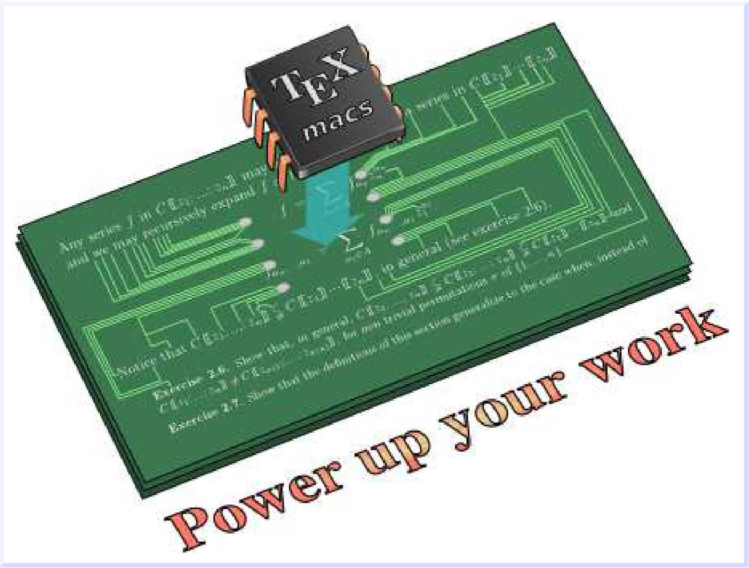
Multiple precision floating point arithmetic on SIMD processors

Joris van der Hoeven

CNRS, École polytechnique



London, July 24, 2017

http://www.TeXmacs.org

Theoretical efficiency of multiple precision floating point arithmetic

- Mainly boils down to the complexity $\mathsf{M}(k)$ of k-"word" multiplication.
- For small k, we have $\mathsf{M}(k) \approx \alpha\, k^2$ for some constant α.
- For large k, we have $\mathsf{M}(k) = O(k \log k\, 8^{\log^* k})$: Harvey–vdH–Lecerf, J. of Complexity 2016.
- For $10 \lessapprox k \lessapprox 1000$, intermediate regimes apply: Karatsuba, Toom–Cook, etc.

Theoretical efficiency of multiple precision floating point arithmetic

- Mainly boils down to the complexity $\mathsf{M}(k)$ of k-“word” multiplication.
- For small k, we have $\mathsf{M}(k) \approx \alpha\, k^2$ for some constant α.
- For large k, we have $\mathsf{M}(k) = O(k \log k\, 8^{\log^* k})$: Harvey–vdH–Lecerf, J. of Complexity 2016.
- For $10 \lessapprox k \lessapprox 1000$, intermediate regimes apply: Karatsuba, Toom–Cook, etc.

How does this translate in practice?

Theoretical efficiency of multiple precision floating point arithmetic

- Mainly boils down to the complexity $\mathsf{M}(k)$ of k-“word” multiplication.
- For small k, we have $\mathsf{M}(k) \approx \alpha\, k^2$ for some constant α.
- For large k, we have $\mathsf{M}(k) = O(k \log k\, 8^{\log^* k})$: Harvey–vdH–Lecerf, J. of Complexity 2016.
- For $10 \lessapprox k \lessapprox 1000$, intermediate regimes apply: Karatsuba, Toom–Cook, etc.

How does this translate in practice?

- How to minimize α for small $k \lessapprox 10$?

Theoretical efficiency of multiple precision floating point arithmetic

- Mainly boils down to the complexity $\mathsf{M}(k)$ of k-"word" multiplication.
- For small k, we have $\mathsf{M}(k) \approx \alpha\, k^2$ for some constant α.
- For large k, we have $\mathsf{M}(k) = O(k \log k\, 8^{\log^* k})$: Harvey–vdH–Lecerf, J. of Complexity 2016.
- For $10 \lessapprox k \lessapprox 1000$, intermediate regimes apply: Karatsuba, Toom–Cook, etc.

How does this translate in practice?

- How to minimize α for small $k \lessapprox 10$?
- How does α depend on the architecture?

Theoretical efficiency of multiple precision floating point arithmetic

- Mainly boils down to the complexity $\mathsf{M}(k)$ of k-"word" multiplication.
- For small k, we have $\mathsf{M}(k) \approx \alpha\, k^2$ for some constant α.
- For large k, we have $\mathsf{M}(k) = O(k \log k\, 8^{\log^* k})$: Harvey–vdH–Lecerf, J. of Complexity 2016.
- For $10 \lessapprox k \lessapprox 1000$, intermediate regimes apply: Karatsuba, Toom–Cook, etc.

How does this translate in practice?

- How to minimize α for small $k \lessapprox 10$?
- How does α depend on the architecture?
- What about SIMD-style vectorization?

Theoretical efficiency of multiple precision floating point arithmetic

- Mainly boils down to the complexity $\mathsf{M}(k)$ of k-“word” multiplication.
- For small k, we have $\mathsf{M}(k) \approx \alpha\, k^2$ for some constant α.
- For large k, we have $\mathsf{M}(k) = O(k \log k\, 8^{\log^* k})$: Harvey–vdH–Lecerf, J. of Complexity 2016.
- For $10 \lessapprox k \lessapprox 1000$, intermediate regimes apply: Karatsuba, Toom–Cook, etc.

