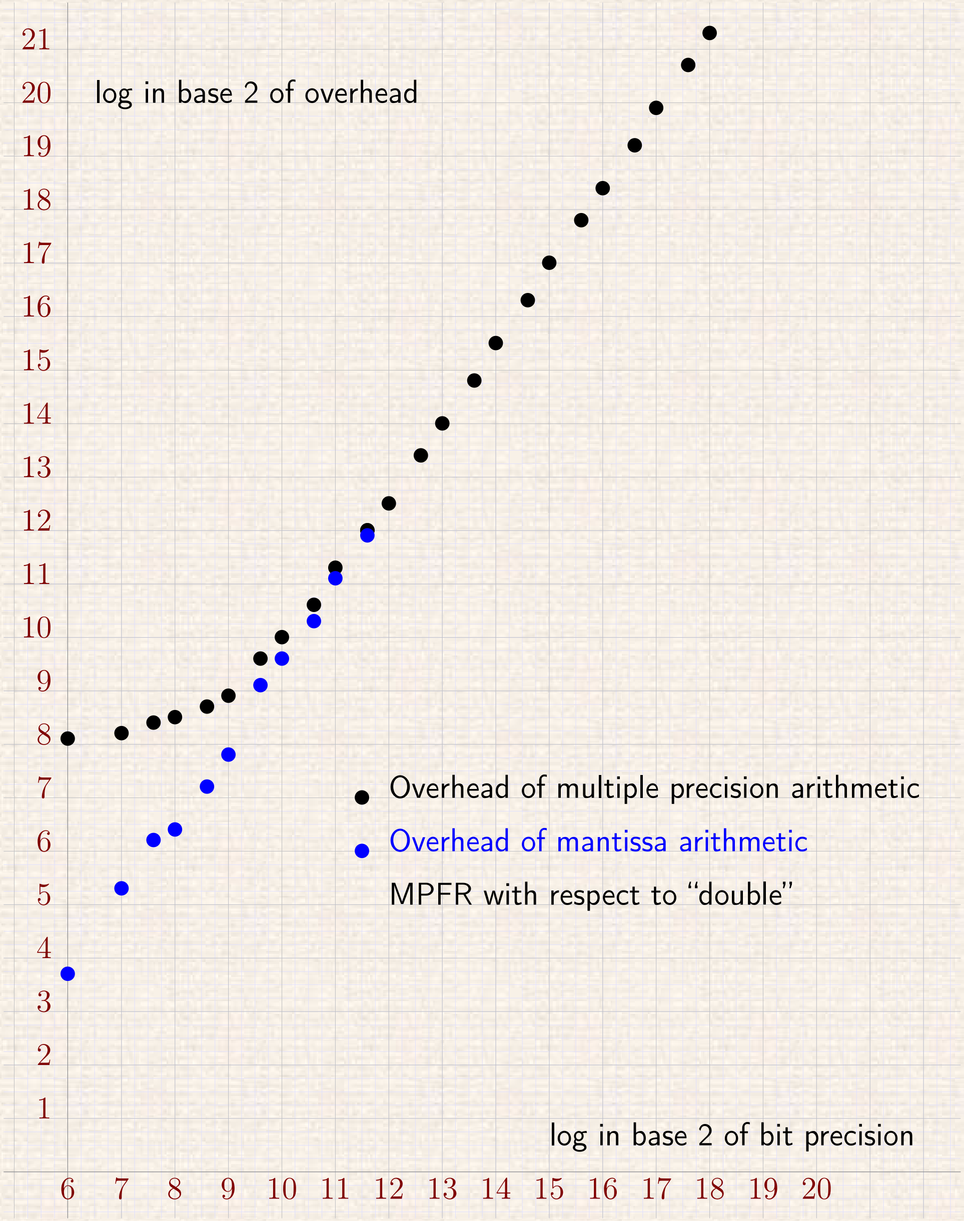
Problem: cost to determine α with relative error $\varepsilon > 0$?

