Reduced Precision Elementary Functions

Jean-Michel Muller
CNRS/LIP, ENS Lyon
France

SIAM 2021 Conference on Computational Science and Engineering
Use of reduced-precision elementary functions?

Sometimes a very fast rough approximation is useful:

- **first step** of a more accurate, iterative, computation;
- **computer vision, multimedia:**
  - tolerate a slight loss in accuracy,
  - need **real-time processing**;
  - frequently deal with inherently inaccurate data;
- **activation functions** (sигmoids, тanh...) in neural nets;
- **entertainment:**
  - Super Mario’s pizza does not need to follow the laws of physics accurately,
  - **fluidity** matters.
Small number formats: the blessing of exhaustivity

- **easy testing** (try all possible inputs);
- **correct rounding** becomes easily achievable → reproducible results;
  
  [Low precision *and* very accurate!]

- **tabulation** becomes an option;
- the return of **hw implementation** of a kernel of functions?
  
  - was the case a long time ago (Intel 8087 and followers);
  - Hw faster than Sftw,
  - shortcoming: impossible to fix or improve an already shipped circuit...

  *partly vanishes with exhaustive testing and correct rounding.*
Methods for elementary function evaluation

Initial range reduction to some interval $I$.

For input variables $\in I$,

1. polynomial or rational approximations (software); or
2. table-based methods (hardware, software); or
3. shift-and-add algorithms (hardware);
Initial **range reduction** to some interval $I$.

For input variables $\in I$,

1. **polynomial or rational approximations** (software); or
2. **table-based methods** (hardware, software); or
3. **shift-and-add algorithms** (hardware);
4. **low precision only**: **bit-manipulation techniques**.
Reduced precision and polynomial or rational approximations

Less accuracy constraints → larger domains and/or smaller degrees.

- simpler, or even non-necessary range reduction when uniform approximations can be used.
  Example [Girones et al., 2013, Computer vision]:
  \[ k = \frac{153}{256}, \ \ \mu = \frac{201}{128}, \]
  \[ \arctan(x) \approx \text{sign}(x) \cdot \mu \cdot \frac{k \cdot |x| + x^2}{2k \cdot |x| + x^2 + 1}, \]
  with error \(< 0.0029.\]

  No branchings → vector implementations much easier.

- small degrees of polynomial or rational approximations
  → fine tuning of approximations,
  → exhaustive search of “best” evaluation schemes,
    - in terms of speed (optimal use of arithmetic pipeline)
    - or in terms of evaluation error bound.
Reduced precision → **tabulation** becomes a very interesting option...
Reduced precision → **tabulation** becomes a very interesting option...

*No, it's not boring news, it can be cleverly done!*
Tabulating functions (hardware)

Reduced precision $\rightarrow$ **tabulation** becomes a very interesting option... 

*No, it’s not boring news, it can be cleverly done!*

With current technology:

- **small tables**: implemented as *boolean functions* of the entries (specific optimizations);
- **big tables**: all values are stored (matrix of 0s and 1s). Address decoding + propagation of signal through a line of the matrix.

$\rightarrow$ **big gap**. Threshold at around $2^{10}$–$2^{12}$ elements.
The Bipartite-Table Method

- hardware-oriented. Goal: overcoming the "$2^{12}$-element limit"
- Matula & Das Sarma (1995): reciprocal tables (seeds for NR)

![Figure 1: A $j+2=3k+1$ bits-in $j=3k-1$ bits out Faithful Bipartite Reciprocal table](image)

- arbitrary (regular enough) functions: Schulte, Stine, (1997).
Approximate $f(x)$, where $x$ is a $p$-bit fixed-point number in $[0, 1]$;

- split $x$ into three $k$-bit numbers $x_0$, $x_1$, and $x_2$, $k = \lceil p/3 \rceil$:
  \[ x = x_0 + 2^{-k}x_1 + 2^{-2k}x_2, \]
  with $x_i$ multiple of $2^{-k}$ and $0 \leq x_i < 1$.

- approximate $f(x)$ by $A(x_0, x_1) + B(x_0, x_2)$, where
  \[
  \begin{align*}
  A(x_0, x_1) &= f(x_0 + 2^{-k}x_1 + 2^{-2k-1}) \\
  B(x_0, x_2) &= 2^{-2k} \left( x_2 - \frac{1}{2} \right) \cdot f'(x_0).
  \end{align*}
  \]

- 2 tables of $2p/3$ address bits instead of one with $p$ address bits.
A small example

**Figure 1:** Error of a bipartite approximation of $\ln(x)$ in $[\frac{1}{2}, 1]$ with $k = 5$, compared with error of a 15-address-bit table (table size $16 \times$ larger).
A very odd trick (game Quake III, 1999)

A similar trick first appears in the game Quake III Arena.

