On the computation of the reciprocal of floating point expansions using an adapted Newton-Raphson iteration

Mioara Joldes, Valentina Popescu, Jean-Michel Muller

ASAP 2014

Motivation

- Numerical problems require floating-point operations with higher precision than double (binary64)

Motivation

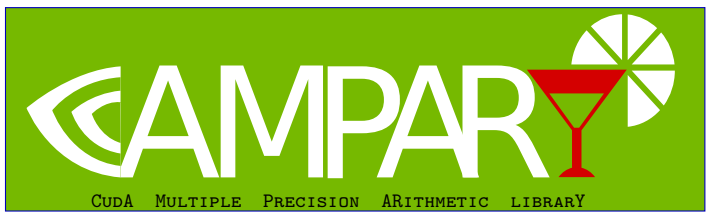
- Numerical problems require floating-point operations with higher precision than double (binary64)
- e.g. in dynamical systems like: finding sinks in Hénon map, iterating Lorenz attractor, long term stability of the solar system

Motivation

- Numerical problems require floating-point operations with higher precision than double (binary64)
- e.g. in dynamical systems like: finding sinks in Hénon map, iterating Lorenz attractor, long term stability of the solar system
- These are usually high performance computing problems also.

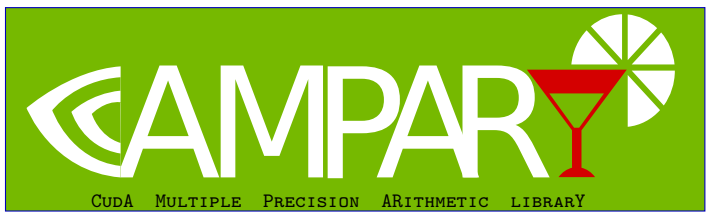
Motivation

- Numerical problems require floating-point operations with higher precision than double (binary64)
- e.g. in dynamical systems like: finding sinks in Hénon map, iterating Lorenz attractor, long term stability of the solar system
- These are usually high performance computing problems also.
- Our project:



Motivation

- Numerical problems require floating-point operations with higher precision than double (binary64)
- e.g. in dynamical systems like: finding sinks in Hénon map, iterating Lorenz attractor, long term stability of the solar system
- These are usually high performance computing problems also.
- Our project:



Existing Multiple-Precision Libraries (on CPU/GPU architectures)

- Multiple-digits representation:
  GNU MPFR, ARPREC,
  GPU variants: GARPREC,
  CUMP

Motivation

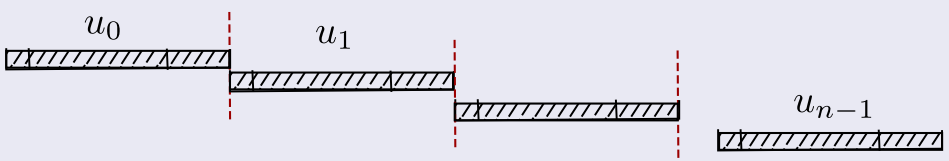
- Numerical problems require floating-point operations with higher precision than double (binary64)
- e.g. in dynamical systems like: finding sinks in Hénon map, iterating Lorenz attractor, long term stability of the solar system
- These are usually high performance computing problems also.
- Our project:

CAMPARY
CudA Multiple Precision ARithmetic libraRY

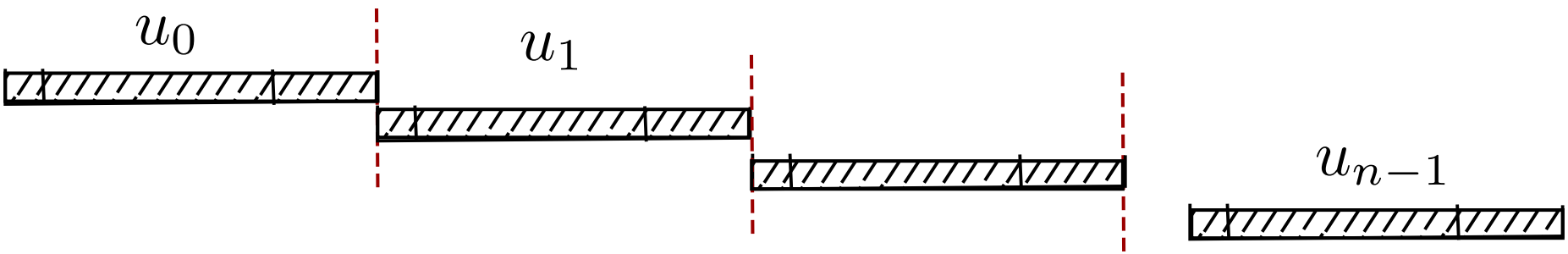
Existing Multiple-Precision Libraries (on CPU/GPU architectures)

