A new multiplication algorithm for extended precision using floating-point expansions

Valentina Popescu, Jean-Michel Muller, Ping Tak Peter Tang

RAIM 2016
28 June

CAMPARY
CudA Multiple Precision ARithmetic librarY

Target applications

1. Need massive parallel computations

   $\rightarrow$ high performance computing using graphics processors – GPUs

2. Need more precision than standard available (up to few hundred bits)

   $\rightarrow$ extend precision using floating-point expansions

Target applications

1. Need massive parallel computations

   → high performance computing using graphics processors – GPUs

2. Need more precision than standard available (up to few hundred bits)

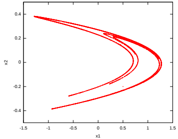
   → extend precision using floating-point expansions

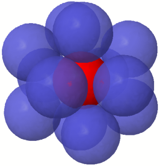
Chaotic dynamical systems:

- bifurcation analysis,
- compute periodic orbits (e.g., finding sinks in the Hénon map, iterating the Lorenz attractor),
- celestial mechanics (e.g., long term stability of the solar system).

Experimental mathematics: ill-posed SDP problems in

- computational geometry (e.g., computation of kissing numbers),
- quantum chemistry/information,
- polynomial optimization etc.





Extended precision

Existing libraries:

- GNU MPFR - not ported on GPU;
- GARPREC & CUMP - tuned for big array operations: data generated on host, operations on device;
- QD & GQD - limited to double-double and quad-double; no correct rounding.

Extended precision

Existing libraries:

- GNU MPFR - not ported on GPU;
- GARPREC & CUMP - tuned for big array operations: data generated on host, operations on device;
- QD & GQD - limited to double-double and quad-double; no correct rounding.

What we need:

- support for arbitrary precision;
- runs both on CPU and GPU;
- easy to use;

CAMPARY – CudaA Multiple Precision ARithmetic librarY –

Our approach: multiple-term representation
– floating-point expansions –

CAMPARY (CudaA Multiple Precision ARithmetic librarY)

Our approach: multiple-term representation
– floating-point expansions –

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

CAMPARY (CudaA Multiple Precision ARithmetic librarY)

Our approach: multiple-term representation
– floating-point expansions –

- Pros:
  – use directly available and highly optimized native FP infrastructure;
  – straightforwardly portable to highly parallel architectures, such as GPUs;
  – sufficiently simple and regular algorithms for addition.
- Cons:
  – more than one representation;
  – existing multiplication algorithms do not generalize well for an arbitrary number of terms;
  – difficult rigorous error analysis → lack of thorough error bounds.

Non-overlapping expansions

$R = 1.11010011e-1$ can be represented, using a $p = 5$ (in radix 2) system, as:

$R = x_0 + x_1 + x_2$:

$$\begin{cases} x_0 = 1.1000e-1; \\ x_1 = 1.0010e-3; \\ x_2 = 1.0110e-6. \end{cases}$$

Most compact $R = z_0 + z_1$:

$$\begin{cases} z_0 = 1.1101e-1; \\ z_1 = 1.1000e-8. \end{cases}$$

Less compact

$R = y_0 + y_1 + y_2 + y_3 + y_4 + y_5$:

$$\begin{cases} y_0 = 1.0000e-1; \\ y_1 = 1.0000e-2; \\ y_2 = 1.0000e-3; \\ y_3 = 1.0000e-5; \\ y_4 = 1.0000e-8; \\ y_5 = 1.0000e-9; \end{cases}$$

$R = 1.11010011e-1$ can be represented, using a $p = 5$ (in radix 2) system, as:

$R = x_0 + x_1 + x_2$:

$$\begin{cases} x_0 = 1.1000e-1; \\ x_1 = 1.0010e-3; \\ x_2 = 1.0110e-6. \end{cases}$$

Most compact $R = z_0 + z_1$:

$$\begin{cases} z_0 = 1.1101e-1; \\ z_1 = 1.1000e-8. \end{cases}$$

Less compact

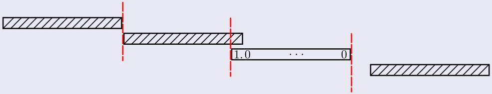
$R = y_0 + y_1 + y_2 + y_3 + y_4 + y_5$:

$$\begin{cases} y_0 = 1.0000e-1; \\ y_1 = 1.0000e-2; \\ y_2 = 1.0000e-3; \\ y_3 = 1.0000e-5; \\ y_4 = 1.0000e-8; \\ y_5 = 1.0000e-9; \end{cases}$$

Solution: the FP expansions are required to be *non-overlapping*.

Definition: *ulp-nonoverlapping*.

