Range Reduction Based on Pythagorean Triples for Trigonometric Function Evaluation
IEEE ASAP 2015 Conference

Hugues de Lassus Saint-Geniès  David Defour  Guillaume Revy

DALI, Université de Perpignan Via Domitia
LIRMM, Université de Montpellier
CNRS UMR 5506

July 27th, 2015
Context and objectives

- IEEE-754 floating-point arithmetic
Context, objectives and achievements

Context and objectives

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
    - exp, log, expm1
    - sin, cos, arctan
    - $\left(1 + x\right)^n$, $x^y$, ...
Context, objectives and achievements

**Context and objectives**

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding

\[ f(x) \]

machine numbers

H. de Lassus Saint-Geniès
Range Reduction Based on Pythagorean Triples for Trigonometric Function Evaluation
Context, objectives and achievements

Context and objectives

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding

- Trigonometric functions

\[
(1 + x)^n \quad x^y \quad \ldots
\]
Context and objectives

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding
- Trigonometric functions
- Table-based range reduction
  - Remove sources of error

Trigonometric functions:
- $\sin$, $\cos$, $\arctan$
- $\exp$, $\log$, $\expm1$
- $\exp(x)$, $\log(x)$, $\expm1(x)$
- $\sin(x)$, $\cos(x)$, $\arctan(x)$
- $(1 + x)^n$, $x^y$, ...
Context and objectives

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding
- Trigonometric functions
- Table-based range reduction
  - Remove sources of error

Achievements

- Algorithm to tabulate exact values for the sine and cosine functions
Context, objectives and achievements

Context and objectives

- IEEE-754 floating-point arithmetic
  - Elementary function evaluation
  - Correct rounding
- Trigonometric functions
- Table-based range reduction
  - Remove sources of error

Achievements

- Algorithm to tabulate exact values for the sine and cosine functions
- For correct-rounding in double precision:
  - Precomputed table size reduced up to 45%
  - Memory accesses and FLOPs reduced up to 45% during the reconstruction
1 Background on sine and cosine evaluation
   - Additive reduction
   - Table-lookup
   - Sources of error

2 Error-free table method for sine and cosine implementations
   - Concentration of the error
   - Tabulating rational numbers
   - Pythagorean triples for trigonometric functions

3 Experimental results
   - Exhaustive search
   - Heuristic search
   - Comparisons with other table-based methods

4 Conclusion and perspectives
1 Background on sine and cosine evaluation
   - Additive reduction
   - Table-lookup
   - Sources of error

2 Error-free table method for sine and cosine implementations
   - Concentration of the error
   - Tabulating rational numbers
   - Pythagorean triples for trigonometric functions

3 Experimental results
   - Exhaustive search
   - Heuristic search
   - Comparisons with other table-based methods

4 Conclusion and perspectives
Additive range reduction

\[ x \in \mathbb{F} \quad \xrightarrow{\text{additive reduction}} \quad \xrightarrow{\text{table-lookup}} \quad \xrightarrow{\text{poly. eval.}} \quad \xrightarrow{\text{reconstruction}} \]

- Use of trigonometric **symmetries and periodicities**

![Graph of \( \sin(x) \)]
Additive range reduction

\[ x \in \mathbb{F} \xrightarrow{\text{additive reduction}} \text{table-lookup} \xrightarrow{\text{poly. eval.}} \text{reconstruction} \]

- Use of trigonometric symmetries and periodicities

\[ \sin(x) = \sin(x + 2\pi) \]
Additive range reduction

\[ x \in \mathbb{F} \]

\[ \text{range reduction} \rightarrow \text{additive reduction} \rightarrow \text{table-lookup} \rightarrow \text{poly. eval.} \rightarrow \text{reconstruction} \]

- Use of trigonometric **symmetries and periodicities**

\[ \sin(-x) = -\sin(x) \]
Additive range reduction

\[ x \in \mathbb{F} \rightarrow \text{additive reduction} \rightarrow \text{table-lookup} \rightarrow \text{poly. eval.} \rightarrow \text{reconstruction} \]

