#### Elementary functions III

Florent de Dinechin, Arénaire Project, ENS-Lyon

Нижний Новгород, 23/06/2010.99999

Trigonometric functions

Logarithms

Gal's accurate table method



#### The few books

- J.M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhauser, 1997, 2006.
- P. Markstein. *IA-64 and Elementary Functions: Speed and Precision.* Prentice Hall, 2000.
- M. Cornea, J. Harrison and P.T.P. Tang. Scientific Computing on Itanium<sup>®</sup>-based Systems. Intel Press, 2002.

Also papers by P.T.P Tang (TOMS 1989, 1990, 1992), S. Gal and B. Bachelis (TOMS 1991), A. Ziv (TOMS 1991), K.C. Ng (Sun report 1992), J. Harrison, T. Kubaska, S. Story, P.T.P Tang, (Intel Tech Journal 1999), C. Anderson, N. Astafiev, S. Story (RNC, 2004), among others.

### **Trigonometric functions**

Trigonometric functions

Logarithms

Gal's accurate table method

Florent de Dinechin, Arénaire Project, ENS-Lyon Elementary functions III





• Range reduction thanks to trigonometric identities



• Range reduction thanks to trigonometric identities such as

• 
$$\sin(X) = \sin(X + 2k\pi)$$



• Range reduction thanks to trigonometric identities such as

• 
$$\sin(X) = \sin(X + 2k\pi)$$

• 
$$\sin(-X) = -\sin(X)$$



• Range reduction thanks to trigonometric identities such as

• 
$$\sin(X) = \sin(X + 2k\pi)$$

• 
$$\sin(-X) = -\sin(X)$$

• 
$$\sin(\frac{\pi}{2} - X) = \cos(X)$$



• Range reduction thanks to trigonometric identities such as

• 
$$\sin(X) = \sin(X + 2k\pi)$$

• 
$$\sin(-X) = -\sin(X)$$

• 
$$\sin(\frac{\pi}{2} - X) = \cos(X)$$

Good approximations to sine and cosine

in a small neighborood of 0

- Polynomial approximation
- CORDIC family of methods (additions only)



• Floating-point is a logarithmic representation in radix 2



- Floating-point is a logarithmic representation in radix 2
- The libm sine and cosine take radian arguments (period  $2\pi$ )



- Floating-point is a logarithmic representation in radix 2
- The libm sine and cosine take radian arguments (period  $2\pi$ )
- 2 and  $2\pi$  are irrational one to the other



- Floating-point is a logarithmic representation in radix 2
- The libm sine and cosine take radian arguments (period  $2\pi$ )
- 2 and  $2\pi$  are irrational one to the other
- Sine and cosine of large floating-point numbers are chaotic.

- Floating-point is a logarithmic representation in radix 2
- The libm sine and cosine take radian arguments (period  $2\pi$ )
- 2 and  $2\pi$  are irrational one to the other
- Sine and cosine of large floating-point numbers are chaotic.
  - people using them should probably be shot...

- Floating-point is a logarithmic representation in radix 2
- The libm sine and cosine take radian arguments (period  $2\pi$ )
- 2 and  $2\pi$  are irrational one to the other
- Sine and cosine of large floating-point numbers are chaotic.
  - people using them should probably be shot...
- However it is possible to compute them accurately, therefore we will compute them accurately.

# But first a lazy solution: functions $sin(\pi x)$ and $cos(\pi x)$



• Now the period is 2

# But first a lazy solution: functions $sin(\pi x)$ and $cos(\pi x)$



- Now the period is 2
- Behaviour for large arguments is not much more informative than the previous, but it is cheaper to compute

The trig-of- $\pi x$  have made it into IEEE 754-2008.

(and we even know how to round them correctly)

(same would hold for degree arguments)

#### Argument reduction for radian arguments



Goal: replace  $X \in \mathbb{R}$  with angle  $\alpha$  in  $\left[0, \frac{\pi}{4}\right]$ 



#### Argument reduction for radian arguments

Goal: replace  $X \in \mathbb{R}$  with angle  $\alpha$  in  $\left[0, \frac{\pi}{4}\right]$ 

- compute  $K = X \times \frac{4}{\pi}$
- integer part int(K) provides the octant k

 $3\pi$ 

- keep only the 3 least significant bits
- fraction part frac(K) is used to build

$$Y = \begin{cases} \operatorname{frac}(K) & \text{when } k \text{ is even} \\ 1 - \operatorname{frac}(K) & \text{when } k \text{ is odd} \end{cases}$$



