Simultaneous floating-point sine and cosine for VLIW integer processors

Jingyan Jourdan-Lu

Computer Arithmetic group *ARIC*
INRIA and ENS Lyon, France

Compilation Expertise Center
STMicroelectronics Grenoble, France

Joint work with Claude-Pierre Jeannerod

Context and motivation

Fast and accurate floating-point support for integer-only microprocessors:

- VLIW cores, like STMicroelectronics' ST231
- Portable C code for IEEE 754 formats
- 'Fast' = low latencies
- 'Accurate' = proven error bounds

Basic operations ($+$, $-$, $\times$, $/$, $\sqrt{}$) are already available: FLIP library

But **no** fast and accurate support for high-level functions like trigonometric functions, logarithms, exponentials,...

Simultaneous sine and cosine (“sincos“)

Sine and cosine: ubiquitous in graphics and signal processing applications, and often evaluated at a same floating-point input x.

Classically, evaluation in 3 steps: [Muller'97, Ercegovac & Lang'04]

1. Range reduction: compute x^* such that

$$x^* \in [-\tfrac{\pi}{4}, \tfrac{\pi}{4}] \quad \text{and} \quad x^* = x - k\tfrac{\pi}{2}, \quad k \in \mathbb{Z}.$$

2. Evaluation of sin and cos at reduced argument $|x^*|$.

3. Reconstruction: $(\sin x, \cos x) = \begin{cases} (\pm \sin x^*, \cos x^*) & \text{if } k \text{ even}, \\ (\pm \cos x^*, \pm \sin x^*) & \text{if } k \text{ odd}. \end{cases}$

$\hookrightarrow$ **Problem:** how to implement step 2 accurately and fast on our target?

Our contributions

- **New algorithms for sine and cosine** over $[0, \frac{\pi}{4}]$:
  - Accurate: error proven to be at most of 1 ulp (unit in the last place)
  - Fast: 19 and 18 cycles on ST231
- **C code for sincos**
  - as fast as sine alone
  - faster than the correctly-rounded multiplication of FLIP
    (which takes 21 cycles, also via a pure software emulation)

$\hookrightarrow$ all this for single precision, including subnormals.

Outline

1. Overview of the ST231 processor
   - Architecture
   - Key features for floating-point support
2. Floating-point sine and cosine
   - Accuracy specification
   - Polynomial-based algorithms
   - Results for sinf, cosf, sincosf on ST231

ST231: overview

The ST231 is a 4-way VLIW integer processor from the ST200 family:

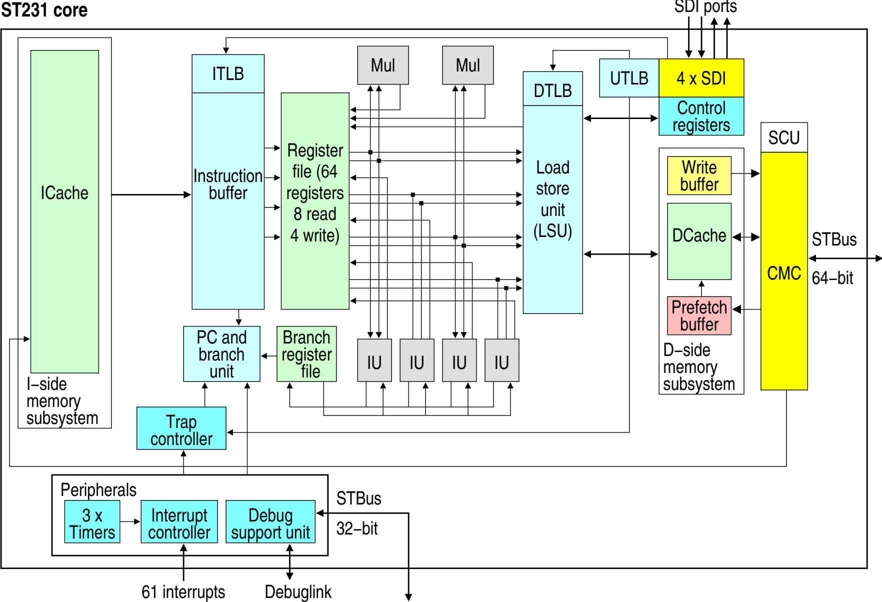
- Up to 4 instruction words grouped into 1 bundle
- Up to 4 instructions executed in 1 cycle