How does this translate in practice?

- How to minimize α for small $k \lessapprox 10$?
- How does α depend on the architecture?
- What about SIMD-style vectorization?
- To what extent do additions and subtractions matter?

The issues

- What are the available word sizes μ in bits?

The issues

- What are the available word sizes μ in bits?
- What is the available SIMD width w for the best word size μ?

The issues

- What are the available word sizes μ in bits?
- What is the available SIMD width w for the best word size μ?

$$\text{Efficiency} \propto \mu^2 w$$

The issues

- What are the available word sizes μ in bits?
- What is the available SIMD width w for the best word size μ?

$$\text{Efficiency} \propto \mu^2 w$$

- Is (efficient) hardware integer arithmetic available? If so,

$$\text{Efficiency}_{\mathbb{Z}} \approx \left(\frac{64}{53}\right)^2 \text{Efficiency}_{\mathbb{F}} \approx 1.5\, \text{Efficiency}_{\mathbb{F}}.$$

The issues

- What are the available word sizes μ in bits?
- What is the available SIMD width w for the best word size μ?

$$\text{Efficiency} \propto \mu^2 w$$

- Is (efficient) hardware integer arithmetic available? If so,

$$\text{Efficiency}_{\mathbb{Z}} \approx \left(\frac{64}{53}\right)^2 \text{Efficiency}_{\mathbb{F}} \approx 1.5\, \text{Efficiency}_{\mathbb{F}}.$$

Currently

- $\mu = 53$ and $w = 4$ on AVX2-enabled processors. No efficient 64-bit integer arithmetic.
- $\mu = 24$ and $16 \leqslant w \leqslant 64$ on cheap GPUs. No efficient 32-bit integer arithmetic.
- $\mu = 53$ and $16 \leqslant w \leqslant 64$ on expensive GPUs. No efficient 64-bit integer arithmetic.
- FGPAs: not considered here.

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$
- Exploit error-free transformations when doing operations

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$
- Exploit error-free transformations when doing operations
- Efficient for $k = 2$, $k = 3$, and $k = 4$ (cf. QD library by Bailey et al.)

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$
- Exploit error-free transformations when doing operations
- Efficient for $k = 2$, $k = 3$, and $k = 4$ (cf. QD library by Bailey et al.)
- Muller–Popescu–Tang, ARITH 2016, "op-count" $\mathrm{M}(k) \leqslant \frac{13}{2} k^2 + \frac{45}{2} k + 67$

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$
- Exploit error-free transformations when doing operations
- Efficient for $k = 2$, $k = 3$, and $k = 4$ (cf. QD library by Bailey et al.)
- Muller–Popescu–Tang, ARITH 2016, "op-count" $\mathrm{M}(k) \leqslant \frac{13}{2} k^2 + \frac{45}{2} k + 67$

Separate treatment of mantissas and exponents

- Standard representation $x = m\, 2^e$ with $m = m_0 + m_1\, 2^{-p} + \cdots + m_{k-1}\, 2^{-(k-1)p}$

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$
- Exploit error-free transformations when doing operations
- Efficient for $k = 2$, $k = 3$, and $k = 4$ (cf. QD library by Bailey et al.)
- Muller–Popescu–Tang, ARITH 2016, "op-count" $\mathrm{M}(k) \leqslant \frac{13}{2} k^2 + \frac{45}{2} k + 67$

Separate treatment of mantissas and exponents

- Standard representation $x = m\, 2^e$ with $m = m_0 + m_1\, 2^{-p} + \cdots + m_{k-1}\, 2^{-(k-1)p}$
- Fixed-point arithmetic operations on the mantissas m

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$
- Exploit error-free transformations when doing operations
- Efficient for $k = 2$, $k = 3$, and $k = 4$ (cf. QD library by Bailey et al.)
- Muller–Popescu–Tang, ARITH 2016, "op-count" $\mathsf{M}(k) \leqslant \frac{13}{2} k^2 + \frac{45}{2} k + 67$

Separate treatment of mantissas and exponents

- Standard representation $x = m\, 2^e$ with $m = m_0 + m_1 2^{-p} + \cdots + m_{k-1} 2^{-(k-1)p}$
- Fixed-point arithmetic operations on the mantissas m
- Very efficient for large k (cf. GMP and MPFR libraries)

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$
- Exploit error-free transformations when doing operations
- Efficient for $k = 2$, $k = 3$, and $k = 4$ (cf. QD library by Bailey et al.)
- Muller–Popescu–Tang, ARITH 2016, "op-count" $\mathrm{M}(k) \leqslant \frac{13}{2}k^2 + \frac{45}{2}k + 67$

Separate treatment of mantissas and exponents

- Standard representation $x = m\,2^e$ with $m = m_0 + m_1\,2^{-p} + \cdots + m_{k-1}\,2^{-(k-1)p}$
- Fixed-point arithmetic operations on the mantissas m
- Very efficient for large k (cf. GMP and MPFR libraries)
- However: GMP and MPFR are currently not vectorized and very inefficient for small k

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$
- Exploit error-free transformations when doing operations
- Efficient for $k = 2$, $k = 3$, and $k = 4$ (cf. QD library by Bailey et al.)
- Muller–Popescu–Tang, ARITH 2016, "op-count" $\mathrm{M}(k) \leqslant \frac{13}{2} k^2 + \frac{45}{2} k + 67$

Separate treatment of mantissas and exponents

- Standard representation $x = m\, 2^e$ with $m = m_0 + m_1 2^{-p} + \cdots + m_{k-1} 2^{-(k-1)p}$
- Fixed-point arithmetic operations on the mantissas m
- Very efficient for large k (cf. GMP and MPFR libraries)
- However: GMP and MPFR are currently not vectorized and very inefficient for small k
- Efficiency for small and medium k?

