CAMPARY
CudA Multiple Precision ARithmetic libraRY

Valentina Popescu

Joined work with:
Mioara Joldes, Jean-Michel Muller

March 2015
When do we need more precision?

**Youtube view counter:**

- 32-bit integer register (limit 2,147,483,647)

---

*courtesy of http://www.reddit.com/r/ProgrammerHumor/
When do we need more precision?

**Youtube view counter:**
- 32-bit integer register (limit 2,147,483,647)
- in 2014 the video *Psy - Gangnam Style* touched the limit

*statement: We never thought a video would be watched in numbers greater than a 32-bit integer... but that was before we met Psy.

When do we need more precision?

**Youtube view counter:**

- 32-bit integer register (limit 2,147,483,647)
- in 2014 the video *Psy - Gangnam Style* touched the limit
- update to 64-bit with a limit of 9,223,372,036,854,775,808 (more than 9 quintillion)

**YouTube statement:**

_We never thought a video would be watched in numbers greater than a 32-bit integer... but that was before we met Psy._

*courtesy of http://www.reddit.com/r/ProgrammerHumor/*
More serious things!

Dynamical systems:
- bifurcation analysis,
- compute periodic orbits (finding sinks in the Hénon map, iterating the Lorenz attractor),
- long term stability of the solar system.
More serious things!

Dynamical systems:
- bifurcation analysis,
- compute periodic orbits (finding sinks in the Hénon map, iterating the Lorenz attractor),
- long term stability of the solar system.

- Need more precision –few hundred bits– than standard available
- Need massive parallel computations:
  –high performance computing (HPC)–
A real number $X$ is approximated in a machine by a rational

$$x = M_x \cdot 2^{e_x - p + 1},$$

where

- $M_x$ is the *significand*, a $p$-digit signed integer in radix 2 s.t.
  $$2^{p-1} \leq |M_x| \leq 2^p - 1;$$
- $e_x$ is the *exponent*, a signed integer ($e_{\text{min}} \leq e_x \leq e_{\text{max}}$).
A real number $X$ is approximated in a machine by a rational

$$x = M_x \cdot 2^{e_x-p+1},$$

where

- $M_x$ is the significand, a $p$-digit signed integer in radix 2 s.t. $2^{p-1} \leq |M_x| \leq 2^p - 1$;
- $e_x$ is the exponent, a signed integer ($e_{min} \leq e_x \leq e_{max}$).

Concepts:

- **unit in the last place** (Goldberg’s definition):

  $$\text{ulp}(x) = 2^{e_x-p+1}.$$

- **unit in the last significant place**:

  $$\text{uls}(x) = \text{ulp}(x) \cdot 2^{z_x},$$

  where $z_x$ is the number of trailing zeros at the end of $M_x$. 
Reminder: IEEE 754-2008 standard

Most common formats

- Single precision format \((p = 24)\):
  
  \[
  \begin{array}{ccc}
  1 & 8 & 23 \\
  s & e & m \\
  \end{array}
  \]

- Double precision format \((p = 53)\):
  
  \[
  \begin{array}{ccc}
  1 & 11 & 52 \\
  s & e & m \\
  \end{array}
  \]

→ Implicit bit that is not stored.
**Reminder: IEEE 754-2008 standard**

### Most common formats

- **Single precision format** ($p = 24$):
  
  
  \[
  \begin{array}{ccc}
  1 & 8 & 23 \\
  s & e & m \\
  \end{array}
  \]

- **Double precision format** ($p = 53$):
  
  
  \[
  \begin{array}{ccc}
  1 & 11 & 52 \\
  s & e & m \\
  \end{array}
  \]

  → Implicit bit that is not stored.

### Rounding modes

- 4 rounding modes: RD, RU, RZ, RN
- Correct rounding for: $+, -, \times, \div, \sqrt{}$ (return what we would get by infinitely precise operations followed by rounding).
- Portability, determinism.
Multiple precision arithmetic libraries

Two ways of representing numbers in extended precision

- **multiple-digit representation** - a number is represented by a sequence of digits coupled with a single exponent (Ex. GNU MPFR, ARPREC, GARPREC, CUMP);

  \[ s \overline{M} e \]

- **multiple-term representation** - a number is expressed as the unevaluated sum of several FP numbers (also called a FP expansion) (Ex. QD, GQD).