- Use of trigonometric symmetries and periodicities

\[ \sin(x) = \pm f_k \left( x - k \cdot \frac{\pi}{2} \right) \quad \text{with} \quad f_k \in \{\sin, \cos\} \]
Additive range reduction

Use of trigonometric symmetries and periodicities

\[ \sin(x) = \pm f_k (x - k \cdot \frac{\pi}{2}) \quad \text{with} \quad f_k \in \{\sin, \cos\} \]
Additive range reduction

\[ \begin{array}{c}
\mathbf{x} \in F \\
\text{range reduction}
\end{array} \xrightarrow{\text{additive reduction}} \xrightarrow{\text{table-lookup}} \xrightarrow{\text{poly. eval.}} \xrightarrow{\text{reconstruction}} \]

- Use of trigonometric symmetries and periodicities

\[ \Rightarrow \text{Range reduction } \mathbf{x} \in F \mapsto \mathbf{x}^* \in [0, \pi/4] \]
Tang’s table-based range reduction

\[ x \in \mathbb{F} \]

range reduction

additive reduction → table-lookup → poly. eval. → reconstruction

\[
\begin{align*}
\cos h & \approx \cos (x h) \\
\sin h & \approx \sin (x h) \\
\end{align*}
\]

→ regularly spaced angles

\[
x h (i) = i \cdot 2^{-p}
\]

\[
x^* = x \cdot 2^{-p}
\]

\[
\sin (x^*) \approx \sin (h \otimes \cos (x^*) \oplus \cos (h \otimes \sin (x^*)))
\]
Tang’s table-based range reduction

$\mathbf{x} \in \mathbb{F} \rightarrow \text{additive reduction} \rightarrow \text{table-lookup} \rightarrow \text{poly. eval.} \rightarrow \text{reconstruction}$

$\rightarrow$ regularly spaced angles

$x_h(i) = i \cdot 2^{-p}$
Tang’s table-based range reduction

- \( x \in \mathbb{F} \)
- additive reduction → table-lookup → poly. eval. → reconstruction

\[ \rightarrow \text{regularly spaced angles} \]
\[ x_h(i) = i \cdot 2^{-p} \]
Tang’s table-based range reduction

\[ x \in \mathbb{F} \xrightarrow{\text{additive reduction}} \text{table-lookup} \xrightarrow{\text{poly. eval.}} \text{reconstruction} \]

\[ x \ast \rightarrow \text{regularly spaced angles} \]

\[ x_h(i) = i \cdot 2^{-p} \]

\[ x^* = \underbrace{x.xx \cdots xx}_{p \text{ bits}} \underbrace{xx \cdots xx}_{p \text{ bits}} \]
Tang’s table-based range reduction

$x \in \mathbb{F}$

range reduction

additive reduction

table-lookup

poly. eval.

reconstruction

$\cos h \approx \cos(x_h)$

$\sin h \approx \sin(x_h)$

$\rightarrow$ regularly spaced angles

$x_h(i) = i \cdot 2^{-p}$

$x^* = \overbrace{XX \cdots XX}^{p \text{ bits: } x_h} \overbrace{XX \cdots XX}^{XX \cdots XX}$
Tang’s table-based range reduction

$x \in \mathbb{F}$

range reduction

additive reduction

table-lookup

poly. eval.