Notation: $\mathbb{F}$ is the set of hardware floating point numbers

Floating point expansions

- Based on an exact representation $x = x_1 + \cdots + x_k$ with $x_1, \ldots, x_k \in \mathbb{F}$
- Exploit error-free transformations when doing operations
- Efficient for $k = 2$, $k = 3$, and $k = 4$ (cf. QD library by Bailey et al.)
- Muller–Popescu–Tang, ARITH 2016, "op-count" $\mathrm{M}(k) \leqslant \frac{13}{2} k^2 + \frac{45}{2} k + 67$

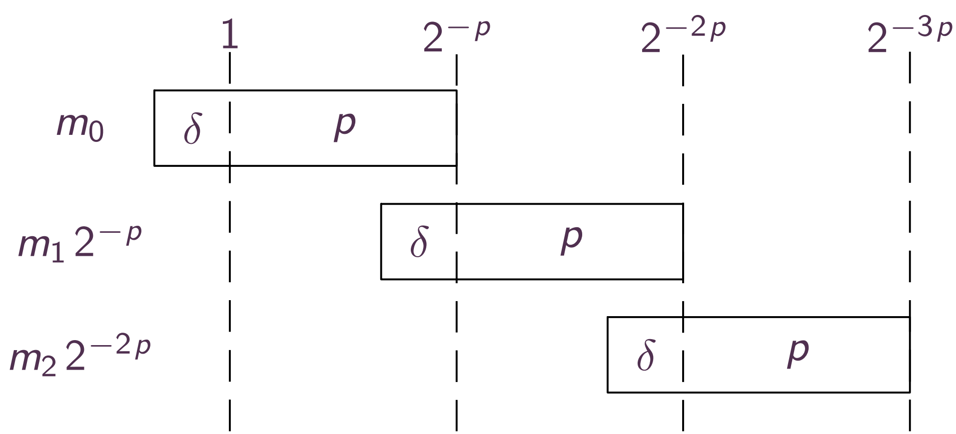
Separate treatment of mantissas and exponents

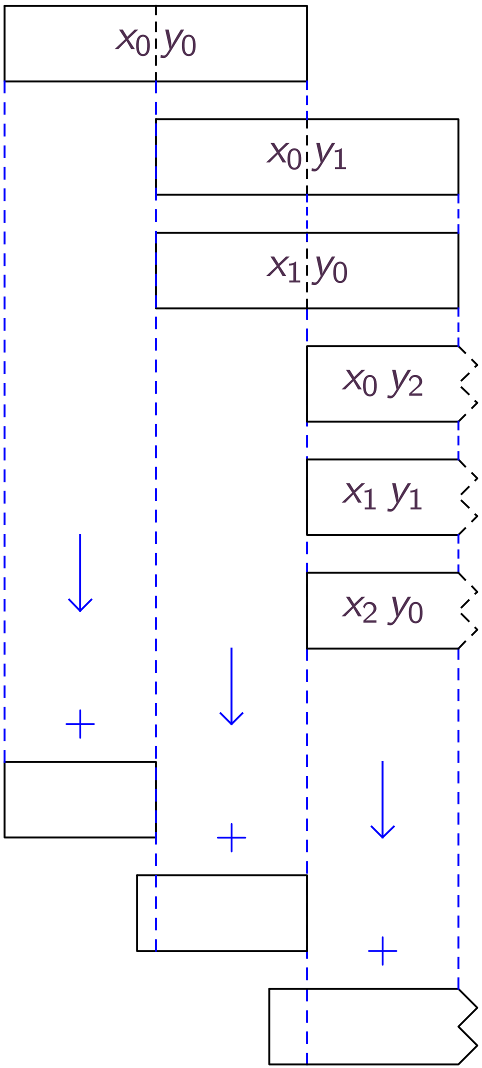
- Standard representation $x = m\, 2^e$ with $m = m_0 + m_1 2^{-p} + \cdots + m_{k-1} 2^{-(k-1)p}$
- Fixed-point arithmetic operations on the mantissas m
- Very efficient for large k (cf. GMP and MPFR libraries)
- However: GMP and MPFR are currently not vectorized and very inefficient for small k
- Efficiency for small and medium k?
- SIMD-style vectorization?