  \[ u_0 \overline{u_1} \ldots u_{n-1} \]
Multiple precision arithmetic libraries

Two ways of representing numbers in extended precision

- **multiple-digit representation** - a number is represented by a sequence of digits coupled with a single exponent (Ex. GNU MPFR, ARPREC, GARPREC, CUMP);

\[ s \overline{M} e \]

- **multiple-term representation** - a number is expressed as the unevaluated sum of several FP numbers (also called a FP expansion) (Ex. QD, GQD).

\[ u_0 \overline{u_1} \ldots \overline{u_{n-1}} \]

Need for another multiple precision library:

- GNU MPFR - not ported on GPU
- GARPREC & CUMP - tuned for big array operations where the data is generated on the host and only the operations are performed on the device
- QD & GQD - offer only double-double and quad-double precision; the results are not correctly rounded
Our approach: multiple-term representation

Drawback: redundant concept, more than one representation.
Our approach: multiple-term representation

Drawback: redundant concept, more than one representation.

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

The real number \( R = 1.11010011e - 1 \) can be represented as:

\[
R = x_0 + x_1 + x_2:
\begin{align*}
x_0 &= 1.1000e - 1; \\
x_1 &= 1.0010e - 3; \\
x_2 &= 1.0110e - 6.
\end{align*}
\]
Our approach: multiple-term representation

Drawback: redundant concept, more than one representation.

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

The real number \( R = 1.11010011e - 1 \) can be represented as:

\[
R = x_0 + x_1 + x_2:
\begin{align*}
x_0 &= 1.1000e - 1; \\
x_1 &= 1.0010e - 3; \\
x_2 &= 1.0110e - 6.
\end{align*}
\]

Most compact \( R = z_0 + z_1:\)

\[
\begin{align*}
z_0 &= 1.1101e - 1; \\
z_1 &= 1.1000e - 8.
\end{align*}
\]
Our approach: multiple-term representation

Drawback: redundant concept, more than one representation.

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

The real number $R = 1.11010011e − 1$ can be represented as:

<table>
<thead>
<tr>
<th>Most compact</th>
<th>$R = z_0 + z_1$:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$z_0 = 1.1101e − 1$;</td>
</tr>
<tr>
<td></td>
<td>$z_1 = 1.1000e − 8$.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>$y_0 = 1.0000e − 1$;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$y_1 = 1.0000e − 2$;</td>
</tr>
<tr>
<td></td>
<td>$y_2 = 1.0000e − 3$;</td>
</tr>
<tr>
<td></td>
<td>$y_3 = 1.0000e − 5$;</td>
</tr>
<tr>
<td></td>
<td>$y_4 = 1.0000e − 8$;</td>
</tr>
<tr>
<td></td>
<td>$y_5 = 1.0000e − 9$.</td>
</tr>
</tbody>
</table>
CAMPARY (CudaA Multiple Precision ARithmetic librarY)

Our approach: multiple-term representation

Drawback: redundant concept, more than one representation.

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

The real number \( R = 1.11010011e - 1 \) can be represented as:

\[
R = x_0 + x_1 + x_2: \\
x_0 = 1.1000e - 1; \\
x_1 = 1.0010e - 3; \\
x_2 = 1.0110e - 6.
\]

Most compact \( R = z_0 + z_1: \)

\[
\begin{align*}
 z_0 &= 1.1101e - 1; \\
 z_1 &= 1.1000e - 8.
\end{align*}
\]

Least compact

\[
R = y_0 + y_1 + y_2 + y_3 + y_4 + y_5: \\
 y_0 = 1.0000e - 1; \\
 y_1 = 1.0000e - 2; \\
 y_2 = 1.0000e - 3; \\
 y_3 = 1.0000e - 5; \\
 y_4 = 1.0000e - 8; \\
 y_5 = 1.0000e - 9;
\]

Solution

To ensure that an expansion carries significantly more information than one FP number only, it is required to be non-overlapping.

\[\rightarrow (re-)\text{normalization algorithms}\]
**Nonoverlapping expansions**

**Definition 1:** $\mathcal{P}$-nonoverlapping (according to Priest's definition).

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

Example:

\[
\begin{align*}
    x_0 &= 1.1010e - 2; \\
    x_1 &= 1.1101e - 7; \\
    x_2 &= 1.0100e - 12; \\
    x_3 &= 1.1000e - 18.
\end{align*}
\]
**Nonoverlapping expansions**

