Pipelined Architecture for Accurate Floating Point Range Reduction

Francisco J. Jaime, Julio Villalba, Javier Hormigo, Emilio L. Zapata
Dept. Computer Architecture
University of Malga
SPAI N
Talk Outline

- Introduction
  - Posing the problem
- Double residue modular range reduction
- Pipelined Architecture
- Radix-4 coding
- Summary and conclusions
Introduction

• Range reduction is required for elementary functions evaluation

• Reductions:
  • Additive
    – Subtracting a multiple of a constant
  • Multiplicative
    – Multiplying by the inverse of a power of a constant

• Our algorithm
  – Additive reduction
    • Trigonometric functions
Introduction

• The reduction can be done in software or in hardware
• Our proposal is oriented toward a hardware implementation for application-specific systems
  – Consider both fixed-point & floating point representation
  – Maximum accuracy for floating point for the specific representation
Introduction

• A poor range reduction may lead to catastrophic accuracy problems in the evaluation of trigonometric functions when

1. the input argument is close to a multiple of the constant (i.e. \( \pi \))

2. the input argument is large [1]
Steps in the evaluation of trigonometric functions:

1) Reduction of the input argument to the interval \([0, 2\pi)\)
2) Compute the function inside the interval

Ej.: \(\sin(114 \text{ rad})\)
   1) Reduction 114 rad \(\rightarrow\) 0.9 rad
   2) Computation \(\sin(0.9 \text{ rad}) = 0.7850\)

A not accurate reduction (step 1) can lead to inaccurate results
1) Example of input argument close a multiple of $\pi$
   - Computation of $\sin(22 \text{ rad})$
1) Example of an input argument close a multiple of $\pi$ 
   \[ \text{Computation of } \sin(22 \text{ rad}) \]
1) Arguments close to a multiple of $\pi$
   - Computation of $\sin(22 \text{ rad})$

$$R = 22 - 7\pi = 0.0000\ 0001\ 0001\ 1101\ 0011\ 0001\ 1001$$
Introduction

2) Large input argument

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D9 FE</td>
<td>FSIN</td>
<td>Replace ST(0) with its sine.</td>
</tr>
</tbody>
</table>

**FSIN—Sine**

**Description**

This instruction calculates the sine of the source operand in register ST(0) and stores the result in ST(0). The source operand must be given in radians and must be within the range $-2^{63}$ to $+2^{63}$. The following table shows the results obtained when taking the sine of various classes of numbers, assuming that underflow does not occur.

<table>
<thead>
<tr>
<th>SRC (ST(0))</th>
<th>DEST (ST(0))</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-\infty$</td>
<td>$\ast$</td>
</tr>
</tbody>
</table>

Range of representable numbers for IEEE 754 (single precision): $[-2^{127},2^{127}]$
Introduction

- Solutions
  - Software
  - Hardware
Introduction

- **Solutions**
  - **Software**
    - Cody and Waite [3]
      - Software Manual for elementary functions (Prent. Hall 1980)
    - Payne and Hanek algorithm [5]
      - Radian reduction for trigonometric functions (SIGNUN let. 1983)
    - Brisebarre, Defour, Kornerup, Muller, Revol
      - A new range reduction algorithm (Trans. Comp. 2005)

Software solutions: good accuracy, slow
Introduction

- Solutions
  - Hardware
    - Double Residue Modular range reduction [6]
      - Based on a table with positive and negative residues
      - Solves both problems
        » Input argument close to multiple of a constant
        » Large argument
      - Architecture: fast, simple hardware and valid for fixed-point & floating-point when the module is a rational or an irrational number.
        » IEEE 754 full range
    - Handicap for floating point:
      » word-serial implementation only

Our aim:
- Design a pipelined architecture supporting the double residue modular range reduction
  - preventing the replication of the table of residues
Double residue modular range reduction
Double residue MRR

- Fundamental of double residue MRR (*fixed-point*)
  - For a number $X$ and a module $M$, obtain a number $R$:
    \[
    R = X - k \cdot M \quad 0 \leq R < M
    \]

- Example: $M=7$, $X=372$
  \[
  R = 372 - k \cdot 7 \quad 0 \leq R < 7
  \]
  - Select $k=53$, thus $R=1$