For an expansion $u_0, u_1, \ldots, u_{n-1}$ if for all $0 < i < n$, we have $|u_i| \leq \text{ulp}\,(u_{i-1})$.



Example: $p = 5$ (in radix 2)

$$\begin{cases} x_0 = 1.1010e-2;; \\ x_1 = 1.1101e-7; \\ x_2 = 1.0000e-11; \\ x_3 = 1.1000e-17. \end{cases}$$

Non-overlapping expansions

$R = 1.11010011e-1$ can be represented, using a $p = 5$ (in radix 2) system, as:

$R = x_0 + x_1 + x_2$:

$$\begin{cases} x_0 = 1.1000e-1; \\ x_1 = 1.0010e-3; \\ x_2 = 1.0110e-6. \end{cases}$$

Most compact $R = z_0 + z_1$:

$$\begin{cases} z_0 = 1.1101e-1; \\ z_1 = 1.1000e-8. \end{cases}$$

Less compact

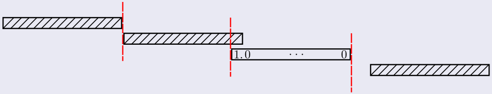
$R = y_0 + y_1 + y_2 + y_3 + y_4 + y_5$:

$$\begin{cases} y_0 = 1.0000e-1; \\ y_1 = 1.0000e-2; \\ y_2 = 1.0000e-3; \\ y_3 = 1.0000e-5; \\ y_4 = 1.0000e-8; \\ y_5 = 1.0000e-9; \end{cases}$$

Solution: the FP expansions are required to be *non-overlapping*.

Definition: *ulp-nonoverlapping*.

For an expansion $u_0, u_1, \ldots, u_{n-1}$ if for all $0 < i < n$, we have $|u_i| \leq \text{ulp}(u_{i-1})$.



Example: $p = 5$ (in radix 2)

$$\begin{cases} x_0 = 1.1010e-2;; \\ x_1 = 1.1101e-7; \\ x_2 = 1.0000e-11; \\ x_3 = 1.1000e-17. \end{cases}$$

Restriction: $n \leq 12$ for *single*-precision and $n \leq 39$ for *double*-precision.

Error-Free Transforms: *Fast2Sum* & *2MultFMA*

Algorithm 1 (*Fast2Sum* (a, b))

$s \leftarrow RN(a+b)$
$z \leftarrow RN(s-a)$
$e \leftarrow RN(b-z)$
return (s, e)

Requirement:

$$e_a \geq e_b;$$

$\rightarrow$ Uses 3 FP operations.

Algorithm 2 (*2MultFMA* (a, b))

$p \leftarrow RN(a \cdot b)$
$e \leftarrow fma(a, b, -p)$
return (p, e)

Requirement:

$$e_a + e_b \geq e_{min} + p - 1;$$

$\rightarrow$ Uses 2 FP operations.

Existing multiplication algorithms

1. Priest's multiplication [Pri91]:
   – very complex and costly;
   – based on scalar products;
   – uses re-normalization after each step;
   – computes the entire result and "truncates" a-posteriori;
   – comes with an error bound and correctness proof;

Existing multiplication algorithms

1. Priest's multiplication [Pri91]:
   – very complex and costly;
   – based on scalar products;
   – uses re-normalization after each step;
   – computes the entire result and "truncates" a-posteriori;
   – comes with an error bound and correctness proof;

2. quad-double multiplication in QD library:
   – does not straightforwardly generalize;
   – can lead to $\mathcal{O}(n^3)$ complexity;
   – worst case error bound is pessimistic;
   – no correctness proof is provided.

New multiplication algorithms

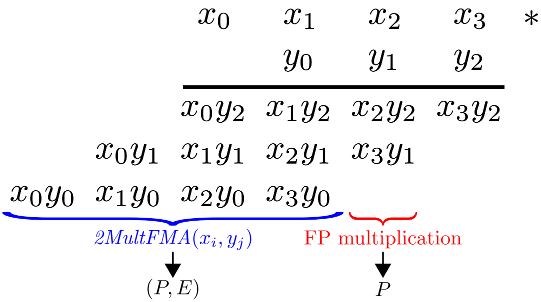
- requires: *ulp-nonoverlapping* FP expansion $x = (x_0, x_1, \ldots, x_{R-1})$ and $y = (y_0, y_1, \ldots, y_{R-1})$.
- ensures: *ulp-nonoverlapping* FP expansion $\pi = (\pi_0, \pi_1, \ldots, \pi_{R-1})$.

Let me explain it with an example ...