**Definition 1:** \(P\)-nonoverlapping (according to Priest's definition).

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

Example:

\[
\begin{align*}
x_0 &= 1.1010e-2; \\
x_1 &= 1.1101e-7; \\
x_2 &= 1.0100e-12; \\
x_3 &= 1.1000e-18.
\end{align*}
\]

**Definition 2 (nonzero-overlapping):** \(S\)-nonoverlapping (according to Shewchuk’s definition).

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

Example:

\[
\begin{align*}
\begin{array}{c}
\text{Example:} \\
x_0 &= 1.1000e-1; \\
x_1 &= 1.0100e-3; \\
x_2 &= 1.1001e-7; \\
x_3 &= 1.1010e-12;
\end{array}
\end{align*}
\]
Nonoverlapping expansions

**Definition 1 (P-nonoverlapping):** According to Priest's definition.

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

**Example:**

\[
\begin{align*}
x_0 &= 1.1010e - 2; \\
x_1 &= 1.1101e - 7; \\
x_2 &= 1.0100e - 12; \\
x_3 &= 1.1000e - 18.
\end{align*}
\]

**Definition 2 (nonzero-overlapping):** According to Shewchuk's definition.

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

**Example:**

\[
\begin{align*}
x_0 &= 1.1000e - 1; \\
x_1 &= 1.0100e - 3; \\
x_2 &= 1.1001e - 7; \\
x_3 &= 1.1010e - 12;
\end{align*}
\]
Theorem 1 (2Sum algorithm)

Let $a$ and $b$ be FP numbers. Algorithm 2Sum computes two FP numbers $s$ and $e$ that satisfy the following:

- $s + e = a + b$ exactly;
- $s = RN(a + b)$.

($RN$ stands for performing the operation in rounding to nearest rounding mode.)

Algorithm 1 (2Sum $(a, b)$)

\[
\begin{align*}
s & \leftarrow RN(a + b) \\
t & \leftarrow RN(s - b) \\
e & \leftarrow RN(RN(a - t) + RN(b - RN(s - t))) \\
\text{return} & \ (s, e)
\end{align*}
\]

$\rightarrow$ 6 FP operations (proved to be optimal unless we have information on the ordering of $|a|$ and $|b|$)
Theorem 2 (Fast2Sum algorithm)

Let $a$ and $b$ be FP numbers that satisfy $|a| \geq |b|$. Algorithm Fast2Sum computes two FP numbers $s$ and $e$ that satisfy the following:

- $s + e = a + b$ exactly;
- $s = \text{RN}(a + b)$.

Algorithm 2 (Fast2Sum ($a, b$))

\[
\begin{align*}
  s & \leftarrow \text{RN}(a + b) \\
  z & \leftarrow \text{RN}(s - a) \\
  e & \leftarrow \text{RN}(b - z) \\
  \text{return} & \quad (s, e)
\end{align*}
\]

\[\rightarrow 3 \text{ FP operations}\]
Error-Free Transforms: 2Sum & 2ProdFMA

Theorem 3 (2ProdFMA algorithm)

Let \( a \) and \( b \) be FP numbers, \( e_a + e_b \geq e_{\text{min}} + p - 1 \). Algorithm 2ProdFMA computes two FP numbers \( p \) and \( e \) that satisfy the following:

- \( p + e = a \cdot b \) exactly;
- \( p = \text{RN}(a \cdot b) \).

Algorithm 3 (2ProdFMA \((a, b)\))

\[
\begin{align*}
p & \leftarrow \text{RN}(a \cdot b) \\
e & \leftarrow \text{fma}(a, b, -p) \\
\text{return} & \ (p, e)
\end{align*}
\]
Distillation Algorithms: VecSum

Algorithm 4 (VecSum \((x_0, \ldots, x_{n-1})\))

\[
\text{for } i \leftarrow n - 1 \text{ to } 1 \text{ do} \\
\quad (s_{i-1}, e_i) \leftarrow 2\text{Sum}(x_i, x_{i-1}) \\
\text{end for} \\
\quad e_0 \leftarrow s_0 \\
\text{return } e_0, \ldots, e_{n-1}
\]
Algorithm 4 (VecSum ($x_0, \ldots, x_{n-1}$))

\[
\text{for } i \leftarrow n - 1 \text{ to } 1 \text{ do} \\
\quad (s_{i-1}, e_i) \leftarrow 2\text{Sum}(x_i, x_{i-1}) \\
\text{end for} \\
\]
\[e_0 \leftarrow s_0\]
\[\text{return } e_0, \ldots, e_{n-1}\]

Recently proven property:

If $x_0, \ldots, x_{n-1}$ overlap by at most $d \leq p - 1$ digits, then the sequence $e_0, \ldots, e_{n-1}$ is $S$-non-overlapping.