- Multiple-digits representation:
  GNU MPFR, ARPREC,
  GPU variants: GARPREC,
  CUMP

- Multiple-terms representation:
  QD,
  GPU variant: GQD

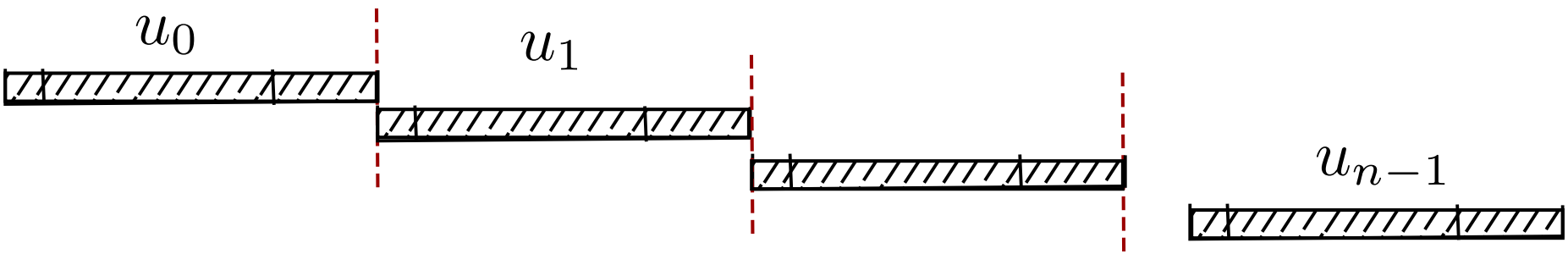


Extending the precision using multiple-terms format: FP expansions



Extending the precision using multiple-terms format: FP expansions

Let a precision-p Floating-point (FP) number $x = M_x \cdot 2^{e_x}$, with $1 \leq |M_x| < 2$. Denote $\mathsf{ulp}\,(x) = 2^{e_x - p + 1}$ (Goldberg's def).

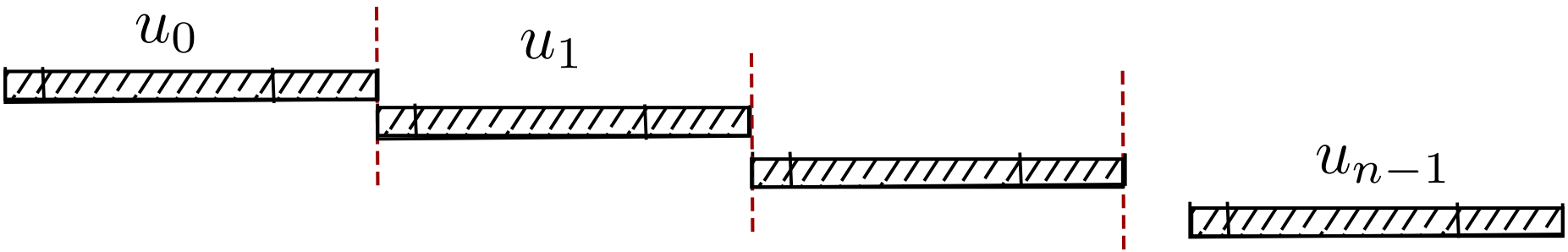


Extending the precision using multiple-terms format: FP expansions

Let a precision-p Floating-point (FP) number $x = M_x \cdot 2^{e_x}$, with $1 \leq |M_x| < 2$. Denote $\text{ulp}(x) = 2^{e_x - p + 1}$ (Goldberg's def).