• then  $\alpha = \frac{\pi}{4}Y$ 

#### Argument reduction for radian arguments

Goal: replace  $X \in \mathbb{R}$  with angle  $\alpha$  in  $\left[0, \frac{\pi}{4}\right]$ 

- compute  $K = X \times \frac{4}{\pi}$
- integer part int(K) provides the octant k

 $3\pi$ 

- keep only the 3 least significant bits
- fraction part frac(K) is used to build

$$Y = \begin{cases} \operatorname{frac}(K) & \text{when } k \text{ is even} \\ 1 - \operatorname{frac}(K) & \text{when } k \text{ is odd} \end{cases}$$



• then 
$$\alpha = \frac{\pi}{4}Y$$
  
• sin X and cos X rebuilt out of k, sin  $\left(\frac{\pi}{4}Y\right)$ , cos  $\left(\frac{\pi}{4}Y\right)$ 

•  $Y = \operatorname{frac}(K)$  may come very close to 0

#### •000 · · · 000 Y

- $Y = \operatorname{frac}(K)$  may come very close to 0
- Then,  $\sin\left(\frac{\pi}{4}Y\right) \approx \frac{\pi}{4}Y$  (Taylor) will also be very small

binary32	$x = 16367173 \cdot 2^{72}$	$\cos(x) < 2^{-29}$
binary32	$x = 6381956970095103 \cdot 2^{797}$	$\cos(x) < 2^{-61}$



- $Y = \operatorname{frac}(K)$  may come very close to 0
- Then,  $\sin\left(\frac{\pi}{4}Y\right) \approx \frac{\pi}{4}Y$  (Taylor) will also be very small

binary32	$x = 16367173 \cdot 2^{72}$	$\cos(x) < 2^{-29}$
binary32	$x = 6381956970095103 \cdot 2^{797}$	$\cos(x) < 2^{-61}$

 To compute the normalized mantissa of a floating-point sine, we need the p + g first significant bits.



- $Y = \operatorname{frac}(K)$  may come very close to 0
- Then,  $\sin\left(\frac{\pi}{4}Y\right) \approx \frac{\pi}{4}Y$  (Taylor) will also be very small

binary32	$x = 16367173 \cdot 2^{72}$	$\cos(x) < 2^{-29}$
binary32	$x = 6381956970095103 \cdot 2^{797}$	$\cos(x) < 2^{-61}$

- To compute the normalized mantissa of a floating-point sine, we need the p + g first significant bits.
- How many leading zeroes may appear?



- $Y = \operatorname{frac}(K)$  may come very close to 0
- Then,  $\sin\left(\frac{\pi}{4}Y\right) \approx \frac{\pi}{4}Y$  (Taylor) will also be very small

binary32	$x = 16367173 \cdot 2^{72}$	$\cos(x) < 2^{-29}$
binary32	$x = 6381956970095103 \cdot 2^{797}$	$\cos(x) < 2^{-61}$

- To compute the normalized mantissa of a floating-point sine, we need the p + g first significant bits.
- How many leading zeroes may appear?
  - Kahan/Douglas algorithm computes a bound  $g_K$

for a given floating-point format

- rule of thumb:  $g_K \approx p$  (the mantissa size)
- double-precision:  $g_K = 62$

$$\begin{array}{c|c} \leq g_{\mathcal{K}} & p+g \\ \bullet & 0 & 0 & \bullet & \bullet & 0 & 0 \\ \hline \end{array}$$

- $Y = \operatorname{frac}(K)$  may come very close to 0
- Then,  $\sin\left(\frac{\pi}{4}Y\right) \approx \frac{\pi}{4}Y$  (Taylor) will also be very small

binary32	$x = 16367173 \cdot 2^{72}$	$\cos(x) < 2^{-29}$
binary32	$x = 6381956970095103 \cdot 2^{797}$	$\cos(x) < 2^{-61}$