- We call “residue” to $R$ and “mod” to the operation, such that:
  \[
  R = X \mod M \quad (372 \mod 7 = 1)
  \]
Double residue MRR

- Double residue MRR
  - Define elementary positive residues and elementary negative residues, which are stored in a table:

\[
\begin{align*}
  m_i^+ &= 2^i \mod M \\
  m_i^- &= 2^i \mod M - M
\end{align*}
\]

- Example: M=7, we define

\[
\begin{align*}
  m_6^+ &= 2^6 \mod 7 = 64 \mod 7 = 1 \\
  m_6^- &= 2^6 \mod 7 - 7 = -6 \\
  m_5^+ &= 2^5 \mod 7 = 32 \mod 7 = 4 \\
  m_5^- &= 2^5 \mod 7 - 7 = -3 \\
  m_4^+ &= 2^4 \mod 7 = 16 \mod 7 = 2 \\
  m_4^- &= 2^4 \mod 7 - 7 = -5
\end{align*}
\]
Double residue MRR

- **Double residue MRR**
  - **Algorithm:**
    - Define an accumulated residue $R(i)$
    - For each power of 2, select the suitable positive or negative residue
    - The accumulated residue is bounded by $M$
      - $R(i)$ is always inside convergence bound

$$X = x_{n-1} \ldots x_1 x_0 x_{-1} x_{-2} \ldots x_{-f}$$

$$R(i + 1) = R(i) + m^*_i x_i$$

$$m^*_i = \begin{cases} m^+_i & \text{if } R(i) < 0 \\ m^-_i & \text{if } R(i) \geq 0 \end{cases}$$
Double residue MRR

- **Double residue MRR: an intuitive idea**
  - Computation of $372 \mod 7 = 1$
  - $372 = 101110100 = 2^8 + 2^6 + 2^5 + 2^4 + 2^2 = 256 + 64 + 32 + 16 + 4$
    - $(256 \mod 7 + 64 \mod 7 + 32 \mod 7 + 16 \mod 7 + 4 \mod 7) = (4, -3) + (1, -6) + (4, -3) + (2, -5) + (4, -3)$
    - $= (-3 + 1 + 4 + -5 + 4) = 1$
Double residue MRR

- **Double residue MRR: an intuitive idea**
  - Computation of $372 \mod 7 = 1$
  - $372 = 101110100 = 2^8 + 2^6 + 2^5 + 2^4 + 2^2 = 256 + 64 + 32 + 16 + 4$

![Diagram of double residue MRR]
Double residue MRR

- Double residue MRR: an intuitive idea
  - Computation of $372 \mod 7 = 1$
    - $372 = 101110100 = 2^8 + 2^6 + 2^5 + 2^4 + 2^2 = 256 + 64 + 32 + 16 + 4$

```
R(1) = -3
```
Double residue MRR

- **Double residue MRR: an intuitive idea**
  - Computation of $372 \mod 7 = 1$
  - $372 = 101110100 = 2^8 + 2^6 + 2^5 + 2^4 + 2^2 = 256 + 64 + 32 + 16 + 4$

![Diagram showing modular residues and powers of 2 modulus 7]
Double residue MRR

- Double residue MRR: an intuitive idea
  - Computation of $372 \mod 7 = 1$
  - $372 = 101110100 = 2^8 + 2^6 + 2^5 + 2^4 + 2^2 = 256 + 64 + 32 + 16 + 4$

\[
\begin{array}{c|c}
\text{i mod 7} & \text{2i mod 7} \\
-3 & 4 \\
-6 & 1 \\
-3 & 4 \\
-5 & 2 \\
-3 & 4 \\
\end{array}
\]

\[R(3)=2\]
Double residue MRR

- Double residue MRR: an intuitive idea
  - Computation of $372 \mod 7 = 1$
    - $372 = 101110100 = 2^8 + 2^6 + 2^5 + 2^4 + 2^2 = 256 + 64 + 32 + 16 + 4$

\[\begin{array}{c|c}
-3 & 4 \\
-6 & 1 \\
-3 & 4 \\
-5 & 2 \\
-3 & 4 \\
2^8 \mod 7 & 2^6 \mod 7 \\
2^5 \mod 7 & 2^4 \mod 7 \\
2^3 \mod 7 & 2^2 \mod 7 \\
2^1 \mod 7 & 2^0 \mod 7 \\
\end{array}\]

\[R(4) = -3\]

\[-3 + 1 + 4 - 5 + 4 = 1\]
Double residue MRR

- **Word-serial Architecture:**

![Diagram of word-serial architecture with symbols for $m_i^+$, $m_i^-$, mux, and adder, leading to a reduced argument output.]
Double residue MRR