Def.

A *floating-point expansion* u with n terms is the unevaluated sum $u = \sum_{i=0}^{n-1} u_i$ of n FP numbers $u_0, \ldots, u_{n-1}$, s.t. $u_i \neq 0 \Rightarrow |u_i| \geq |u_{i+1}|$.

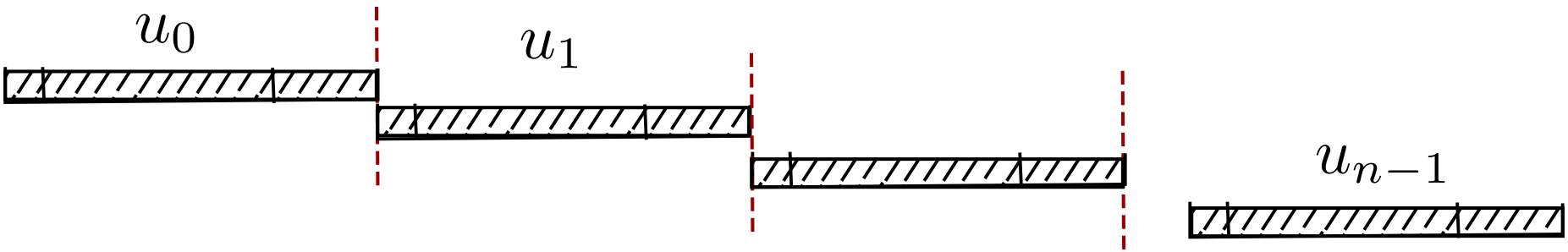


Extending the precision using multiple-terms format: FP expansions

Let a precision-p Floating-point (FP) number $x = M_x \cdot 2^{e_x}$, with $1 \leq |M_x| < 2$. Denote $\text{ulp}(x) = 2^{e_x - p + 1}$ (Goldberg's def).

Def.

A *floating-point expansion* u with n terms is the unevaluated sum $u = \sum_{i=0}^{n-1} u_i$ of n FP numbers $u_0, \ldots, u_{n-1}$, s.t. $u_i \neq 0 \Rightarrow |u_i| \geq |u_{i+1}|$.



Non-overlapping FP expansions

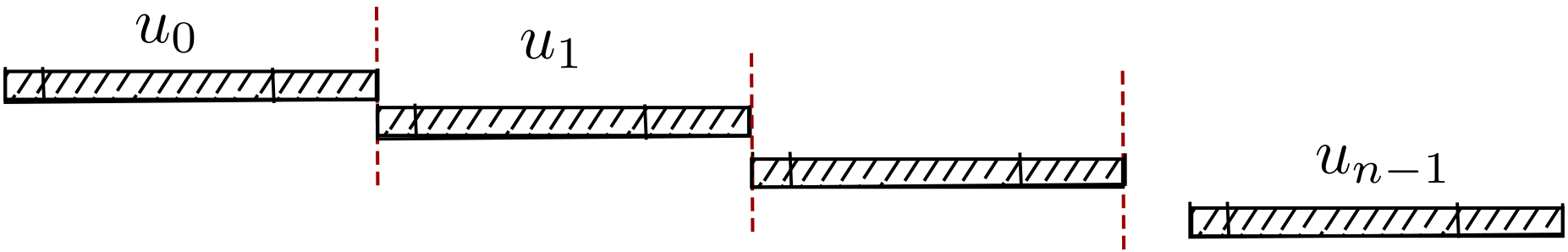
u is *Bailey-nonoverlapping* if for all $0 < i < n$, we have $|u_i| \leq \frac{1}{2}\,\text{ulp}(u_{i-1})$.

Extending the precision using multiple-terms format: FP expansions

Let a precision-p Floating-point (FP) number $x = M_x \cdot 2^{e_x}$, with $1 \leq |M_x| < 2$. Denote $\text{ulp}(x) = 2^{e_x - p + 1}$ (Goldberg's def).