Example: $n = 4, m = 3$ and $r = 4$

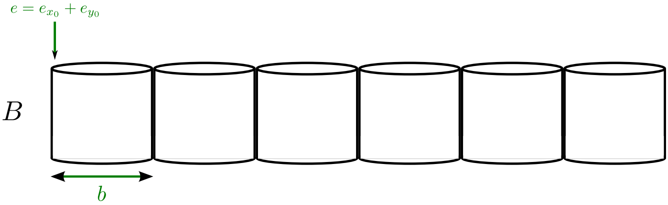
		x_0	x_1	x_2	x_3	$*$
			y_0	y_1	y_2	
		x_0y_2	x_1y_2	x_2y_2	x_3y_2	
	x_0y_1	x_1y_1	x_2y_1	x_3y_1		
x_0y_0	x_1y_0	x_2y_0	x_3y_0			

Example: $n = 4, m = 3$ and $r = 4$



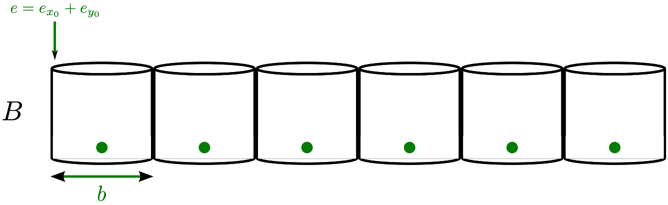
- paper-and-pencil intuition;
- term-times-expansion products, $x_i \cdot y$;
- on-the-fly "truncation";
- error correction term, π_r.

Example: $n = 4, m = 3$ and $r = 4$

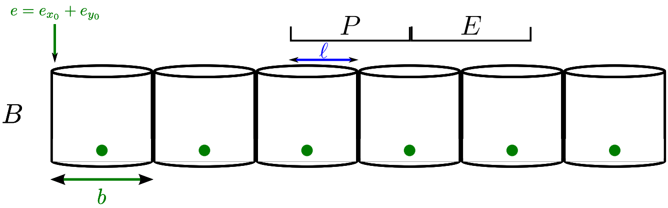


- $[\frac{r \cdot p}{b}] + 2$ containers of size b (s.t. $3b > 2p$);
- $b + c = p - 1$, s.t. we can add 2^c numbers without error; (*binary64* $\rightarrow b = 45$, *binary32* $\rightarrow b = 18$)
- starting exponent $e = e_{x_0} + e_{y_0}$;
- each bin's LSB has a fixed weight;

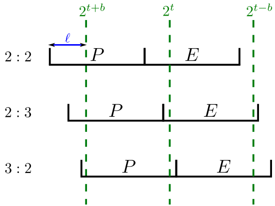
Example: $n = 4, m = 3$ and $r = 4$



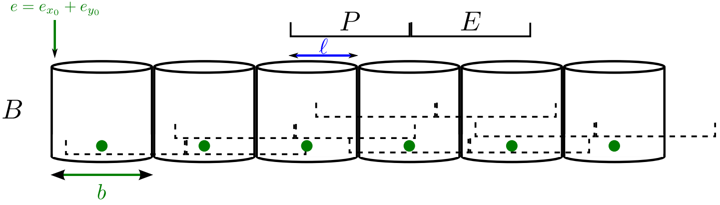
- $\lceil \frac{r \cdot p}{b} \rceil + 2$ containers of size b (s.t. $3b > 2p$);
- $b + c = p - 1$, s.t. we can add 2^c numbers without error; (*binary64* $\rightarrow b = 45$, *binary32* $\rightarrow b = 18$)
- starting exponent $e = e_{x_0} + e_{y_0}$;
- each bin's LSB has a fixed weight;
- bins initialized with $1.5 \cdot 2^{e-(i+1)b+p-1}$;



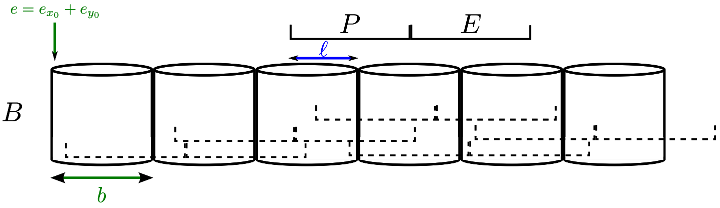
- $\lceil \frac{r \cdot p}{b} \rceil + 2$ containers of size b (s.t. $3b > 2p$);
- $b + c = p - 1$, s.t. we can add 2^c numbers without error; (*binary64* $\to b = 45$, *binary32* $\to b = 18$)
- starting exponent $e = e_{x_0} + e_{y_0}$;
- each bin's LSB has a fixed weight;
- bins initialized with $1.5 \cdot 2^{e-(i+1)b+p-1}$;
- the number of leading bits, ℓ;
- accumulation done using a *Fast2Sum* and addition [Rump09];