Compute $f_0, \ldots, f_N$

Analysis: computation of $\alpha, a_0, b_0, \ldots, a_{k-1}, b_{k-1}$ yields α with relative error $\approx N^{-k}$
However: we need a precision p with $2^{-p} \lessapprox N^{-2k}$, i.e. $p \geqslant 2\, p_\varepsilon + o(p_\varepsilon)$
Choice of N: depends and to be analyzed in detail

Mathematical simulations

Remark. Multiple precision computations can be particularly useful in order to “simulate” an equation with simple exact mathematical boundary conditions.


log in base 2 of overhead
Overhead of multiple precision arithmetic
Overhead of mantissa arithmetic
MPFR with respect to "double"
log in base 2 of bit precision
21
20
19
18
17
16
15
14
13
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

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

Software implementation, strategy I

Use built-in floating point arithmetic

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

Software implementation, strategy I

Use built-in floating point arithmetic
Problem: hardware implementation of three sum

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

Software implementation, strategy I

Use built-in floating point arithmetic
Question: will CPU manufacturers help us?

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

Software implementation, strategy I

Use built-in floating point arithmetic
Question: will CPU manufacturers help us?
Answer: no, unless turbulence will be interesting for computer games

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

Software implementation, strategy I

Use built-in floating point arithmetic
Question: will CPU manufacturers help us?
Answer: no, unless turbulence will be interesting for computer games

Software implementation, strategy II

Use built-in integer arithmetic as in MPFR library

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

Software implementation, strategy I

Use built-in floating point arithmetic
Question: will CPU manufacturers help us?
Answer: no, unless turbulence will be interesting for computer games

Software implementation, strategy II

Use built-in integer arithmetic as in MPFR library
Problem 1: overhead for emulation of exponents

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

Software implementation, strategy I

Use built-in floating point arithmetic
Question: will CPU manufacturers help us?
Answer: no, unless turbulence will be interesting for computer games

Software implementation, strategy II

Use built-in integer arithmetic as in MPFR library
Problem 1: overhead for emulation of exponents
Problem 2: overhead for emulation of signs

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

Software implementation, strategy I

Use built-in floating point arithmetic
Question: will CPU manufacturers help us?
Answer: no, unless turbulence will be interesting for computer games

Software implementation, strategy II

Use built-in integer arithmetic as in MPFR library
Problem 1: overhead for emulation of exponents
Problem 2: overhead for emulation of signs
Problem 3: overhead for emulation of correct rounding

How to implement multiple precision arithmetic?

What the hardware provides

195*, 196*, 197*: software implementation of floating point arithmetic
198*, 199*: mathematical co-processors, single precision, later double precision
200*: single and double precision IEEE arithmetic, integer arithmetic, GPUs
201*: single, double, quadruple precision FP and integer arithmetic, wide SIMD, GPUs

Software implementation, strategy I

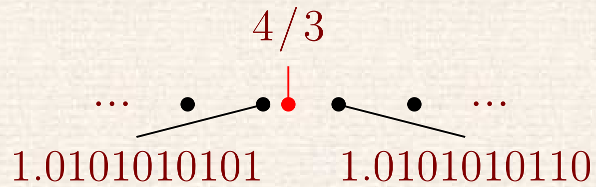
Use built-in floating point arithmetic
Question: will CPU manufacturers help us?
Answer: no, unless turbulence will be interesting for computer games

Software implementation, strategy II

Use built-in integer arithmetic as in MPFR library
Problem 1: overhead for emulation of exponents
Problem 2: overhead for emulation of signs
Problem 3: overhead for emulation of correct rounding
Problem 4: overhead for emulation of exceptions

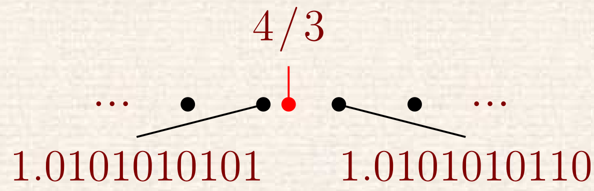
Using built-in floating point arithmetic