Def.

A *floating-point expansion* u with n terms is the unevaluated sum $u = \sum_{i=0}^{n-1} u_i$ of n FP numbers $u_0, \ldots, u_{n-1}$, s.t. $u_i \neq 0 \Rightarrow |u_i| \geq |u_{i+1}|$.



Non-overlapping FP expansions

u is *Bailey-nonoverlapping* if for all $0 < i < n$, we have $|u_i| \leq \frac{1}{2}\,\text{ulp}(u_{i-1})$.

Example: π in double-double

$$p_0 = 11.0010010000111111011010101000100010000101101000110000_2,$$

and

$$p_1 = 1.0001101001100010011000110011000101000101110000000111_2 \times 2^{-53}.$$

$p_0 + p_1 \leftrightarrow 107$ bits FP approx.

Extending the precision using FP expansions

- Pros:
  - use directly available and highly optimized native FP operations
  - sufficiently simple and regular algorithms for addition and multiplication
  - straightforwardly portable to highly parallel architectures, such as GPUs.

Extending the precision using FP expansions

- Pros:
  - use directly available and highly optimized native FP operations
  - sufficiently simple and regular algorithms for addition and multiplication
  - straightforwardly portable to highly parallel architectures, such as GPUs.
- Cons:
  - lack of thorough error bounds
  - no correct rounding
  - QD only supports 2-doubles and 4-doubles format

Extending the precision using FP expansions

- Pros:
  - use directly available and highly optimized native FP operations
  - sufficiently simple and regular algorithms for addition and multiplication
  - straightforwardly portable to highly parallel architectures, such as GPUs.
- Cons:
  - lack of thorough error bounds
  - no correct rounding
  - QD only supports 2-doubles and 4-doubles format

Existing algorithms

- Addition and Multiplication:
  - generalized/adapted versions of [Priest'91], [Shewchuck'97], [Bailey'01]
  - based on Error-Free Transforms: 2Sum, 2Prod, 2ProdFMA

Extending the precision using FP expansions

- Pros:
  – use directly available and highly optimized native FP operations
  – sufficiently simple and regular algorithms for addition and multiplication
  – straightforwardly portable to highly parallel architectures, such as GPUs.
- Cons:
  – lack of thorough error bounds
  – no correct rounding
  – QD only supports 2-doubles and 4-doubles format

Existing algorithms

- Addition and Multiplication:
  – generalized/adapted versions of [Priest'91], [Shewchuck'97], [Bailey'01]
  – based on Error-Free Transforms: 2Sum, 2Prod, 2ProdFMA
- Division based on classical "paper and pencil" long division algorithm [Bailey'01, Priest'91, Daumas'99]

Our contribution: Reciprocal of FP expansions with an adapted Newton-Raphson Iteration & explicit error bound

Newton iteration for root α of f

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$,
when x_0 close to α, $f'(\alpha) \neq 0 \rightarrow$ quadratic convergence.

Our contribution: Reciprocal of FP expansions with an adapted Newton-Raphson Iteration & explicit error bound

Newton iteration for root α of f

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$,
when x_0 close to α, $f'(\alpha) \neq 0 \rightarrow$ quadratic convergence.

Newton-Raphson iteration for reciprocal i.e., root $1/a$ of $f(x) = 1/x - a$

$x_{n+1} = x_n(2 - ax_n)$,
when x_0 close to $1/a$, $\rightarrow$ quadratic convergence, $x_{n+1} - \frac{1}{a} = -a(x_n - \frac{1}{a})^2$.

Our contribution: Reciprocal of FP expansions with an adapted Newton-Raphson Iteration & explicit error bound

Newton iteration for root α of f

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$,
when x_0 close to α, $f'(\alpha) \neq 0 \rightarrow$ quadratic convergence.

Newton-Raphson iteration for reciprocal i.e., root $1/a$ of $f(x) = 1/x - a$

$x_{n+1} = x_n(2 - ax_n)$,
when x_0 close to $1/a$, $\rightarrow$ quadratic convergence, $x_{n+1} - \frac{1}{a} = -a(x_n - \frac{1}{a})^2$.