reconstruction

$\sin_h \approx \sin(x_h)$

$\cos_h \approx \cos(x_h)$

$\rightarrow$ regularly spaced angles

$x_h(i) = i \cdot 2^{-p}$

$x^* = \underbrace{x.x \cdot \cdot \cdot x}_{p \text{ bits: } x_h} \underbrace{x \cdot \cdot \cdot x}_{x_\ell}$
Tang’s table-based range reduction

\[ x \in \mathbb{F} \]

range reduction

additive reduction

\[ x^* \rightarrow \text{regularly spaced angles} \]

\[ x_h(i) = i \cdot 2^{-p} \]

\[ x^* = \underbrace{x_{.XX \cdots XX}}_{p \text{ bits: } x_h} \underbrace{XX \cdots XX}_{x_{\ell}} \]

\[ \sin(x^*) \approx \sin_h \odot \cos(x_{\ell}) \oplus \cos_h \odot \sin(x_{\ell}) \]
Sources of error in the classical method

Errors appear in $x \in \mathbb{F}$, because of the additive range reduction in $\sin h$ and $\cos h$, which are rounded approximations in $\sin(x^\ell)$ and $\cos(x^\ell)$, computed by polynomial evaluations during the reconstruction process.

Drawback: additional bits in computation steps for accuracy purposes.
Sources of error in the classical method

Errors appear:

- in $x^*$, because of the additive range reduction
Sources of error in the classical method

Errors appear:

- in $x^*$, because of the additive range reduction
- in $\sin_h$ and $\cos_h$, which are rounded approximations
Sources of error in the classical method

Errors appear:

- in $x^*$, because of the additive range reduction
- in $\sin_h$ and $\cos_h$, which are rounded approximations
- in $\sin(x_\ell)$ and $\cos(x_\ell)$, computed by polynomial evaluations
Errors appear:

- in $x^*$, because of the additive range reduction
- in $\sin_h$ and $\cos_h$, which are rounded approximations
- in $\sin(x_\ell)$ and $\cos(x_\ell)$, computed by polynomial evaluations
- during the reconstruction process
Sources of error in the classical method

Errors appear:

- in $x^*$, because of the **additive range reduction**
- in $\sin_h$ and $\cos_h$, which are **rounded approximations**
- in $\sin(x_\ell)$ and $\cos(x_\ell)$, computed by polynomial evaluations
- during the **reconstruction** process

**Drawback:** **additional bits** in computation steps for accuracy purposes
An improvement: Gal’s accurate tables

Small perturbations to the table inputs → \textit{corr}

\( \sin_h \) and \( \cos_h \) 10 to 21 bits closer to machine numbers

\[
\begin{align*}
\sin(\times_h) \approx \sin_h \otimes \cos(\times_{\ell}) \\
\oplus \cos_h \otimes \sin(\times_{\ell})
\end{align*}
\]
An improvement: Gal’s accurate tables

- Small perturbations to the table inputs → \( corr \)
- \( \sin_h \) and \( \cos_h \) 10 to 21 bits closer to machine numbers

\[
\sin(x_h) \approx \sin(x_h + corr)
\]
\[
\cos(x_h) \approx \cos(x_h + corr)
\]

\[
\sin(x^*) \approx \sin_h \otimes \cos(x_\ell - corr) \oplus \cos_h \otimes \sin(x_\ell - corr)
\]
1 Background on sine and cosine evaluation
   ● Additive reduction
   ● Table-lookup
   ● Sources of error

2 Error-free table method for sine and cosine implementations
   ● Concentration of the error
   ● Tabulating rational numbers
   ● Pythagorean triples for trigonometric functions

3 Experimental results
   ● Exhaustive search
   ● Heuristic search
   ● Comparisons with other table-based methods

4 Conclusion and perspectives
Key idea: remove the error on tabulated values

\[ x \in \mathbb{F} \rightarrow \text{range reduction} \rightarrow \text{additive reduction} \rightarrow \text{table-lookup} \rightarrow \text{poly. eval.} \rightarrow \text{reconstruction} \]

Our conception of how error must be treated:
Key idea: remove the error on tabulated values

Our conception of **how error must be treated:**

*remove it whenever you can!*

- tables should store **exact values** on machine numbers
- it saves bits
- it **concentrates the error** on subsequent steps
- it makes the reconstruction step **easier**
What if we had rational numbers to tabulate?