$\hookrightarrow$ key to realize **instruction level parallelism** (ILP)

Typical applications:

- a media processor with an embedded OS
- a host processor running Linux and applications

ST231: architecture



ST231: key features for floating-point support

- select instruction → transform branches to straight line code
- leading-zero count instruction → subnormal numbers support
- 32×32-bit multiplication → polynomial evaluation
- immediate number up to 32 bits
  can be encoded in 1 instruction word
  → masking, encoding of polynomial coefficients
  and of special floating-point values (NaN,...)
- min, max, shift-and-add $(a \ll b) + c$ with $b \in \{1, 2, 3, 4\}$

Multiplier features:

- Latency of 3 cycles
- At most 2 multiplications can be issued at each cycle

Notation for floating-point

- $\mathbb{F}$:= **binary32 finite floating-point numbers** of IEEE 754
- This set is defined by a precision p and an exponent range $[e_{\text{min}}, e_{\text{max}}]$ such that

$$p = 24, \qquad e_{\text{max}} = 1 - e_{\text{min}} = 127.$$

- It consists of
  1. signed zeroes $+0$ and -0
  2. subnormal numbers $\pm m \cdot 2^{e_{\text{min}}}$ with $m = (0.m_1 \dots m_{p-1})$ nonzero
  3. normal numbers $\pm m \cdot 2^{e}$ with $m = (1.m_1 \dots m_{p-1})$

- **Unit roundoff** $u := 2^{-p}$

Specification of sine and cosine

Input and output:

- 32-bit unsigned integers
- encode floats in the standard way:

$$x \in \mathbb{F} \quad \longleftrightarrow \quad X = \left[\text{sign bit}\middle|\text{biased exponent}\middle|\text{fraction bits}\right]$$

Specification of sine and cosine (cont'd)

Accuracy:

- The key tool is the **ulp function**:

$$\forall x \in \mathbb{R}, \qquad \mathsf{ulp}(x) = \begin{cases} 0 & \text{if } x = 0, \\ 2^{\max\{e_{\mathsf{min}}, e\} - p + 1} & \text{if } |x| \in [2^e, 2^{e+1}). \end{cases}$$

$\Rightarrow$ $\mathsf{ulp}(x) \leqslant 2u|x|$ for x normal.

- Given $f \in \{\sin, \cos\}$ and $x \in \mathbb{F} \cap [0, \frac{\pi}{4}]$, we want $r \in \mathbb{F}$ such that

$$\big| r - f(x) \big| \leqslant \mathsf{ulp}\big(f(x)\big).$$

$\hookrightarrow$ This is **"1-ulp accuracy"** ($\approx$ all bits correct but possibly the last one).
$\hookrightarrow$ Such a precise specification is indispensable for the mathematical proof of the accuracy of our codes.

Algorithms for sine and cosine

Since 32×32-bit multipliers are available, a classical approach is via the **evaluation of polynomial approximants** [Tang'90, Gal & Bachelis'91, ...]

$\hookrightarrow$ high-level algorithm for $x \in [0, \frac{\pi}{4}] \subset [0, 1)$:

1. If x close enough to zero then
   return x for $\sin x$, and $1 - u$ for $\cos x$
2. Else
   evaluate a pair of polynomials approximating $\sin x$ and $\cos x$

- Software toolchain for step 2: Sollya $\rightarrow$ CGPE $\rightarrow$ Gappa.
- Steps 1 and 2 are independent $\Longrightarrow$ obvious source of ILP.
- Much more ILP can be exposed at the polynomial evaluation level.