Adapted Newton-Raphson iteration for reciprocal of FP expansion:

FP Expansions

$$x_{n+1} = \underbrace{x_n \cdot \underbrace{(2 - \underbrace{a \cdot x_n}_{RdMulE})}_{RdSubE}}_{RdMulE}.$$

$\rightarrow$ quadratic convergence

Main Algorithm

Algorithm 1 Truncated Newton iteration based algorithm for reciprocal of an FP expansion.

Input: FP expansion $a = a_0 + \ldots + a_{2^k-1}$; length of output FP expansion 2^q.
Output: FP expansion $x = x_0 + \ldots + x_{2^q-1}$ s.t.

$$\left| x - \frac{1}{a} \right| \leq \frac{2^{-2^q(p-3)-1}}{|a|}. \tag{1}$$

1: $x_0 = \mathsf{RN}\,(1/a_0)$
2: **for** $i \leftarrow 0$ to $q-1$ **do**
3: $\hat{v}[0 : 2^{i+1}-1] \leftarrow RdMulE(x[0 : 2^i - 1],\, a[0 : 2^{i+1}-1],\, 2^{i+1})$
4: $\hat{w}[0 : 2^{i+1}-1] \leftarrow RdSubE(2,\, \hat{v}[0 : 2^{i+1}-1],\, 2^{i+1})$
5: $x[0 : 2^{i+1}-1] \leftarrow RdMulE(x[0 : 2^i - 1],\, \hat{w}[0 : 2^{i+1}-1],\, 2^{i+1})$
6: **end for**
7: **return** FP expansion $x = x_0 + \ldots + x_{2^q-1}$.

Example, adapted Newton-Raphson iteration on FP expansions

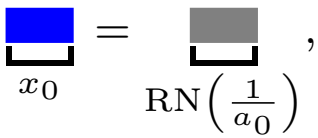
General procedure: $x_{n+1} = x_n(2 - ax_n)$

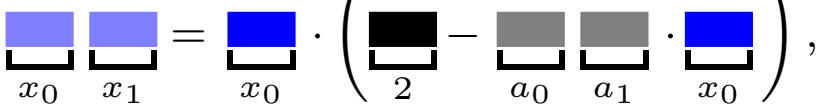
iter 0  $= \mathrm{RN}\left(\frac{1}{a_0}\right)$,

x_0

Example, adapted Newton-Raphson iteration on FP expansions

General procedure: $x_{n+1} = x_n(2 - ax_n)$

iter 0 

iter 1 

Example, adapted Newton-Raphson iteration on FP expansions

General procedure: $x_{n+1} = x_n(2 - ax_n)$

iter 0 $x_0 = \mathrm{RN}\left(\frac{1}{a_0}\right)$,

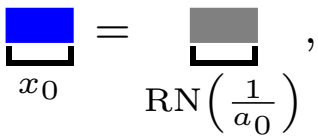
iter 1 $x_0\ x_1 = x_0 \cdot \left(2 - a_0\ a_1 \cdot x_0\right)$,

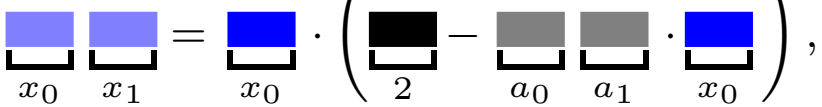
iter 2 $x_0\ x_1\ x_2\ x_3 = x_0\ x_1 \cdot \left(2 - a_0\ a_1\ a_2\ a_3 \cdot x_0\ x_1\right)$,