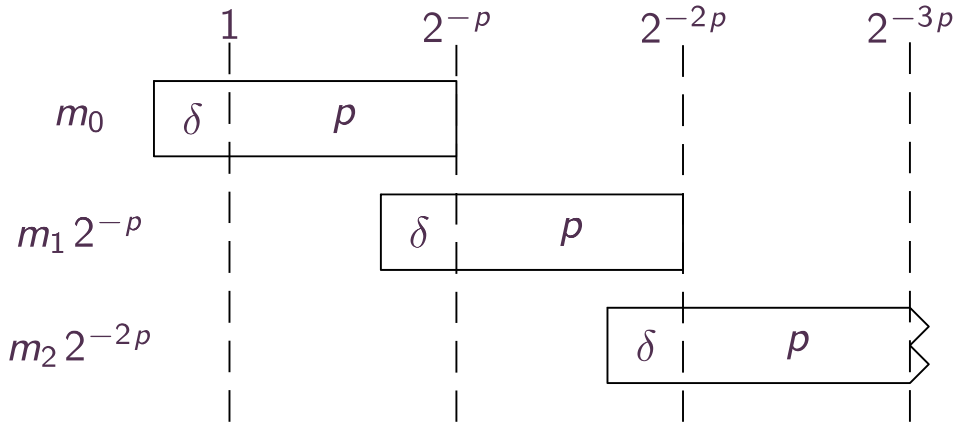
Idea (vdH–Lecerf, ARITH 2015): use redundant representation with "nail bits":

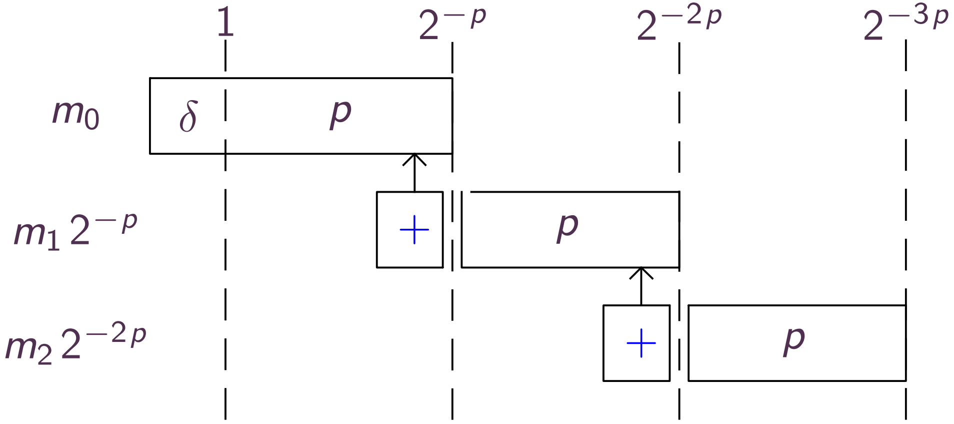
$$\begin{aligned} x &= m\,2^e \\ m &= m_0 + m_1\,2^{-p} + \cdots + m_{k-1}\,2^{-(k-1)p} \\ p &= \mu - \delta \\ \delta &\approx 4, \text{ suitable number of "nail bits"} \\ m_i &\in \mathbb{Z}\,2^{-p} \end{aligned}$$

Thus: $|m| < 2^{\delta}$ and $m \in \mathbb{Z}\,2^{-kp}$.