Correct rounding



Using built-in floating point arithmetic

Correct rounding

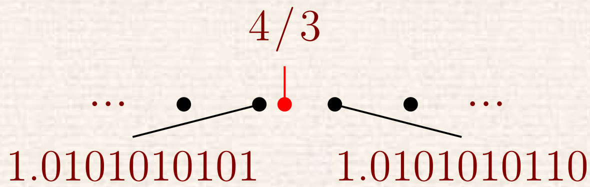


Fused multiply subtract

Problem: exact multiplication of $x, y \in \mathbb{F}_{52}$ as $x\,y = h + l \in \mathbb{F}_{104}$ with $h, l \in \mathbb{F}_{52}$

Using built-in floating point arithmetic

Correct rounding

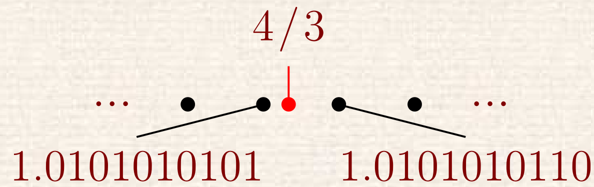


Fused multiply subtract

Problem: exact multiplication of $x, y \in \mathbb{F}_{52}$ as $x\,y = h + l \in \mathbb{F}_{104}$ with $h, l \in \mathbb{F}_{52}$
Solution: $h := (x \times y)_{\mathbb{F}_{52}}$, $l := (x \times y - h)_{\mathbb{F}_{52}}$

Using built-in floating point arithmetic

Correct rounding



Fused multiply subtract

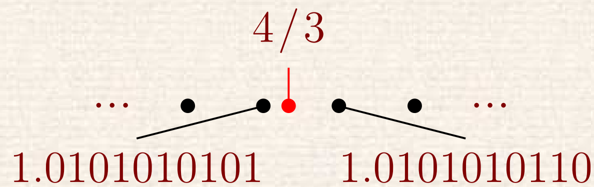
Problem: exact multiplication of $x, y \in \mathbb{F}_{52}$ as $x\,y = h + l \in \mathbb{F}_{104}$ with $h, l \in \mathbb{F}_{52}$
Solution: $h := (x \times y)_{\mathbb{F}_{52}}$, $l := (x \times y - h)_{\mathbb{F}_{52}}$

Three sum (fused add subtract)

Similar operation for addition: $h := (x + y)_{\mathbb{F}_{52}}$, $l := (x + y - h)_{\mathbb{F}_{52}}$

Using built-in floating point arithmetic

Correct rounding



Fused multiply subtract

Problem: exact multiplication of $x, y \in \mathbb{F}_{52}$ as $x\,y = h + l \in \mathbb{F}_{104}$ with $h, l \in \mathbb{F}_{52}$
Solution: $h := (x \times y)_{\mathbb{F}_{52}}$, $l := (x \times y - h)_{\mathbb{F}_{52}}$

Three sum (fused add subtract)

Similar operation for addition: $h := (x + y)_{\mathbb{F}_{52}}$, $l := (x + y - h)_{\mathbb{F}_{52}}$

Exercise

Design multiple precision arithmetic using these operations

Using built-in integer arithmetic

MPFR library

Represent mantissas by GMP integers (with separate field for signs)

Using built-in integer arithmetic

MPFR library

Represent mantissas by GMP integers (with separate field for signs)

Unsigned fixed point arithmetic

$X \in \{0, \ldots, 2^p - 1\}$ represents $x = X\, 2^{-p}$

Multiplication at precision $2\,p$:

$$(X_1\, 2^p + X_0)\,(Y_1\, 2^p + Y_0)\, 2^{-4p} = X_0\, X_1\, 2^{-2p} + (X_0\, Y_1 + X_1\, Y_0)\, 2^{-3p} + \cdots$$

That is: three integer multiplications and four additions

Using built-in integer arithmetic

MPFR library

Represent mantissas by GMP integers (with separate field for signs)

Unsigned fixed point arithmetic

$X \in \{0, \ldots, 2^p - 1\}$ represents $x = X\,2^{-p}$

Multiplication at precision $2\,p$:

$$(X_1\,2^p + X_0)\,(Y_1\,2^p + Y_0)\,2^{-4p} = X_0\,X_1\,2^{-2p} + (X_0\,Y_1 + X_1\,Y_0)\,2^{-3p} + \cdots$$