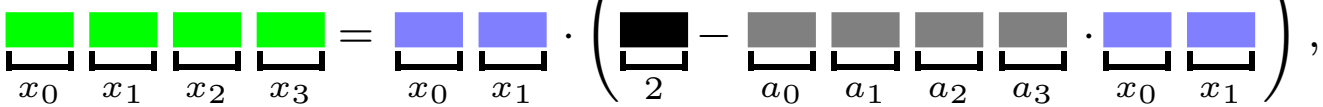
$\vdots$

Example, adapted Newton-Raphson iteration on FP expansions

General procedure: $x_{n+1} = x_n(2 - ax_n)$

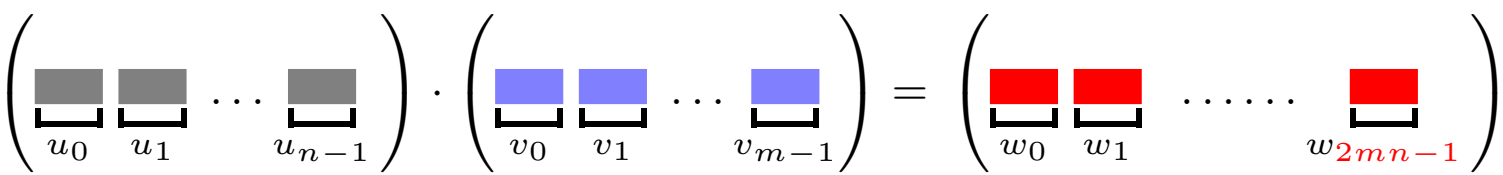
iter 0 

iter 1 

iter 2 

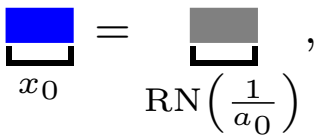
$\vdots$

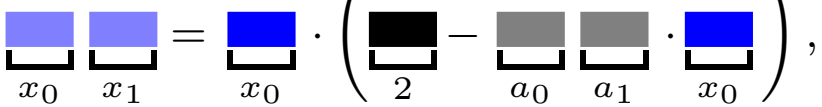
But... when using Error Free Transforms,

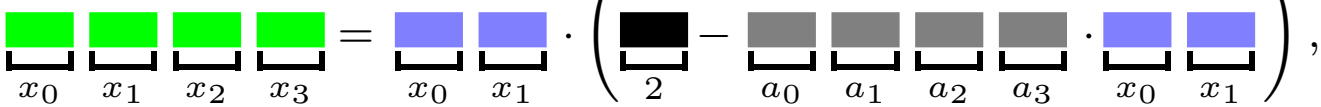


Example, adapted Newton-Raphson iteration on FP expansions

General procedure: $x_{n+1} = x_n(2 - ax_n)$

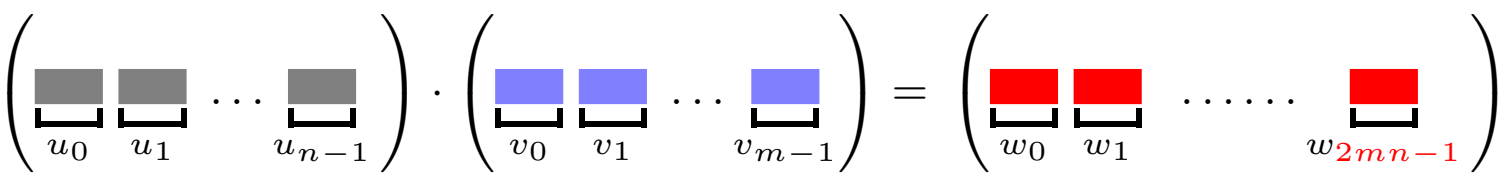
iter 0 

iter 1 

iter 2 

$\vdots$

But... when using Error Free Transforms,



Use truncated Addition/Multiplication.

Truncations and Error Analysis

General procedure: $x_{n+1} = x_n(2 - ax_n)$

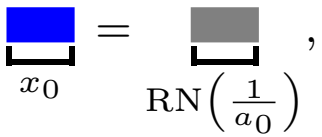
iter 0 $x_0 = \mathrm{RN}\left(\frac{1}{a_0}\right)$,