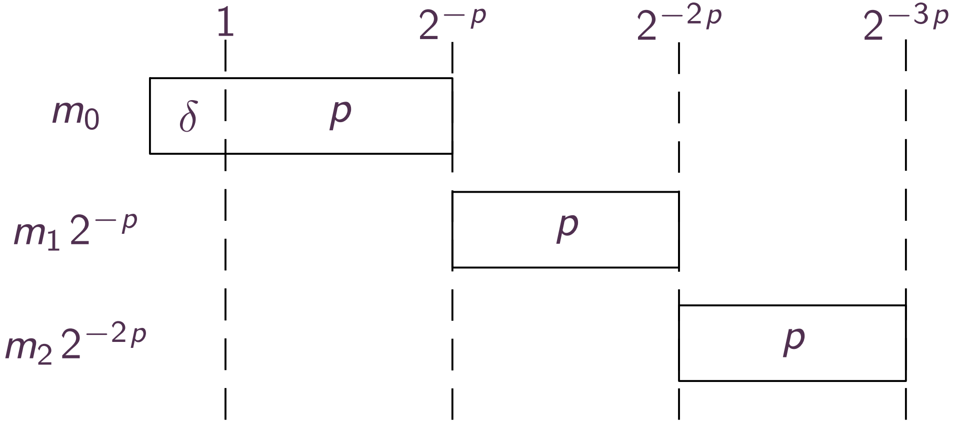


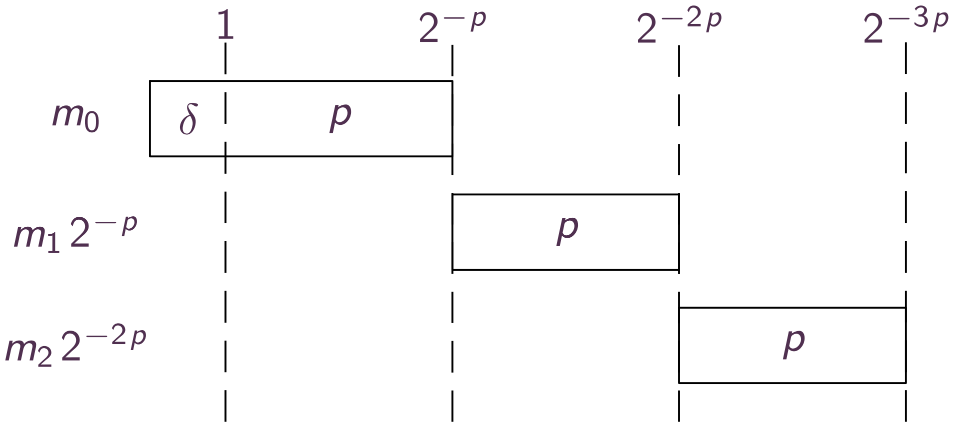

$x_0\,y_0$
$x_0\,y_1$
$x_1\,y_0$
$x_0\,y_2$
$x_1\,y_1$
$x_2\,y_0$
+
+
+


1
2^{-p}
2^{-2p}
2^{-3p}
m_0
δ
p
$m_1 2^{-p}$
δ
p
$m_2 2^{-2p}$
δ
p



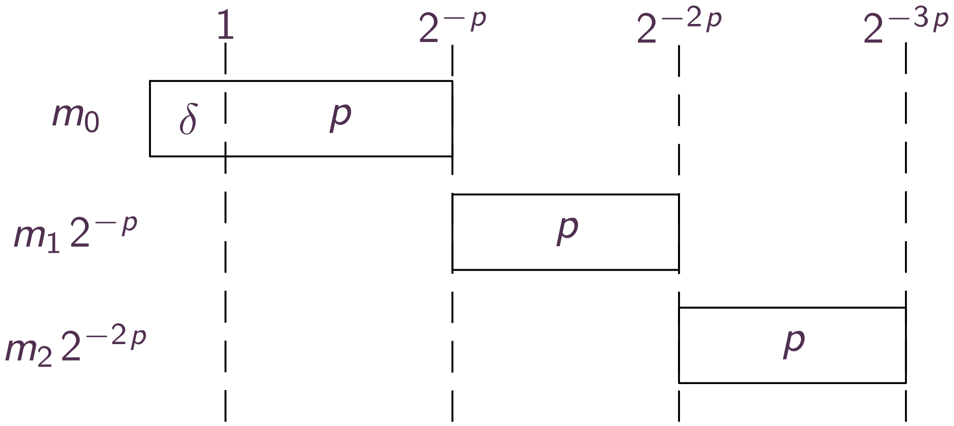
1
2^{-p}
2^{-2p}
2^{-3p}
m_0
δ
p
$m_1 2^{-p}$
$+$
p
$m_2 2^{-2p}$
$+$
p


1
2^{-p}
2^{-2p}
2^{-3p}
m_0
δ
p
$m_1 2^{-p}$
p
$m_2 2^{-2p}$
p



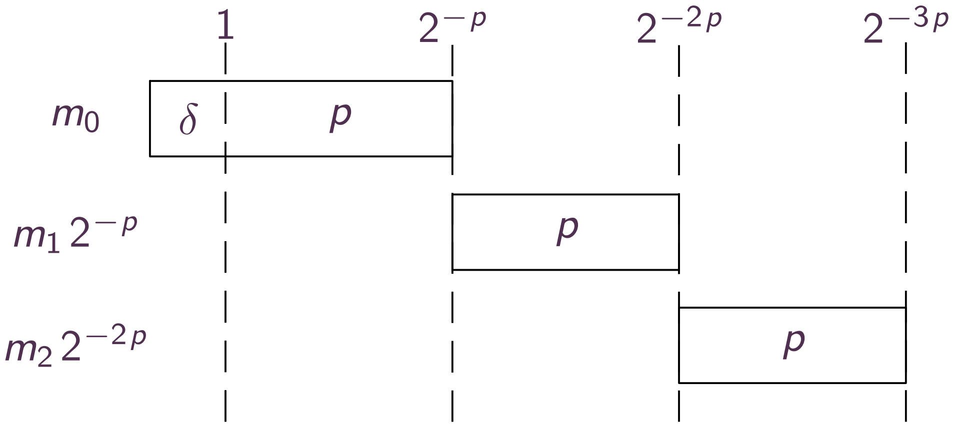
Operation counts

- Multiplication: $5\binom{k}{2}+1=\frac{5}{2}\,k^2-\frac{5}{2}\,k+1$



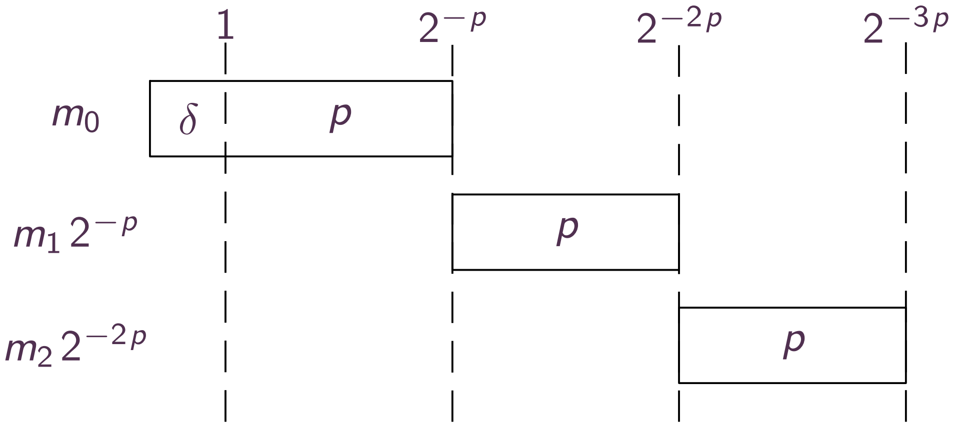
Operation counts

- Multiplication: $5\binom{k}{2} + 1 = \frac{5}{2}k^2 - \frac{5}{2}k + 1$
- Carry-normalization: $4k - 4$