- Implementation in carry-save arithmetic
  - \( R(i) = S(i) + C(i) \)
  - Estimation of the sign of \( R(i) \) from the 3-MSBs of \( R(i) \):
    - Define a new selection function of the elementary residues:

\[
m_i^* = \begin{cases} 
  m_i^+ & \text{if } R(i) < 0 \\
  m_i^- & \text{if } R(i) \geq 0
\end{cases}
\]

\[
\overline{R}(i) \quad \text{Truncated value of } R(i) \text{ using the 3 MSBs of } S(i), C(i)
\]
Double residue MRR

- Implementation in carry-save arithmetic
  - The iterations are self-correcting
    - The deviation produced by a wrong estimation of the sign of $R(i)$ is corrected by the next iteration
    - A final correction iteration is required to ensure the convergence
Double residue MRR

Architecture in carry-save arithmetic (fixed-point)

\[ R(i + 1) = R(i) + m_i x_i \]

\[ R(0) = X_L \]

\[ X = x_{n-1} \ldots x_1 x_0, x_{-1} x_{-2} \ldots x_{-f} \]
Double residue MRR


\[ R(i + 1) = R(i) + m_i x_i \]
Pipelined architecture
Pipelined architecture

- Fixed-point pipelined architecture for a fixed M
  - Distributed elementary residues $m_i$ (hardwired)
Pipelined architecture

- Floating point pipeline implementation
  - Problem with the pipeline implementation for a fixed M:
    - Requires replication of the table (as many tables as stages)
Pipelined architecture

- Our proposal:
  - Prevents the replication of the table by generation of the elementary residue for the next stage
Pipelined architecture

- Preventing the replication:
  - Define a *elementary residue generator* as:

\[
\begin{align*}
m_{i+1}^+ &= \begin{cases} 2m_i^+ & \text{if } 2m_i^+ < M \\ 2m_i^+ - M & \text{if } 2m_i^+ \geq M \end{cases} \\
m_{i+1}^- &= \begin{cases} 2m_i^- + M & \text{if } 2m_i^- < M \\ 2m_i^- & \text{if } 2m_i^- \geq M \end{cases}
\end{align*}
\]

- A single table of elementary residues is required
- The elementary residue for the stage \( i \) is obtained from the elementary residue of the previous stage \( i-1 \)
Pipelined architecture

- Reducing the hardware cost of the elementary residue generator

\[
\begin{align*}
\text{if} & \quad 2m_i^+ < M \quad \text{then} & \quad \begin{cases} 
    m_{i+1}^+ &= 2m_i^+ \\
    m_{i+1}^- &= 2m_i^- + M 
\end{cases} & \quad \text{FLAG}_i = 0 \\
\text{else} & \quad \begin{cases} 
    m_{i+1}^+ &= 2m_i^+ - M \\
    m_{i+1}^- &= 2m_i^- 
\end{cases} & \quad \text{FLAG}_i = 1
\end{align*}
\]

- A single comparison is required
- The evolution of the generation of elementary residues can be precomputed and stored (Flags)

  » Comparison is prevented
  » Only one addition is required in each stage
Pipelined architecture

- Example (M=7)

X = 1.101001.. 2^{10}

\[
\begin{align*}
\text{Stage 0:} & \quad m_0^+ = 5, m_0^- = -2 \\
\text{Stage 1:} & \quad m_1^+ = 5 \times 2^2 - 7 = 3, m_1^- = -2 \times 2^1 = -4 \\
\text{Stage 2:} & \quad m_2^+ = 3 \times 2^3 - 7 = 6, m_2^- = 4 \times 2^2 + 7 = 1 \\
\text{Stage 3:} & \quad m_3^+ = 6 \times 2^3 - 7 = 5, m_3^- = 1 \times 2^2 = -2 \\
\end{align*}
\]