That is: three integer multiplications and four additions

Signed fixed point arithmetic

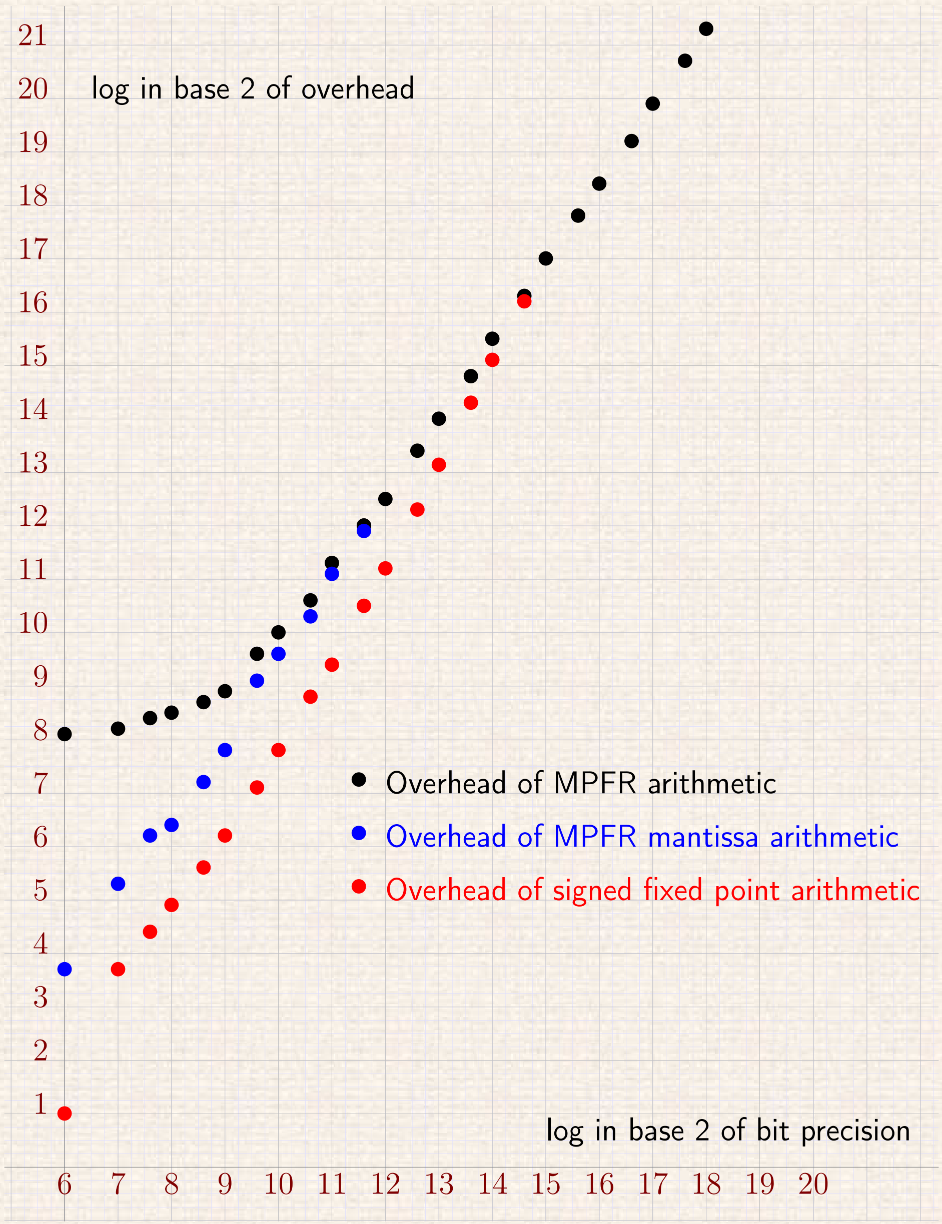
$X \in \{0, \ldots, 2^p - 1\}$ represents $\tilde{x} = X\,2^{-p} - \frac{1}{2}$