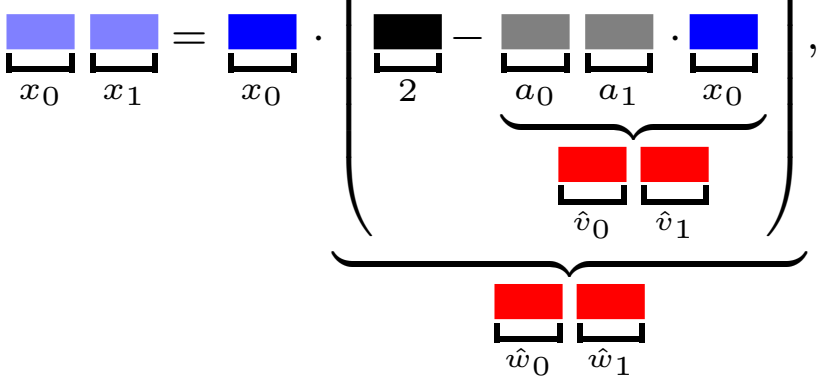
iter 1 $x_0\ x_1 = x_0 \cdot \underbrace{\left(2 - \underbrace{a_0\ a_1}_{\hat{v}_0\ \hat{v}_1} \cdot x_0 \right)}_{\hat{w}_0\ \hat{w}_1}$,

$\vdots$

Truncations and Error Analysis

General procedure: $x_{n+1} = x_n(2 - ax_n)$

iter 0 

iter 1 

$\vdots$

Error analysis based on triangular inequalities.

Let

$\eta = \sum_{j=0}^{\infty} 2^{(-j-1)p} = \frac{2^{-p}}{1-2^{-p}}$,

$\gamma_i = 2^{-(2^{i+1}-1)p} \frac{\eta}{1-\eta}$,

$$|x_{i+1} - \tau_i| \leq \gamma_i \left|x_i \cdot \hat{w}_i\right|, \tag{2a}$$

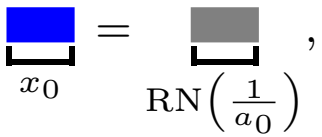
$$|w_i - \hat{w}_i| \leq \gamma_i \left|w_i\right| \leq \gamma_i \left|2 - \hat{v}_i\right|, \tag{2b}$$

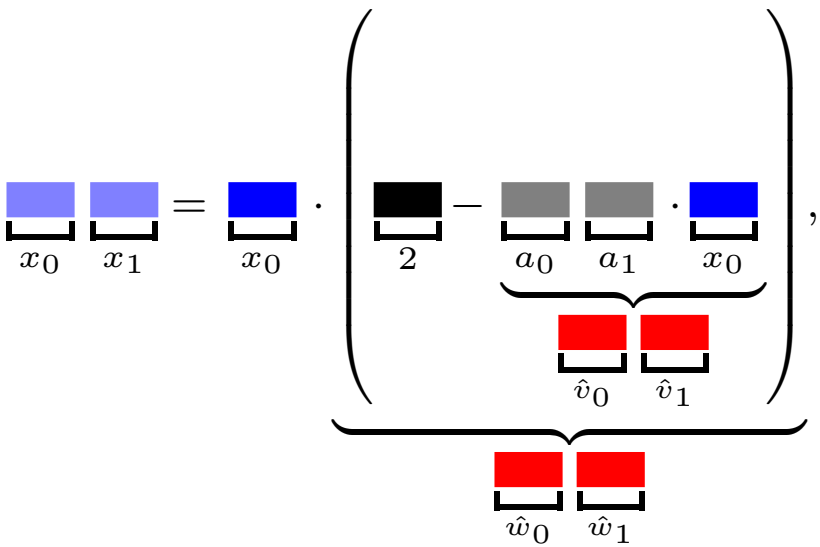
$$|v_i - \hat{v}_i| \leq \gamma_i \left|a^{(f_i)} \cdot x_i\right|, \tag{2c}$$

$$\left|a - a^{(f_i)}\right| \leq \gamma_i \left|a\right|. \tag{2d}$$

Truncations and Error Analysis

General procedure: $x_{n+1} = x_n(2 - ax_n)$

iter 0 

iter 1 

⋮

And finally,

$$\left|\frac{x_i - a^{-1}}{a^{-1}}\right| \leq 2^{-2^i(p-3)-1}$$

Error analysis based on triangular inequalities.