Operation counts

- Multiplication: $5\binom{k}{2}+1=\frac{5}{2}k^2-\frac{5}{2}k+1$
- Carry-normalization: $4k-4$
- Total: $\frac{5}{2}k^2+\frac{3}{2}k-3$



Operation counts

- Multiplication: $5\binom{k}{2} + 1 = \frac{5}{2}\,k^2 - \frac{5}{2}\,k + 1$
- Carry-normalization: $4\,k - 4$
- Total: $\frac{5}{2}\,k^2 + \frac{3}{2}\,k - 3$
- Remember: $\frac{13}{2}\,k^2 + \frac{45}{2}\,k + 67$

Main problem: putting arguments under a common exponent (e.g. $0.7 \times 2^{-7} + 0.8 \times 2^{-12}$)

Main problem: putting arguments under a common exponent (e.g. $0.7 \times 2^{-7} + 0.8 \times 2^{-12}$)

$\rightsquigarrow$ how to shift mantissas efficiently? (e.g. $0.8 \times 2^{-12} = 0.025 \times 2^{-7}$)

Main problem: putting arguments under a common exponent (e.g. $0.7 \times 2^{-7} + 0.8 \times 2^{-12}$)

$\rightsquigarrow$ how to shift mantissas efficiently? (e.g. $0.8 \times 2^{-12} = 0.025 \times 2^{-7}$)

$\rightsquigarrow$ how to perform "dot normalization"?

Main problem: putting arguments under a common exponent (e.g. $0.7 \times 2^{-7} + 0.8 \times 2^{-12}$)

$\rightsquigarrow$ how to shift mantissas efficiently? (e.g. $0.8 \times 2^{-12} = 0.025 \times 2^{-7}$)

$\rightsquigarrow$ how to perform "dot normalization"?

Decomposition of a shift by s bits

Main problem: putting arguments under a common exponent (e.g. $0.7 \times 2^{-7} + 0.8 \times 2^{-12}$)

$\rightsquigarrow$ how to shift mantissas efficiently? (e.g. $0.8 \times 2^{-12} = 0.025 \times 2^{-7}$)

$\rightsquigarrow$ how to perform "dot normalization"?

Decomposition of a shift by s bits

- as a long shift by $\sigma = \lfloor s / p \rfloor$ words

Main problem: putting arguments under a common exponent (e.g. $0.7 \times 2^{-7} + 0.8 \times 2^{-12}$)

$\rightsquigarrow$ how to shift mantissas efficiently? (e.g. $0.8 \times 2^{-12} = 0.025 \times 2^{-7}$)

$\rightsquigarrow$ how to perform "dot normalization"?

Decomposition of a shift by s bits

- as a long shift by $\sigma = \lfloor s / p \rfloor$ words
- and a short shift by $s' = s - \sigma p < p$ bits

Main problem: putting arguments under a common exponent (e.g. $0.7\times 2^{-7}+0.8\times 2^{-12}$)

$\rightsquigarrow$ how to shift mantissas efficiently? (e.g. $0.8\times 2^{-12}=0.025\times 2^{-7}$)

$\rightsquigarrow$ how to perform "dot normalization"?

Decomposition of a shift by s bits

- as a long shift by $\sigma=\lfloor s/p\rfloor$ words
- and a short shift by $s'=s-\sigma p<p$ bits
- This should be done using SIMD vector instructions

Main problem: putting arguments under a common exponent (e.g. $0.7 \times 2^{-7} + 0.8 \times 2^{-12}$)

$\rightsquigarrow$ how to shift mantissas efficiently? (e.g. $0.8 \times 2^{-12} = 0.025 \times 2^{-7}$)

$\rightsquigarrow$ how to perform "dot normalization"?

Decomposition of a shift by s bits

- as a long shift by $\sigma = \lfloor s / p \rfloor$ words
- and a short shift by $s' = s - \sigma p < p$ bits
- This should be done using SIMD vector instructions

Main design decisions to be made

- Work with arbitrary exponents (*à la* MPFR) or multiples of p (*à la* GMP)?

Main problem: putting arguments under a common exponent (e.g. $0.7\times 2^{-7}+0.8\times 2^{-12}$)

$\rightsquigarrow$ how to shift mantissas efficiently? (e.g. $0.8\times 2^{-12}=0.025\times 2^{-7}$)

$\rightsquigarrow$ how to perform "dot normalization"?

Decomposition of a shift by s bits

- as a long shift by $\sigma=\lfloor s/p\rfloor$ words
- and a short shift by $s'=s-\sigma p<p$ bits
- This should be done using SIMD vector instructions

Main design decisions to be made

- Work with arbitrary exponents (*à la* MPFR) or multiples of p (*à la* GMP)?
- Numbers in an SIMD vector share the same exponent or not?