\[ \sin(x) \approx (a \otimes \cos(x_\ell) \oplus b \otimes \sin(x_\ell)) \ominus \cosh \]
We’ve been having them since Euclid (≈ 300 BC)
We’ve been having them since Euclid ($\approx 300$ BC)

Each dot represents a primitive Pythagorean triple (PPT):

$$\exists (a, b, c) \in \mathbb{N}^3 \text{ coprime} \mid a^2 + b^2 = c^2$$

$$\Rightarrow \exists x \in \left[0, \frac{\pi}{2}\right] \mid \sin(x) = \frac{a}{c} \text{ and } \cos(x) = \frac{b}{c}$$
Primitive Pythagorean triples (PPT) generation

One easy way: ternary-trees with root \((3, 4, 5)\)

- 3 linear relationships \(\rightarrow\) 3 children/node, e.g:

\[
\begin{pmatrix}
1 & -2 & 2 \\
2 & -1 & 2 \\
2 & -2 & 3
\end{pmatrix},
\begin{pmatrix}
-1 & 2 & 2 \\
2 & -1 & 2 \\
-2 & 2 & 3
\end{pmatrix},
\begin{pmatrix}
1 & 2 & 2 \\
2 & 1 & 2 \\
2 & 2 & 3
\end{pmatrix}
\]
One easy way: ternary-trees with root (3, 4, 5)

- 3 linear relationships → 3 children/node, e.g:

\[
\begin{pmatrix}
1 & -2 & 2 \\
2 & -1 & 2 \\
2 & -2 & 3
\end{pmatrix}
\quad,
\begin{pmatrix}
-1 & 2 & 2 \\
2 & -1 & 2 \\
-2 & 2 & 3
\end{pmatrix}
\quad,
\begin{pmatrix}
1 & 2 & 2 \\
2 & 1 & 2 \\
2 & 2 & 3
\end{pmatrix}
\]
Primitive Pythagorean triples with $c \leq 2^{12}$
Primitive Pythagorean triples with $c \leq 2^{12}$
Primitive Pythagorean triples with $c \leq 2^{12}$

H. de Lassus Saint-Geniès

Range Reduction Based on Pythagorean Triples for Trigonometric Function Evaluation
What are “good” Pythagorean triples?

\[
\sin(x) \approx (a_i \otimes \cos(x_\ell - corr) \oplus b_i \otimes \sin(x_\ell - corr)) \otimes c_i
\]
What are “good” Pythagorean triples?

\[
\sin(x) \approx \left( \frac{a_i \cdot k}{c_i} \otimes \cos(x_\ell - corr) \oplus \frac{b_i \cdot k}{c_i} \otimes \sin(x_\ell - corr) \right) \otimes k
\]
What are “good” Pythagorean triples?

\[ \sin(x) \approx A_i \otimes \left[ \cos(x_\ell - corr)/k \right] \oplus B_i \otimes \left[ \sin(x_\ell - corr)/k \right] \]
For each table entry $i$:

- store integers $A_i = \frac{a_i}{c_i} \cdot k$ and $B_i = \frac{b_i}{c_i} \cdot k$ ($A_i, B_i \in \mathbb{F}$)
Actual use of Pythagorean triples

- For each table entry $i$:
  - store integers $A_i = \frac{a_i}{c_i} \cdot k$ and $B_i = \frac{b_i}{c_i} \cdot k \quad (A_i, B_i \in \mathbb{F})$

- Incorporate $\frac{1}{k}$ into polynomial approximants:
  \[
  \frac{\sin(x_\ell - corr)}{k} \quad \text{and} \quad \frac{\cos(x_\ell - corr)}{k}
  \]

- $k$ should be a small least common multiple (LCM) of any combination of one hypotenuse $c$ per table entry
  - small $\rightarrow$ each $A_i$ and $B_i$ fits in a machine word
1 Background on sine and cosine evaluation
   - Additive reduction
   - Table-lookup
   - Sources of error