Flags vector: 1 0 1 1 0 ...

\[
\begin{align*}
\text{if } 2m_i^+ & < M \text{ then } \begin{cases} m_{i+1}^+ = 2m_i^+ \\ m_{i+1}^- = 2m_i^- - M \end{cases} \\
\text{else } & \begin{cases} m_{i+1}^+ = 2m_i^+ - M \\ m_{i+1}^- = 2m_i^- \end{cases}
\end{align*}
\]
Pipelined architecture

- Carry-save elementary residue generator (module)

\[
\begin{align*}
M &\quad -M \\
2 &\quad 2 \\
2 &\quad 2 \\
2 &\quad 2 \\
\end{align*}
\]

\[
\begin{align*}
\text{FLAG} &\quad \text{FLAG} = 0 \\
&\quad \text{FLAG} = 1 \\
\end{align*}
\]

\[
\begin{align*}
m_{i+1}^+ &= 2m_i^+ \\
m_{i+1}^- &= 2m_i^- + M \\
m_i^+ &= 2m_i^+ - M \\
m_i^- &= 2m_i^-
\end{align*}
\]
Pipelined architecture

- Final Architecture

![Diagram showing pipelined architecture stages 0, 1, and 2 with operations for mantissa and EXP, flags, and element residual.]
Pipelined architecture

- Error propagation
  - Computing the accumulative addition $R(i)$
    \[ R(i + 1) = R(i) + m_i^* x_i \]
  - Computing elementary residues $m_i$
    \[ m_{i+1}^* = 2m_i^* - M \]

Aim: precision 1 ULP using guard bits
Pipelined architecture

- Error propagation
  - Computing the accumulative addition $R(i)$

$$R(i + 1) = R(i) + m_i x_i$$

- $n$ iterations:
  » guard bits: $g = \lfloor \log_2 n \rfloor$
Pipe-lined architecture

- Error propagation

- Computing elementary residues $m_i$

\[ m_{i+1}^* = 2m_i^* - M \]

Initial error $\varepsilon$:

\[ \varepsilon < 2^{-t-j} \]
Pipelined architecture

- Error propagation

  - Computing elementary residues $m_j$

    $y_1 y_2 y_3 y_4 \ldots y_t y_{t+1} y_{t+2} \ldots y_{t+g} y_{t+g+1} y_{t+g+2} \ldots y_{t+j}$

    Initial error $\varepsilon$: $\varepsilon < 2^{-t-j}$

    Recurrence:

    $\left\{\begin{array}{l}
a_0 = \varepsilon \\
a_k = (2a_{k-1} + 1)\varepsilon
\end{array}\right.$

    $j > g + \log_2 (2^{k+1} - 1)$
Pipelined architecture

- Error propagation conclusion:
  - Computing the accumulative addition $R(i)$
    \[ R(i + 1) = R(i) + m_i^* x_i \]
    Guard bits: $g = \lceil \log_2(n+1) \rceil$
  - Computing elementary residues $m_i$
    \[ m_{i+1} = 2m_i^* - M \]
    Guard bits: $j = n + \lceil \log_2(n+1) \rceil$
  - The number of guard bits required for $m_i$ decreases by one in every iteration
    - Hardware reduction in every stage
Pipelined architecture

- Architecture for IEEE-754 single precision (M=2π, n=24)

- g=5
- j=29
Future works

- **Radix-4 coding**
  \[ X = x_1x_2x_3 \ldots x_i \ldots x_n, x_i \in \{-2,-1,0,1,2\} \]
  - Reduces the number of stages by half
  - Selection function more complex
  - Increases the stage delay

\[ R(i+1) = R(i) + \sigma_i m_i^* \]

- Selection function more complex
  - Increases the stage delay
Summary and Conclusion

- **Novel pipeline architecture for range reduction**
  - Carry-save arithmetic
  - Based on double elementary residues
  - Prevent replication of tables of the word-serial version
    - obtaining next elementary residue from the previous one
      - Elementary residue generator
    - Precision 1 ULP

- **Valid for floating point reduction**
  - IEEE 754 compliant

- **Radix-4**
  Reduces the number of stages by half
THANK YOU!