$$2\,\tilde{x}\,\tilde{y} + \frac{1}{2} \;=\; 2\,x\,y - (x + y) + 1$$

Multiplication at precision $2\,p$: six extra additions

Moderate precision arithmetic

Precision	Unsigned	Signed	Multipl.
p	1	2	1
$2\,p$	7	13	3
$3\,p$	12	21	6
$4\,p$	18	30	10
$5\,p$	25	40	15
$6\,p$	33	51	21
$7\,p$	42	63	28
$8\,p$	52	76	36



log in base 2 of overhead
log in base 2 of bit precision
Overhead of MPFR arithmetic
Overhead of MPFR mantissa arithmetic
Overhead of signed fixed point arithmetic
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Rethinking scientific computation

Challenge 0: for what kind of problems do we need multiple precision arithmetic?

Rethinking scientific computation

Challenge 0: for what kind of problems do we need multiple precision arithmetic?

Challenge 1: rethink numerical analysis from the multiple precision perspective

Rethinking scientific computation

Challenge 0: for what kind of problems do we need multiple precision arithmetic?

Challenge 1: rethink numerical analysis from the multiple precision perspective

Challenge 2: rethink numerical algorithms together with the underlying arithmetic

Rethinking scientific computation

Challenge 0: for what kind of problems do we need multiple precision arithmetic?

Challenge 1: rethink numerical analysis from the multiple precision perspective

Challenge 2: rethink numerical algorithms together with the underlying arithmetic

Challenge 3: rethink compilers for automatic code generation of non trivial arithmetic

Rethinking scientific computation

Challenge 0: for what kind of problems do we need multiple precision arithmetic?

Challenge 1: rethink numerical analysis from the multiple precision perspective

Challenge 2: rethink numerical algorithms together with the underlying arithmetic

Challenge 3: rethink compilers for automatic code generation of non trivial arithmetic

Challenge 4: how to benefit from massively parallel architectures?

Rethinking scientific computation

Challenge 0: for what kind of problems do we need multiple precision arithmetic?

Challenge 1: rethink numerical analysis from the multiple precision perspective

Challenge 2: rethink numerical algorithms together with the underlying arithmetic

Challenge 3: rethink compilers for automatic code generation of non trivial arithmetic

Challenge 4: how to benefit from massively parallel architectures?

Challenge 5: how to incorporate automatic computation of error bounds?

Rethinking scientific computation

Challenge 0: for what kind of problems do we need multiple precision arithmetic?

Challenge 1: rethink numerical analysis from the multiple precision perspective

Challenge 2: rethink numerical algorithms together with the underlying arithmetic

Challenge 3: rethink compilers for automatic code generation of non trivial arithmetic

Challenge 4: how to benefit from massively parallel architectures?

Challenge 5: how to incorporate automatic computation of error bounds?

Challenge 6: how to interface with symbolic computation?

Claim: we can do all our computations using the signed fixed point representation

Claim: we can do all our computations using the signed fixed point representation

Indeed:

- Assume that we want to transform $a_0, \ldots, a_{n-1}$ with $n = 2^k$
- Write $b_i = \frac{a_i}{2\,n\,\|a\|}$ with $\|a\| = \max_i |a_i|$, so that $\hat{a}_i = 2\,n\,\|a\|\,\hat{b}_i$
- Then all numbers occurring in the FFT are in $\left[-\frac{1}{2}, \frac{1}{2}\right]$