Restriction: $n \leq 12$ for single precision and $n \leq 39$ for double precision.
Renormalization algorithms

Priest's renormalization algorithm [Priest'91]

Schematic drawing for $n = 5$.

Drawbacks: many conditional branches → no pipelined operations → slow in practice.
Priest’s renormalization algorithm [Priest’91]

Schematic drawing for $n = 5$.

Drawbacks: many conditional branches $\rightarrow$ no pipelined operations $\rightarrow$ slow in practice
New renormalization algorithm

Based on chained levels of $2\text{Sum}$ and $\text{Fast2Sum}$. 
Input:
- $x_0, \ldots, x_{n-1}$, FP numbers that satisfy one of the following cases:
  1. overlap by at most $d \leq p - 1$ digits;
  2. overlap by at most $d \leq p - 2$ digits and may contain pairs of at most 2 consecutive terms that overlap by $p$ digits;
Remark: in both cases we allow interleaving 0s;
- $m$, input parameter, with $1 \leq m \leq n - 1$;

Output: "truncation" to $m$ terms of a $P$-nonoverlapping FP expansion $f = f_0, \ldots, f_{n-1}$ such that $x_0 + \ldots + x_{n-1} = f$ and
$$f_{i+1} \leq \left( \frac{1}{2} + 2^{-p+2} + 2^{-p} \right) \text{ulp}(f_i), \text{ for all } 0 \leq i < m - 1.$$
Input: \( x_0, \ldots, x_{n-1} \), FP numbers that overlap by at most \( d \leq p - 2 \) digits and can contain pairs of at most 2 consecutive terms that overlap by \( p \) digits and may also contain interleaving 0s;

Output: \( e = (e_0, \ldots, e_{n-1}) \) that satisfies:
\[ |e_0| > |e_1| \geq \ldots \geq |e_{i-1}| \geq |e_i| > |e_{i+1}| \geq |e_{i+2}| > \ldots, \]
where:
- \( |e_i| > |e_{i+1}| \) implies they are \( S\)-nonoverlapping;
- \( |e_i| \geq |e_{i+1}| \) implies they are \( S\)-nonoverlapping for strict inequality or they are equal to a power of 2.
Input:
- \( e = (e_0, \ldots, e_{n-1}) \) that satisfies:
  - \(|e_0| > |e_1| \geq \ldots \geq |e_{i-1}| \geq |e_i| > |e_{i+1}| \geq |e_{i+2}| > \ldots\), where:
    - \(|e_i| > |e_{i+1}|\) implies they are \( S \)-nonoverlapping;
    - \(|e_i| \geq |e_{i+1}|\) implies they are \( S \)-nonoverlapping for strict inequality or they are equal to a power of 2
- \( m \), with \( 1 \leq m \leq n \) the required number of output terms;

Output: \( f = (f_0, \ldots, f_{m-1}) \), with \( 0 \leq m \leq n - 1 \) satisfies \(|f_{i+1}| \leq \text{ulp}(f_i)\) for all \( 0 \leq i < m - 1 \).
Input: $f = (f_0, \ldots, f_m)$, with $|f_{i+1}| \leq \text{ulp}(f_i)$, for all $0 \leq i \leq m - 1$;

Output: $g = (g_0, \ldots, g_m)$ satisfies $|g_1| \leq (\frac{1}{2} + 2^{-p+2}) \text{ulp}(g_0)$ and $|g_{i+1}| \leq \text{ulp}(g_i)$, for $0 < i \leq m - 1$.

Remark: we can use $\text{Fast2Sum}(\rho_i, f_{i+1})$, because we either have $\rho_i = 0$ or $|f_{i+1}| \leq |\rho_i|$.
After third level:

- \(\mathcal{P}\)-nonoverlapping condition for the first two elements of \(g\);
- the rest of \(g\) keeps the existing bound, \(|g_{i+1}| \leq \text{ulp}(g_i)\) for all \(0 < i \leq m - 1\).