Idea: any shift by $\sigma = \sigma_0 + \sigma_1 2 + \cdots + \sigma_{\ell-1} 2^{\ell-1}$ words with $\sigma_i \in \{0, 1\}$
decomposes as ℓ special shifts by $\sigma_i 2^i \in \{0, 2^i\}$ words (done using `blend` instruction)

$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$m_{0,4}$	$m_{0,5}$	$m_{0,6}$	$m_{0,7}$
$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$m_{1,4}$	$m_{1,5}$	$m_{1,6}$	$m_{1,7}$
$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$m_{2,4}$	$m_{2,5}$	$m_{2,6}$	$m_{2,7}$
$m_{3,0}$	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$m_{3,4}$	$m_{3,5}$	$m_{3,6}$	$m_{3,7}$
$m_{4,0}$	$m_{4,1}$	$m_{4,2}$	$m_{4,3}$	$m_{4,4}$	$m_{4,5}$	$m_{4,6}$	$m_{4,7}$
$m_{5,0}$	$m_{5,1}$	$m_{5,2}$	$m_{5,3}$	$m_{5,4}$	$m_{5,5}$	$m_{5,6}$	$m_{5,7}$
$m_{6,0}$	$m_{6,1}$	$m_{6,2}$	$m_{6,3}$	$m_{6,4}$	$m_{6,5}$	$m_{6,6}$	$m_{6,7}$
$m_{7,0}$	$m_{7,1}$	$m_{7,2}$	$m_{7,3}$	$m_{7,4}$	$m_{7,5}$	$m_{7,6}$	$m_{7,7}$

Shift by	σ_0	σ_1	σ_2
3	1	1	0
2	0	1	0
5	1	0	1
11	1	1	1
0	0	0	0
4	0	0	1
4	0	0	1
6	0	1	1

Operation count: $k \log_2 k$

Idea: any shift by $\sigma = \sigma_0 + \sigma_1 2 + \cdots + \sigma_{\ell-1} 2^{\ell-1}$ words with $\sigma_i \in \{0, 1\}$
decomposes as ℓ special shifts by $\sigma_i 2^i \in \{0, 2^i\}$ words (done using `blend` instruction)

0	$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$m_{0,4}$	$m_{0,5}$	$m_{0,6}$
$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$m_{1,4}$	$m_{1,5}$	$m_{1,6}$	$m_{1,7}$
0	$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$m_{2,4}$	$m_{2,5}$	$m_{2,6}$
0	$m_{3,0}$	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$m_{3,4}$	$m_{3,5}$	$m_{3,6}$
$m_{4,0}$	$m_{4,1}$	$m_{4,2}$	$m_{4,3}$	$m_{4,4}$	$m_{4,5}$	$m_{4,6}$	$m_{4,7}$
$m_{5,0}$	$m_{5,1}$	$m_{5,2}$	$m_{5,3}$	$m_{5,4}$	$m_{5,5}$	$m_{5,6}$	$m_{5,7}$
$m_{6,0}$	$m_{6,1}$	$m_{6,2}$	$m_{6,3}$	$m_{6,4}$	$m_{6,5}$	$m_{6,6}$	$m_{6,7}$
$m_{7,0}$	$m_{7,1}$	$m_{7,2}$	$m_{7,3}$	$m_{7,4}$	$m_{7,5}$	$m_{7,6}$	$m_{7,7}$

Shift by	σ_0	σ_1	σ_2
3	1	1	0
2	0	1	0
5	1	0	1
11	1	1	1
0	0	0	0
4	0	0	1
4	0	0	1
6	0	1	1

Operation count: $k \log_2 k$

long SIMD shifts

Idea: any shift by $\sigma = \sigma_0 + \sigma_1 2 + \cdots + \sigma_{\ell-1} 2^{\ell-1}$ words with $\sigma_i \in \{0, 1\}$
decomposes as ℓ special shifts by $\sigma_i 2^i \in \{0, 2^i\}$ words (done using `blend` instruction)

0	0	0	$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$m_{0,4}$
0	0	$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$m_{1,4}$	$m_{1,5}$
0	$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$m_{2,4}$	$m_{2,5}$	$m_{2,6}$
0	0	0	$m_{3,0}$	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$m_{3,4}$
$m_{4,0}$	$m_{4,1}$	$m_{4,2}$	$m_{4,3}$	$m_{4,4}$	$m_{4,5}$	$m_{4,6}$	$m_{4,7}$
$m_{5,0}$	$m_{5,1}$	$m_{5,2}$	$m_{5,3}$	$m_{5,4}$	$m_{5,5}$	$m_{5,6}$	$m_{5,7}$
$m_{6,0}$	$m_{6,1}$	$m_{6,2}$	$m_{6,3}$	$m_{6,4}$	$m_{6,5}$	$m_{6,6}$	$m_{6,7}$
0	0	$m_{7,0}$	$m_{7,1}$	$m_{7,2}$	$m_{7,3}$	$m_{7,4}$	$m_{7,5}$

Shift by	σ_0	σ_1	σ_2
3	1	1	0
2	0	1	0
5	1	0	1
11	1	1	1
0	0	0	0
4	0	0	1
4	0	0	1
6	0	1	1