Example: $n = 4, m = 3$ and $r = 4$

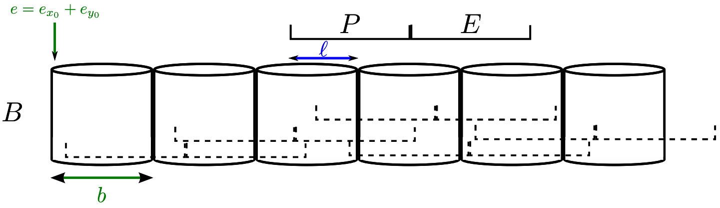


Example: $n = 4, m = 3$ and $r = 4$



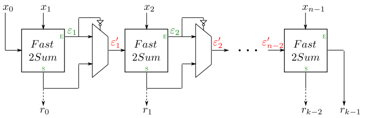
- subtract initial value;

Example: $n = 4, m = 3$ and $r = 4$



- subtract initial value;
- apply renormalization step to B:
  – *Fast2Sum* and branches;
  – render the result *ulp-nonoverlapping*;
- tight error bound:

$$|x_0 y_0| 2^{-(p-1)r} [1 + (r+1)2^{-p} + \\ + 2^{-(p-1)} \left(\frac{-2^{-(p-1)}}{(1-2^{-(p-1)})^2} + \frac{m+n-r-2}{1-2^{-(p-1)}} \right)]$$



Comparison

Table : Worst case FP operation count when the input and output expansions are of size r.

r	2	4	8	16
New algorithm	138	261	669	2103
Priest's mul.[Pri91]	3174	16212	87432	519312

Comparison

Table : Worst case FP operation count when the input and output expansions are of size r.

r	2	4	8	16
New algorithm	138	261	669	2103
Priest's mul.[Pri91]	3174	16212	87432	519312

Table : Performance in MFlops/s for multiplying two FP expansions on a Tesla K40c GPU, using CUDA 7.5 software architecture, running on a single thread of execution. $*$ precision not supported

d_x, d_y, d_r	New algorithm	QD
$2, 2, 2$	0.027	0.1043
$1, 2, 2$	0.365	0.1071
$3, 3, 3$	0.0149	$*$
$2, 3, 3$	0.0186	$*$
$4, 4, 4$	0.0103	0.0174
$1, 4, 4$	0.0215	0.0281
$2, 4, 4$	0.0142	$*$
$8, 8, 8$	0.0034	$*$
$4, 8, 8$	0.0048	$*$
$16, 16, 16$	0.001	$*$

Conclusions

CAMPARY
CudA Multiple Precision ARithmetic libraRY

Available online at: http://homepages.laas.fr/mmjoldes/campary/.

- algorithm with strong regularity;
- based on partial products accumulation;

A new multiplication algorithm for extended precision using floating-point expansions, joint work with J.-M. Muller and, P.Tang. To be presented at *IEEE 23rd Symposium on Computer Arithmetic*, ARITH 2016.

Available online at: http://homepages.laas.fr/mmjoldes/campary/.

- algorithm with strong regularity;
- based on partial products accumulation;
- uses a fixed-point structure that is floating-point friendly;
- thorough error analysis and tight error bound;

A new multiplication algorithm for extended precision using floating-point expansions, joint work with J.-M. Muller and, P.Tang. To be presented at *IEEE 23rd Symposium on Computer Arithmetic*, ARITH 2016.

Conclusions

CAMPARY
CudA Multiple Precision ARithmetic libraRY

Available online at: http://homepages.laas.fr/mmjoldes/campary/.

- algorithm with strong regularity;
- based on partial products accumulation;
- uses a fixed-point structure that is floating-point friendly;
- thorough error analysis and tight error bound;
- natural fit for GPUs;

A new multiplication algorithm for extended precision using floating-point expansions, joint work with J.-M. Muller and, P.Tang. To be presented at *IEEE 23rd Symposium on Computer Arithmetic*, ARITH 2016.

Conclusions

CudA Multiple Precision ARithmetic libraRY

Available online at: http://homepages.laas.fr/mmjoldes/campary/.

- algorithm with strong regularity;
- based on partial products accumulation;
- uses a fixed-point structure that is floating-point friendly;
- thorough error analysis and tight error bound;
- natural fit for GPUs;
- proved to be too complex for small precisions;
- performance gains with increased precision.

A new multiplication algorithm for extended precision using floating-point expansions, joint work with J.-M. Muller and, P.Tang. To be presented at *IEEE 23rd Symposium on Computer Arithmetic*, ARITH 2016.