1-bit right shift

No operation, just consider it is an integer

Integer addition

No operation, just consider it is a FP number

\[ y \approx \sqrt{x} \]
Bit-manipulation techniques

- use the fact that the exponent field of $x$ encodes $\lfloor \log_2 |x| \rfloor$.
- **Binary32** (a.k.a. single precision) representation of $x$:

<table>
<thead>
<tr>
<th>$S_x$</th>
<th>$E_x$</th>
<th>$F_x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>23</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 1-bit sign $S_x$, 8-bit biased exponent $E_x$, 23-bit fraction $F_x$ s.t.

$$x = (-1)^{S_x} \cdot 2^{E_x - 127} \cdot (1 + 2^{-23} \cdot F_x).$$

- the same bit-chain, if interpreted as 2’s complement integer, represents the number

$$I_x = (1 - 2S_x) \cdot 2^{31} + (2^{23} \cdot E_x + F_x).$$
Bit-manipulation method for $\sqrt{x}$

Remember:

$$x = (-1)^{S_x} \cdot 2^{E_x - 127} \cdot (1 + 2^{-23} \cdot F_x) = (-1)^{S_x} \cdot 2^{e_x} \cdot (1 + f_x).$$

<table>
<thead>
<tr>
<th>$S_x$</th>
<th>$E_x$</th>
<th>$F_x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>23</td>
</tr>
</tbody>
</table>

- If $e_x = E_x - 127$ is even (i.e., $E_x$ is odd), we use:
  $$\sqrt{(1 + f_x) \cdot 2^{e_x}} \approx \left(1 + \frac{f_x}{2}\right) \cdot 2^{e_x/2}, \quad (1)$$

- If $e_x$ is odd (i.e., $E_x$ is even), we use:
  $$\sqrt{(1 + f_x) \cdot 2^{e_x}} = \sqrt{4 + \epsilon_x} \cdot 2^{\frac{e_x - 1}{2}}$$
  $$\approx (2 + \frac{\epsilon_x}{4}) \cdot 2^{\frac{e_x - 1}{2}}$$
  $$= \left(\frac{3}{2} + \frac{f_x}{2}\right) \cdot 2^{\frac{e_x - 1}{2}}, \quad (2)$$

(Taylor series for $\sqrt{4 + \epsilon_x}$ at $\epsilon_x = 0$, with $\epsilon_x = 2f_x - 2$)
Bit-manipulation method for $\sqrt{x}$ (Blinn)

$$x = (-1)^{S_x} \cdot 2^{E_x-127} \cdot (1 + 2^{-23} \cdot F_x) = (-1)^{S_x} \cdot 2^{e_x} \cdot (1 + f_x).$$

<table>
<thead>
<tr>
<th>$S_x$</th>
<th>$E_x$</th>
<th>$F_x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>23</td>
</tr>
</tbody>
</table>

- $E_x$ odd $\rightarrow (1 + \frac{f_x}{2}) \cdot 2^{\frac{e_x}{2}}$,

$$1 + F_y \cdot 2^{-23} \cdot 2^{E_y-127} \approx (1 + F_x \cdot 2^{-24}) \cdot 2^{\frac{E_x-127}{2}}$$

$$\Rightarrow E_y = \frac{E_x+127}{2} \text{ and } F_y = \lfloor \frac{F_x}{2} \rfloor$$

- $E_x$ even $\rightarrow \left(\frac{3}{2} + \frac{f_x}{2}\right) \cdot 2^{\frac{e_x-1}{2}}$.

$$(1 + F_y \cdot 2^{-23}) \cdot 2^{E_y-127} \approx \left(\frac{3}{2} + F_x \cdot 2^{-24}\right) \cdot 2^{\frac{E_x-128}{2}}$$

$$\Rightarrow E_y = \frac{E_x+127}{2} - \frac{1}{2} \text{ and } F_y = 2^{22} + \lfloor \frac{F_x}{2} \rfloor$$

In both cases:

$$I_y = \left[\frac{l_x}{2}\right] + 127 \cdot 2^{22}$$
Bit-manipulation method for $\sqrt{x}$ (Blinn)

**Figure 2:** Plot of $(\text{approx} - \sqrt{x})/\sqrt{x}$.

- fast but rough approximation;
- *always* $\geq \sqrt{x} \rightarrow$ replace $127 \cdot 2^{22}$ by a smaller value?
$\sqrt{x}$ with a better constant

Figure 3: Plot of $(\text{approx} - \sqrt{x})/\sqrt{x}$ with $127 \cdot 2^{22}$ replaced by 532369100.
Quake III function (1999): inverse square root

- $1/\sqrt{x}$, possibly followed by Newton-Raphson iteration;
- reasoning very similar to above-presented $\sqrt{x}$.

$$I_y = -\left\lfloor \frac{I_x}{2} \right\rfloor + 1597463007$$

- non-optimal constant: 1597465647 is slightly better.

**Figure 4:** Plot of $(\text{approx } -1/\sqrt{x}) \times \sqrt{x}$ with the “magic constant” 1597463007.
Bit-manipulation technique for the logarithm (Blinn)

\[ 2^{-23}(l_x - l_1) = (E_x - 127) + 2^{-23} \cdot F_x \]
\[ = e_x + f_x \]
\[ \approx \log_2(x). \]

\[ \rightarrow \log_2(x) \approx 2^{-23} \cdot \text{Float}(l_x - l_1). \]

where Float(l) is the FP number mathematically equal to l.

Figure 5: Approx. to $\log_2(x)$

Linear interpolation at powers of 2.
And the winner is...
And the winner is... Nobody!

Different targets: hw vs sftw, low accuracy only vs “scalable”, only a few functions vs versatile.

- **Software:**
  - if error < a few % suffices, *bit-manipulation techniques* hard to beat (typically 1 integer addition and 1 shift). They don’t scale up and are not versatile;
  - otherwise: polynomial/rational (scalable and versatile).

- **Hardware:**
  - *bipartite methods* very versatile, do not scale-up well;
Thank you!