Dynamical systems near singularities

The system to integrate

$$
\begin{aligned}
Y' &= \Phi(Y) \\
Y(0) &= C
\end{aligned}
$$

Dynamical systems near singularities

The system to integrate

$$\begin{aligned} Y' &= \Phi(Y) \\ Y(0) &= C \end{aligned}$$

Power series solution

$$Y(z) = \sum_{k=0}^{\infty} Y_k z^k$$

Dynamical systems near singularities

The system to integrate

$$
\begin{aligned}
Y' &= \Phi(Y) \\
Y(0) &= C
\end{aligned}
$$

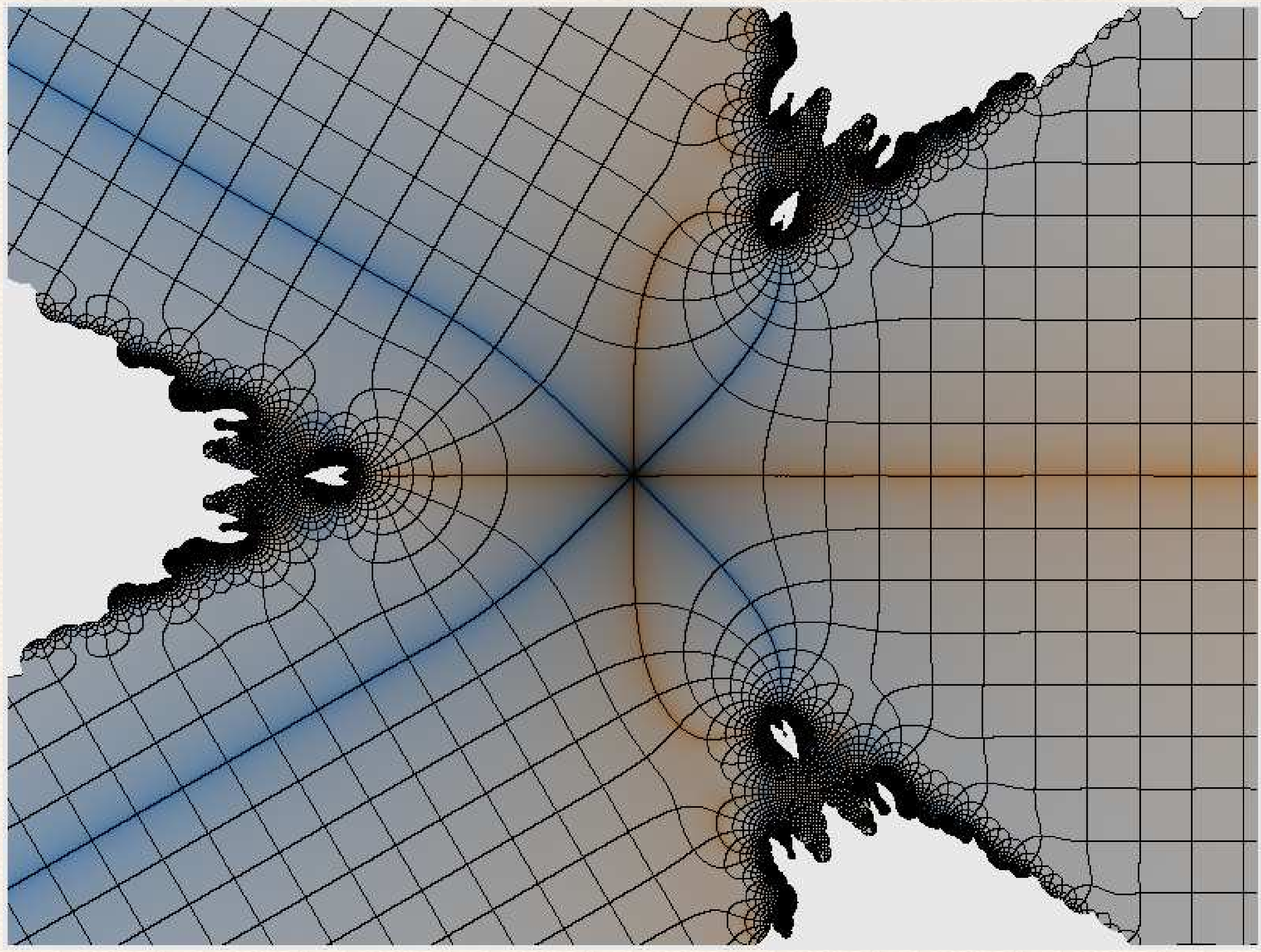
Power series solution

$$
Y(z) = \sum_{k=0}^{\infty} Y_k z^k
$$

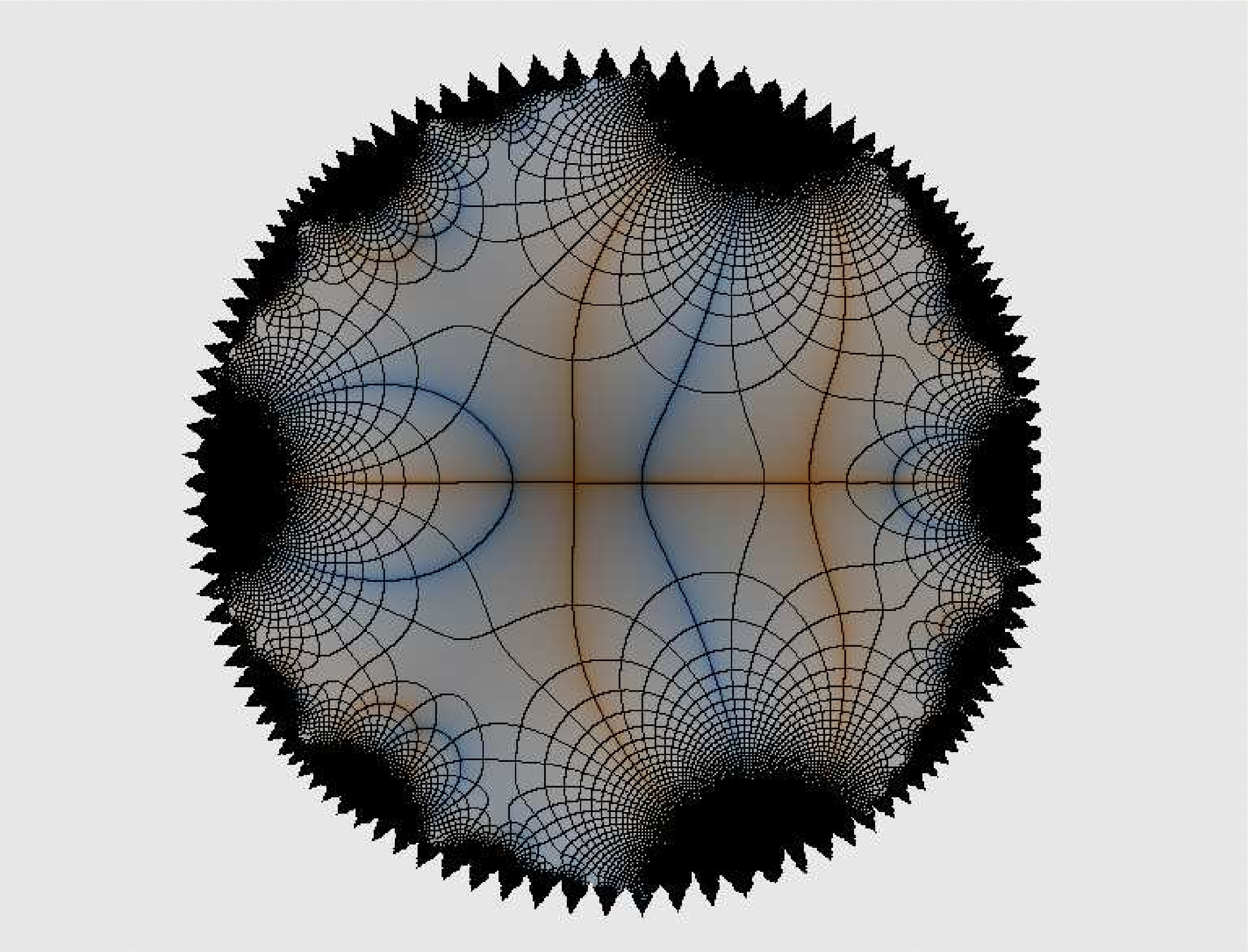
Preconditioning for signed fixed point arithmetic

$$
\begin{aligned}
Y(z) &= \tilde{Y}(\varrho z) \\
\tilde{Y}' &= \varrho \Phi(\tilde{Y})
\end{aligned}
$$