**Advantage:** if zeros appear, they are pushed at the end.

**Solution:** continue applying \(m - 1\) subsequent levels of \(\text{VecSumErr}\) on the remaining elements that overlap.
Example

$p = 5; n = 6; m = 3.$
Operation count

- $R_{Priest}(n) = 20(n - 1)$ FP operations;
- $R_{new}(n, m) = 13n + \frac{3}{2}m^2 + \frac{3}{2}m - 22$ FP operations.

Table: FP operation count for the new renormalization algorithm vs. Priest’s one [Priest’91]. We consider that both algorithms compute $n - 1$ terms in the output expansion.

<table>
<thead>
<tr>
<th>$n$</th>
<th>2</th>
<th>4</th>
<th>7</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>New algorithm</td>
<td>13</td>
<td>60</td>
<td>153</td>
<td>190</td>
<td>273</td>
<td>368</td>
<td>594</td>
</tr>
<tr>
<td>Priest’s algorithm</td>
<td>20</td>
<td>60</td>
<td>120</td>
<td>140</td>
<td>180</td>
<td>220</td>
<td>300</td>
</tr>
</tbody>
</table>

Observe that for $n > 4$ Priest’s algorithm performs better. But, in practice, the last $m - 1$ levels will take advantage of the computers pipeline —→ our algorithm performs better.
Conclusions

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

- Use multiple-term format for FP multiple-precision numbers → FP expansions;

---


Conclusions

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

- Use multiple-term format for FP multiple-precision numbers → FP expansions;
- Implementation: templated class in CUDA C;


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

- Use multiple-term format for FP multiple-precision numbers → FP expansions;
- Implementation: templated class in CUDA C;
- Support for addition, multiplication, reciprocal/division and square root of FP expansions on both CPU and GPU;

*On the computation of the reciprocal of floating point expansions using an adapted Newton-Raphson iteration, joint work with M. Joldes and, J.-M. Muller. In IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2014;*

Conclusions

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

- Use multiple-term format for FP multiple-precision numbers → FP expansions;
- Implementation: templated class in CUDA C;
- Support for addition, multiplication, reciprocal/division and square root of FP expansions on both CPU and GPU;
- Thorough error analysis and explicit error bounds;

* On the computation of the reciprocal of floating point expansions using an adapted Newton-Raphson iteration, joint work with M. Joldes and, J.-M. Muller. In IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2014;
Conclusions

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

- Use multiple-term format for FP multiple-precision numbers → FP expansions;
- Implementation: templated class in CUDA C;
- Support for addition, multiplication, reciprocal/division and square root of FP expansions on both CPU and GPU;
- Thorough error analysis and explicit error bounds;
- Renormalization algorithm that guarantees to render the result nonoverlapping.

* On the computation of the reciprocal of floating point expansions using an adapted Newton-Raphson iteration, joint work with M. Joldes and, J.-M. Muller. In IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2014;

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

- Use multiple-term format for FP multiple-precision numbers → FP expansions;
- Implementation: templated class in CUDA C;
- Support for addition, multiplication, reciprocal/division and square root of FP expansions on both CPU and GPU;
- Thorough error analysis and explicit error bounds;
- Renormalization algorithm that guarantees to render the result nonoverlapping.

**On going work**

- Provide error analysis for addition and multiplication algorithms.


Main algorithm for renormalization of FP expansion

Algorithm 5 (Renormalization algorithm.)

Require: FP expansion $x = x_0 + \ldots + x_{n-1}$ consisting of FP numbers that overlap by at most $d$ digits, with $d \leq p - 1$ or $d \leq p - 2$ and may contain pairs of at most 2 consecutive terms that overlap by $p$ digits; $m$ length of output FP expansion.

Ensure: FP expansion $f = f_0 + \ldots + f_{m-1}$ s.t.

$$f_{i+1} \leq \left( \frac{1}{2} + 2^{-p+2} + 2^{-p} \right) \text{ulp}(f_i), \text{ for all } 0 \leq i < m - 1. \quad (1)$$

1: $e[0 : n - 1] \leftarrow \text{VecSum}(x[0 : n - 1])$
2: $f^{(0)}[0 : m] \leftarrow \text{VecSumErrBranch}(e[0 : n - 1], m + 1)$
3: **for** $i \leftarrow 0$ **to** $m - 2$ **do**
4: \hspace{1em} $f^{(i+1)}[i : m] \leftarrow \text{VecSumErr}(f^{(i)}[i : m])$
5: **end for**
6: **return** FP expansion $f = f^{(1)}_0 + \ldots + f^{(m-1)}_{m-2} + f^{(m-1)}_{m-1}$. 