2 Error-free table method for sine and cosine implementations
   - Concentration of the error
   - Tabulating rational numbers
   - Pythagorean triples for trigonometric functions

3 Experimental results
   - Exhaustive search
   - Heuristic search
   - Comparisons with other table-based methods

4 Conclusion and perspectives
Table generation algorithm

- **Input**
  - p: table index size

- **Algorithm steps**
  1. \( n \leftarrow 4 \)
  2. repeat
  3. Generate all PPTs \((a, b, c)\) such that \( c \leq 2^n \).
  4. Search for the LCM \( k \) among all generated hypotenuses \( c \).
  5. \( n \leftarrow n + 1 \)
  6. until such a \( k \) is found
  7. Build tabulated values \((A, B, corr)\) for every entry.
Exhaustive search

Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz (32 cores) 125 GB RAM

<table>
<thead>
<tr>
<th>$p$</th>
<th>$k_{min}$</th>
<th>$n$</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>725</td>
<td>10</td>
<td>$\ll 1$</td>
</tr>
<tr>
<td>4</td>
<td>10,625</td>
<td>14</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>130,645</td>
<td>17</td>
<td>0.14</td>
</tr>
<tr>
<td>6</td>
<td>1,676,285</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>32,846,125</td>
<td>25</td>
<td>1000</td>
</tr>
</tbody>
</table>
Exhaustive search

Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz (32 cores) 125 GB RAM

<table>
<thead>
<tr>
<th>$p$</th>
<th>$k_{min}$</th>
<th>$n$</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>725</td>
<td>10</td>
<td>$\ll 1$</td>
</tr>
<tr>
<td>4</td>
<td>10,625</td>
<td>14</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>130,645</td>
<td>17</td>
<td>0.14</td>
</tr>
<tr>
<td>6</td>
<td>1,676,285</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>32,846,125</td>
<td>25</td>
<td>1000</td>
</tr>
</tbody>
</table>

- **Bottlenecks** = searching the LCM and memory
- Finding $k$ for a 10-bit addressed table is desperate with this exhaustive technique
  - our estimations show that $n = \lceil \log_2(k_{min}) \rceil \approx 37$
Heuristic search

<table>
<thead>
<tr>
<th>$p$</th>
<th>$k_{min}$</th>
<th>Prime Factorization</th>
<th>Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>130,645</td>
<td>$5 \cdot 17 \cdot 29 \cdot 53$</td>
<td>21,588</td>
</tr>
<tr>
<td>6</td>
<td>1,676,285</td>
<td>$5 \cdot 13 \cdot 17 \cdot 37 \cdot 41$</td>
<td>338,660</td>
</tr>
<tr>
<td>7</td>
<td>32,846,125</td>
<td>$5^3 \cdot 13 \cdot 17 \cdot 29 \cdot 41$</td>
<td>5,365,290</td>
</tr>
</tbody>
</table>

**Observation:** $k$ is always a product of *small* Pythagorean primes

- Pythagorean primes less than 70:
  - $p = \{5, 13, 17, 29, 37, 41, 53, 61\}$
Heuristic search

<table>
<thead>
<tr>
<th>$p$</th>
<th>$k_{min}$</th>
<th>Prime Factorization</th>
<th>Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>130,645</td>
<td>$5 \cdot 17 \cdot 29 \cdot 53$</td>
<td>21,588</td>
</tr>
<tr>
<td>6</td>
<td>1,676,285</td>
<td>$5 \cdot 13 \cdot 17 \cdot 37 \cdot 41$</td>
<td>338,660</td>
</tr>
<tr>
<td>7</td>
<td>32,846,125</td>
<td>$5^3 \cdot 13 \cdot 17 \cdot 29 \cdot 41$</td>
<td>5,365,290</td>
</tr>
</tbody>
</table>