Blasius equation



Chazy equation



Matrix multiplication over $\mathbb{Z}$

Problem

Given $M, N \in \mathbb{Z}_{;p}^{n \times n}$, $\mathbb{Z}_{;p} = \{-2^{p-1}, ..., 0, ..., 2^{p-1} - 1\}$, compute MN

Matrix multiplication over $\mathbb{Z}$

Problem

Given $M, N \in \mathbb{Z}_{;p}^{n \times n}$, $\mathbb{Z}_{;p} = \{-2^{p-1}, \ldots, 0, \ldots, 2^{p-1} - 1\}$, compute $M N$

Chinese remaindering when $p \ll n$

- Pick primes $q_1, \ldots, q_l$ with $q_1 \cdots q_l > n\, 2^p$
- Reduce M and N modulo q_i for each i ($O(n^2\, p \log p \log \log p)$ operations)
- Multiply $(M N \bmod q_i) = (M \bmod q_i)\,(N \bmod q_i)$ for each i ($O(n^3\, p)$ operations)
- Reconstruct $M N$ from the $M N \bmod q_i$ ($O(n^2\, p \log^2 p \log \log p)$ operations)

Matrix multiplication over $\mathbb{Z}$

Problem

Given $M, N \in \mathbb{Z}_{;p}^{n \times n}$, $\mathbb{Z}_{;p} = \{-2^{p-1}, \ldots, 0, \ldots, 2^{p-1} - 1\}$, compute $M N$

Chinese remaindering when $p \ll n$

- Pick primes $q_1, \ldots, q_l$ with $q_1 \cdots q_l > n\, 2^p$
- Reduce M and N modulo q_i for each i ($O(n^2\, p \log p \log \log p)$ operations)
- Multiply $(M N \bmod q_i) = (M \bmod q_i)\,(N \bmod q_i)$ for each i ($O(n^3\, p)$ operations)
- Reconstruct $M N$ from the $M N \bmod q_i$ ($O(n^2\, p \log^2 p \log \log p)$ operations)

FFT over a finite field $\mathbb{F}_q$ when $p \not\ll n$

- Pick $q = 3 \times 2^{30} + 1$ and $\omega = 125$ with $\omega^{2^{29}} = -1$ in $\mathbb{F}_q$
- Write integers in base 2^k with $n\, 2^{2k} < q$, i.e. as evaluations $P(2^k)$, $P \in \mathbb{Z}_{;k}[2^k]$
- Compute products of polynomials $P, Q \in \mathbb{Z}_{;k}[2^k]^{n \times n}$ using FFT w.r.t. ω over $\mathbb{F}_q$
- Cost: $O(n^2\, p \log p \log \log p + n^3\, p)$

Conclusion

Classical double precision methods are and will continue to be a powerful workhorse

Conclusion

Classical double precision methods are and will continue to be a powerful workhorse

But

Conclusion

Classical double precision methods are and will continue to be a powerful workhorse

But

- **If** you are curious

Conclusion

Classical double precision methods are and will continue to be a powerful workhorse

But

- **If** you are curious
- **If** you have time and energy

Conclusion

Classical double precision methods are and will continue to be a powerful workhorse

But

- **If** you are curious
- **If** you have time and energy
- **If** you are good at programming

Conclusion

Classical double precision methods are and will continue to be a powerful workhorse

But

- **If** you are curious
- **If** you have time and energy
- **If** you are good at programming
- **If** you are sufficiently subversive

Conclusion

Classical double precision methods are and will continue to be a powerful workhorse

But

- **If** you are curious
- **If** you have time and energy
- **If** you are good at programming
- **If** you are sufficiently subversive

Then you may try and help developing multiple precision methods

Conclusion

Classical double precision methods are and will continue to be a powerful workhorse

But

- **If** you are curious
- **If** you have time and energy
- **If** you are good at programming
- **If** you are sufficiently subversive

Then you may try and help developing multiple precision methods

Also: no need for a complete theory or big computers
one can start building useful basic libraries for FFT, linear algebra, ...