Operation count: $k \log_2 k$

Idea: any shift by $\sigma = \sigma_0 + \sigma_1 2 + \cdots + \sigma_{\ell-1} 2^{\ell-1}$ words with $\sigma_i \in \{0,1\}$ decomposes as ℓ special shifts by $\sigma_i 2^i \in \{0, 2^i\}$ words (done using `blend` instruction)

0	0	0	$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$m_{0,4}$
0	0	$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$m_{1,4}$	$m_{1,5}$
0	0	0	0	0	$m_{2,0}$	$m_{2,1}$	$m_{2,2}$
0	0	0	0	0	0	0	$m_{3,0}$
$m_{4,0}$	$m_{4,1}$	$m_{4,2}$	$m_{4,3}$	$m_{4,4}$	$m_{4,5}$	$m_{4,6}$	$m_{4,7}$
0	0	0	0	$m_{5,0}$	$m_{5,1}$	$m_{5,2}$	$m_{5,3}$
0	0	0	0	$m_{6,0}$	$m_{6,1}$	$m_{6,2}$	$m_{6,3}$
0	0	0	0	0	0	$m_{7,0}$	$m_{7,1}$

Shift by	σ_0	σ_1	σ_2
3	1	1	0
2	0	1	0
5	1	0	1
11	1	1	1
0	0	0	0
4	0	0	1
4	0	0	1
6	0	1	1

Operation count: $k \log_2 k$

Idea: any shift by $\sigma = \sigma_0 + \sigma_1 2 + \cdots + \sigma_{\ell-1} 2^{\ell-1}$ words with $\sigma_i \in \{0, 1\}$
decomposes as ℓ special shifts by $\sigma_i 2^i \in \{0, 2^i\}$ words (done using `blend` instruction)

								Shift by	σ_0	σ_1	σ_2
0	0	0	$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$m_{0,4}$	3	1	1	0
0	0	$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$m_{1,4}$	$m_{1,5}$	2	0	1	0
0	0	0	0	0	$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	5	1	0	1
0	0	0	0	0	0	0	$m_{3,0}$	11	1	1	1
$m_{4,0}$	$m_{4,1}$	$m_{4,2}$	$m_{4,3}$	$m_{4,4}$	$m_{4,5}$	$m_{4,6}$	$m_{4,7}$	0	0	0	0
0	0	0	0	$m_{5,0}$	$m_{5,1}$	$m_{5,2}$	$m_{5,3}$	4	0	0	1
0	0	0	0	$m_{6,0}$	$m_{6,1}$	$m_{6,2}$	$m_{6,3}$	4	0	0	1
0	0	0	0	0	0	$m_{7,0}$	$m_{7,1}$	6	0	1	1

Operation count: $k \log_2 k$

Similar to carry-normalization

Operation count: $4\,k - 1$

Similar to carry-normalization

Operation count: $4\,k-1$

Note

One addition $r=x+y$ requires

- One general right shift for x (put under common exponent)
- One general right shift for y (put under common exponent)
- One fixed-point addition
- One general left shift for r (dot normalization)

Base 2

	k	2	3	4	5	6	7	8	9	10	11	12
$\pm$	Individual exponents	51	84	108	150	177	204	231	288	318	348	378
	Shared exponents	55	79	103	127	151	175	199	223	247	271	295
$\times$	Individual exponents	31	35	54	78	107	141	180	224	273	327	386
	Shared exponents	32	36	55	79	108	142	181	225	274	328	387
$\times$	FP expansions	138	193	261	342	436	543	663	796	942	1101	1273

Base 2^p

	$k-1$	2	3	4	5	6	7	8	9	10	11	12
$\pm$	Individual exponents	31	49	67	92	107	122	147	183	201	219	237
	Shared exponents	31	43	55	67	79	91	103	115	127	139	151
$\times$	Individual exponents	40	61	87	118	154	195	241	292	348	409	475
	Shared exponents	41	62	88	119	155	196	242	293	349	410	476
$\times$	FP expansions	138	193	261	342	436	543	663	796	942	1101	1273

Conclusion

- ARITH 2015: our multiple precision arithmetic is very efficient for fixed-point arithmetic

  We were able to achieve $\alpha \leqslant 2$ for practical FFT computations

This was really our best case situation

Conclusion

- ARITH 2015: our multiple precision arithmetic is very efficient for fixed-point arithmetic

  We were able to achieve $\alpha \leqslant 2$ for practical FFT computations

  This was really our best case situation

- ARITH 2017: we expect our approach to outperform floating point expansions for $k \geqslant 5$

  This holds for any of the known approaches: Priest, Bailey, Muller–Popescu–Tang, ...

  Although this is really our worst case situation

Conclusion

- ARITH 2015: our multiple precision arithmetic is very efficient for fixed-point arithmetic

  We were able to achieve $\alpha \leqslant 2$ for practical FFT computations

  This was really our best case situation

- ARITH 2017: we expect our approach to outperform floating point expansions for $k \geqslant 5$

  This holds for any of the known approaches: Priest, Bailey, Muller–Popescu–Tang, ...

  Although this is really our worst case situation

Perspectives

- To make better use of our arithmetic, one should implement dedicated functions for
  - Sums $x_1 + \cdots + x_t$ of several numbers
  - Important specific operations: FFT, matrix multiplication, etc.
  - Etc.
- Can compilers use such optimized routines automatically when possible?