**Observation:** $k$ is always a product of *small* Pythagorean primes

- Pythagorean primes less than 70:
  $p = \{5, 13, 17, 29, 37, 41, 53, 61\}$
- During the generation, **only keep triples with hypotenuse**

$$c = \prod_{i} p[i]^{r_i} \quad \text{with} \quad \begin{cases} r_i = 0 \ \text{or} \ \ 1 & \text{if } p[i] \neq 5 \\ r_i \in \mathbb{N} & \text{if } p[i] = 5 \end{cases}$$
**Heuristic search results**

<table>
<thead>
<tr>
<th>$p$</th>
<th>$k_{\text{min}}$</th>
<th>$n$</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1,676,285</td>
<td>21</td>
<td>0.12</td>
</tr>
<tr>
<td>7</td>
<td>32,846,125</td>
<td>25</td>
<td>0.88</td>
</tr>
<tr>
<td>8</td>
<td>243,061,325</td>
<td>28</td>
<td>4.63</td>
</tr>
<tr>
<td>9</td>
<td>12,882,250,225</td>
<td>34</td>
<td>269</td>
</tr>
<tr>
<td>10</td>
<td>370,668,520,625</td>
<td>39</td>
<td>8563</td>
</tr>
</tbody>
</table>

- **Same results** as the exhaustive search for $p \leq 7$
- **Exponentially better** than exhaustive search with respect to $p$
- New **bottleneck**: heuristic test over $\approx 2^n$ hypotenuses
Background on sine and cosine evaluation
- Additive reduction
- Table-lookup
- Sources of error

Error-free table method for sine and cosine implementations
- Concentration of the error
- Tabulating rational numbers
- Pythagorean triples for trigonometric functions

Experimental results
- Exhaustive search
- Heuristic search
- Comparisons with other table-based methods

Conclusion and perspectives
Comparisons with other table-based methods

- Correct rounding in double precision
  - 2-step Ziv strategy:
    - quick phase accurate to $2^{-66}$
    - slow phase accurate to $2^{-150}$

- Table index size $p = 10$ bits

- Estimate memory accesses (MA), FLOPs, and table size

<table>
<thead>
<tr>
<th>Method</th>
<th>Table size (B)</th>
<th>Quick phase</th>
<th>Slow phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tang</td>
<td>38,640</td>
<td>4 MA + 64 FLOP</td>
<td>6 MA + 241 FLOP</td>
</tr>
<tr>
<td>Gal</td>
<td>57,960</td>
<td>3 MA + 53 FLOP</td>
<td>9 MA + 268 FLOP</td>
</tr>
<tr>
<td>Proposed</td>
<td>32,200</td>
<td>3 MA + 53 FLOP</td>
<td>5 MA + 148 FLOP</td>
</tr>
</tbody>
</table>

- Table-size 17% lower than Tang’s, 45% lower than Gal’s
- Quick phase 25% + 17% less expensive than Tang’s, same as Gal’s
- Slow phase 17% + 39% less expensive than Tang’s, 45% + 45% than Gal’s
Comparisons with other table-based methods

- Correct rounding in double precision
  - 2-step Ziv strategy:
    - quick phase accurate to $2^{-66}$
    - slow phase accurate to $2^{-150}$

- Table index size $p = 10$ bits

- Estimate memory accesses (MA), FLOPs, and table size

<table>
<thead>
<tr>
<th>Method</th>
<th>Table size (B)</th>
<th>Quick phase</th>
<th>Slow phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tang</td>
<td>38,640</td>
<td>4 MA + 64 FLOP</td>
<td>6 MA + 241 FLOP</td>
</tr>
<tr>
<td>Gal</td>
<td>57,960</td>
<td>3 MA + 53 FLOP</td>
<td>9 MA + 268 FLOP</td>
</tr>
<tr>
<td>Proposed</td>
<td>32,200</td>
<td>3 MA + 53 FLOP</td>
<td>5 MA + 148 FLOP</td>
</tr>
</tbody>
</table>