Polynomial evaluation for cosine

For our accuracy constraint, a polynomial of degree 6 is enough:

$$a(y) = a_0 + a_1 y + \cdots + a_6 y^6.$$

- Each a_i has $\leqslant 32$ fraction bits and is encoded in a uint32_t.
- y is a fixed-point approximation of x.
- We have chosen a **highly-parallel evaluation scheme**:

$$\Big((a_0 + a_1 y) + (a_2 + a_3 y) z\Big) + \Big((a_4 + a_5 y) + a_6 z\Big)(z^2)$$

with $z = y^2$.

$\hookrightarrow$ **accurate enough** and **2.2x faster than Horner's rule**

$$a_0 + y\big(\cdots + y(a_5 + a_6 y)\big).$$

Polynomial evaluation for sine

Over $[0, \frac{\pi}{4}]$, things are more difficult than for cosine:

- cosine was 'flat', ranging in $[0.707..., 1]$ $\Longrightarrow$ already fixed point
- sine ranges in $[0, 0.707...]$ $\Longrightarrow$ 'exponent' not known in advance

Classical workaround:

1. instead of $\sin x$, approximate the flat function $\frac{\sin x}{x}$ ranging in $[0.8, 1]$
2. reconstruct using $\sin x = \frac{\sin x}{x} \times x$

$\hookrightarrow$ drawback: **steps 1 and 2 are NOT independent**.

Polynomial evaluation for sine (cont'd)

Our solution is to interleave steps 1 and 2:

- For $x = m \cdot 2^e$ we have $\frac{\sin x}{x} \times x = m \frac{\sin x}{x} \cdot 2^e$
- View $m \frac{\sin x}{x}$ as a bivariate function and approximate it by

$$b(m, x) = \tfrac{u}{2} + m\, c(x), \qquad c(x) = c_0 + c_2 x^2 + c_4 x^4 + c_6 x^6$$

- Evaluate b at (m, y) using a highly-parallel evaluation scheme:

$$b(m, x) = \Big(\big(\tfrac{u}{2} + mc_0\big) + \big(mc_2\big)x \Big) + \Big(mc_4 + \big(mc_6\big)z \Big)\big(z^2\big),$$

with $z = x^2$.

Results on ST231 for sine, cosine, and sincos

Careful implementation in C for the binary32 format (= single precision), compiled by the st200cc compiler (in -O3 and for the ST231 core):

- Straight-line assembly code thanks to 'if conversion'
- The schedules produced are optimal
- C code for sincosf obtained by inlining

Results on ST231 for sine, cosine, and sincos (cont'd)

	latency L (cycles)	**instr. number N**	**IPC = N/L**
sinf	19	31	1.6
cosf	18	25	1.4
sincosf	19	47	2.4

- sincos as fast as sine alone
- faster than best-known software implementation of IEEE 754 floating-point multiplication (21 cycles in single precision)

Bundle occupancy and shared resources for sincos

Cycle	issue 1	issue 2	issue 3	issue 4
0	shared	shared		
1	shared	shared	shared	sin
2	shared	shared	sin	sin
3	shared	sin	sin	
4	shared	sin	sin	
5	sin	sin	cos	cos
6	cos	cos	cos	cos
7	shared	sin	sin	sin
8	sin	sin	cos	cos
9	cos	cos	cos	cos
10	sin	cos	cos	
11	sin	sin	cos	
12	cos			
13	cos			
14	sin	sin		
15	sin	sin	cos	
16	sin	sin	cos	
17	sin	cos	cos	
18	shared	sin	cos	cos

- In 80% of the bundles, at least 3 slots used
- shared resources:
  - unpacking of input x
  - $y \approx x$, y^2, y^4

Conclusion

Summary:

- Fast and 1-ulp accurate algorithms for sine and cosine
- Key ingredient = parallel evaluation of uni-/bi-variate polynomials
- C code for sincos having the same latency as sine alone

On-going work:

- Efficient range reduction $x \to x^*$ on VLIW integer architectures:
  - Easy case: $x = \pi y$ for any y in $\mathbb{F}$
  - General case: any x in $\mathbb{F}$
- Performance impact of relaxing the 1-ulp accuracy constraint