Let

$\eta = \sum_{j=0}^{\infty} 2^{(-j-1)p} = \frac{2^{-p}}{1-2^{-p}}$,

$\gamma_i = 2^{-(2^{i+1}-1)p} \frac{\eta}{1-\eta}$,

$$|x_{i+1} - \tau_i| \leq \gamma_i |x_i \cdot \hat{w}_i|, \tag{2a}$$

$$|w_i - \hat{w}_i| \leq \gamma_i |w_i| \leq \gamma_i |2 - \hat{v}_i|, \tag{2b}$$

$$|v_i - \hat{v}_i| \leq \gamma_i \left|a^{(f_i)} \cdot x_i\right|, \tag{2c}$$

$$\left|a - a^{(f_i)}\right| \leq \gamma_i |a|. \tag{2d}$$

Comparison and Results

Table: (a) Error bounds values for Priest's formula [Priest'91] vs. Daumas [Daumas] vs. our analysis (1); $d = 2^q$ terms are computed in the quotient

Prec, iteration	Eq. Priest	Eq. Daumas	Eq. (1)
$p = 53, q = 0$	2	2^{-49}	2^{-51}
$p = 53, q = 1$	1	2^{-98}	2^{-101}
$p = 53, q = 2$	2^{-2}	2^{-195}	2^{-201}
$p = 53, q = 3$	2^{-6}	2^{-387}	2^{-401}
$p = 53, q = 4$	2^{-13}	2^{-764}	2^{-801}
$p = 24, q = 0$	2	2^{-20}	2^{-22}
$p = 24, q = 1$	1	2^{-40}	2^{-43}
$p = 24, q = 2$	2^{-2}	2^{-79}	2^{-85}
$p = 24, q = 3$	2^{-5}	2^{-155}	2^{-169}
$p = 24, q = 4$	2^{-12}	2^{-300}	2^{-337}

Comparison and Results

Table: Timings[†] in MFlops/s for Alg. 1 vs. QD implementation for reciprocal (A) and division (B) of expansions; the numerator, denominator and quotient have respectively d_n, d_i and d_o terms.

d_i, d_o	Alg. 1	QD
$1, 1$	10^7	10^7
$2, 2$	62	70
$4, 4$	10	3.6
$1, 2$	62	86.2
$2, 4$	10.7	3.7
$4, 2$	61	86.2
$1, 4$	12.6	7.36
$1, 8$	2	*
$2, 8$	1.7	*
$4, 8$	1.4	*
$8, 8$	1.3	*
$1, 16$	0.3	*
$2, 16$	0.27	*
$4, 16$	0.22	*
$8, 16$	0.19	*
$16, 16$	0.17	*

(A) Reciprocal

d_n, d_i, d_o	Alg. 1	QD
$2, 2, 2$	46.3	70
$4, 4, 4$	6.8	3.6
$2, 1, 2$	46.7	86.2
$4, 2, 4$	7	3.7
$2, 4, 2$	46.1	86.2
$4, 1, 4$	7.7	7.36

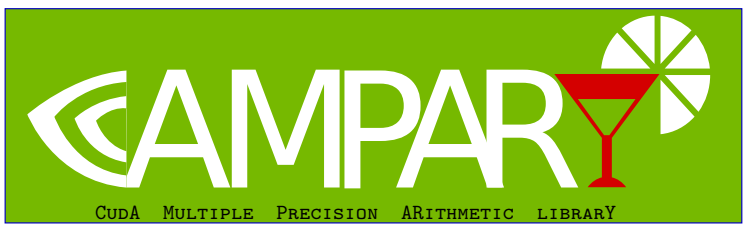
(B) Division

[†] Intel(R) Core(TM) i7 CPU 3820, 3.6GHz computer

* precision not supported

Conclusion

- Use multiple-term format for multiple-precision floating-point numbers → FP expansions
- Method for computing the reciprocal/division of FP expansions based on:
  - "truncated" addition and multiplication
  - "adapted" Newton-Raphson Iteration
- Thorough error analysis and explicit error bound



Thank you!