- Table-size 17% lower than Tang’s, 45% lower than Gal’s
- Quick phase 25% + 17% less expensive than Tang’s, same as Gal’s
- Slow phase 17% + 39% less expensive than Tang’s, 45% + 45% than Gal’s
Comparisons with other table-based methods

- Correct rounding in double precision
  - 2-step Ziv strategy:
    - quick phase accurate to $2^{-66}$
    - slow phase accurate to $2^{-150}$

- Table index size $p = 10$ bits

- Estimate memory accesses (MA), FLOPs, and table size

<table>
<thead>
<tr>
<th>Method</th>
<th>Table size (B)</th>
<th>Quick phase</th>
<th>Slow phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tang</td>
<td>38,640</td>
<td>4 MA + 64 FLOP</td>
<td>6 MA + 241 FLOP</td>
</tr>
<tr>
<td>Gal</td>
<td>57,960</td>
<td>3 MA + 53 FLOP</td>
<td>9 MA + 268 FLOP</td>
</tr>
<tr>
<td>Proposed</td>
<td>32,200</td>
<td>3 MA + 53 FLOP</td>
<td>5 MA + 148 FLOP</td>
</tr>
</tbody>
</table>

- Table-size 17% lower than Tang’s, 45% lower than Gal’s
- Quick phase 25% + 17% less expensive than Tang’s, same as Gal’s
- Slow phase 17% + 39% less expensive than Tang’s, 45% + 45% than Gal’s
Comparisons with other table-based methods

- Correct rounding in double precision
  - 2-step Ziv strategy:
    - quick phase accurate to $2^{-66}$
    - slow phase accurate to $2^{-150}$

- Table index size $p = 10$ bits

- Estimate memory accesses (MA), FLOPs, and table size

<table>
<thead>
<tr>
<th>Method</th>
<th>Table size (B)</th>
<th>Quick phase</th>
<th>Slow phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tang</td>
<td>38,640</td>
<td>4 MA + 64 FLOP</td>
<td>6 MA + 241 FLOP</td>
</tr>
<tr>
<td>Gal</td>
<td>57,960</td>
<td>3 MA + 53 FLOP</td>
<td>9 MA + 268 FLOP</td>
</tr>
<tr>
<td>Proposed</td>
<td>32,200</td>
<td>3 MA + 53 FLOP</td>
<td>5 MA + 148 FLOP</td>
</tr>
</tbody>
</table>

- Table-size 17% lower than Tang’s, 45% lower than Gal’s
- Quick phase 25% + 17% less expensive than Tang’s, same as Gal’s
- Slow phase 17% + 39% less expensive than Tang’s, 45% + 45% than Gal’s
1 Background on sine and cosine evaluation
   - Additive reduction
   - Table-lookup
   - Sources of error

2 Error-free table method for sine and cosine implementations
   - Concentration of the error
   - Tabulating rational numbers
   - Pythagorean triples for trigonometric functions

3 Experimental results
   - Exhaustive search
   - Heuristic search
   - Comparisons with other table-based methods

4 Conclusion and perspectives
We propose:

- a **new algorithm** for table-based range reductions:
  - using **error-free** values (integers)
  - **concentrating** the error on the other steps
  - estimated gains of **45% in table-size, FLOPs and memory accesses** for correctly-rounded double precision implementations

- a **prototype** able to pre-compute tables up to 10 indexing bits

Perspectives:

- Evaluation of **accuracy and performance** in available libraries
- Consider **other functions** (sinh, cosh, . . .)
- **Integration** in a full code-generation chain (MetaLibm)
Range Reduction Based on Pythagorean Triples for Trigonometric Function Evaluation

IEEE ASAP 2015 Conference

Hugues de Lassus Saint-Geniès    David Defour    Guillaume Revy

DALI, Université de Perpignan Via Domitia
LIRMM, Université de Montpellier
CNRS UMR 5506

July 27th, 2015
Heuristic gains

- Exponential speedup
- Exponential memory reduction