- To compute the normalized mantissa of a floating-point sine, we need the p + g first significant bits.
- How many leading zeroes may appear?
  - Kahan/Douglas algorithm computes a bound  $g_K$

for a given floating-point format

- rule of thumb:  $g_K \approx p$  (the mantissa size)
- double-precision:  $g_K = 62$
- therefore at least  $3 + p + g_K + g$  correct bits of the product

$$\mathcal{K} = \mathcal{X} imes rac{4}{\pi}$$
 need to be computed

$$K = \frac{4}{\pi} \times X$$

$$\boldsymbol{K} = \frac{4}{\pi} \times 2^{\boldsymbol{E}_{\boldsymbol{X}}} \times 1.\boldsymbol{F}_{\boldsymbol{X}}$$

$$K = \frac{4}{\pi} \times 2^{E_X} \times 1.F_X$$

$$K = \frac{4}{\pi} \times 2^{E_X} \times 1.F_X$$

- Very large multiplication
  - on one side,  $\frac{4}{\pi}$  shifted left by the exponent of X
    - (at most e<sub>max</sub> positions, where e<sub>max</sub> is the maximum exponent of the FP format)
  - on the other side, the *p*-bit mantissa of X



$$K = \frac{4}{\pi} \times 2^{E_X} \times 1.F_X$$

Very large multiplication

- on one side,  $\frac{4}{\pi}$  shifted left by the exponent of X
  - (at most e<sub>max</sub> positions, where e<sub>max</sub> is the maximum exponent of the FP format)
- on the other side, the *p*-bit mantissa of X
- Fortunately we need to compute only
  - 3 bits to the left of the point (the octant k)
  - $p + g_{K} + g$  bits to the right of the point (frac(K))



Florent de Dinechin, Arénaire Project, ENS-Lyon

Elementary functions III

$$K = \frac{4}{\pi} \times 2^{E_X} \times 1.F_X$$

Very large multiplication

- on one side,  $\frac{4}{\pi}$  shifted left by the exponent of X
  - (at most e<sub>max</sub> positions, where e<sub>max</sub> is the maximum exponent of the FP format)
- on the other side, the *p*-bit mantissa of X
- Fortunately we need to compute only
  - 3 bits to the left of the point (the octant k)
  - $p + g_{K} + g$  bits to the right of the point (frac(K))
- therefore the multiplication may be left-truncated



$$K = \frac{4}{\pi} \times 2^{E_X} \times 1.F_X$$

Very large multiplication

- on one side,  $\frac{4}{\pi}$  shifted left by the exponent of X
  - (at most e<sub>max</sub> positions, where e<sub>max</sub> is the maximum exponent of the FP format)
- on the other side, the *p*-bit mantissa of X
- Fortunately we need to compute only
  - 3 bits to the left of the point (the octant k)
  - $p + g_{\kappa} + g$  bits to the right of the point  $(frac(\kappa))$
- therefore the multiplication may be left-truncated
- Need to store  $e_{\max} + 3 + p + g_K + g$  bits of  $\frac{4}{\pi}$



#### The cost of argument reduction, summed up

Accurate argument reduction over the full floating-point range:

- extract  $\approx 3p$  bits out of  $\approx e_{\max} + 2p$  bits of  $\frac{4}{-1}$ 
  - single precision: 180 bits in, 75 bits out
  - double precision: 1144 bits in, 160 bits out
- a multiplication of  $\approx p \times 3p$  bits
  - single precision:  $24 \times 75$
  - $\bullet~$  double precision:  $53\times160$

• and a leading zero count on the result to remove cancellation

All this because 
$$\frac{4}{\pi}$$
 is irrational.

#### In practice

#### Make the common case fast!

A sequence of tests on x:

- No argument reduction at all for  $x < \pi/4$ .
- Use Cody and Waite argument reduction for small x
  - store  $\pi/4$  as a double-double  $C_h + C_l$
  - with as many as possible of the lower bits of  $C_h$  set to zero
  - (consider the worst case of cancellation so that p + g bits remain)
- Use the expensive Payne and Hanek only for large x
  - did I mention already that a program that computes the sine of a very large argument is probably doing something wrong anyway ?

Markstein suggests that, on Itanium, an efficient Payne and Hanek can make such (expensive) tests useless.

Best implemented by dividing sine by cosine, at least near the vertical asymptot.

- good polynomial approximations away from it
- but then code more complex
- and shared argument reduction anyway

Nice paper by Markstein (again):

Accelerating Sine and Cosine Evaluation with Compiler Assistance.

- Computing both sine and cosine of the same value is quite common (rotations, etc).
- Therefore, split each function into two functions: trigstart, which is shared, and sinfinishcosfinishtanfinish which performs the polynomial evaluation.
- Then let the compiler optimize share the argument reduction in this common case.

Is the previously reduced  $\alpha \in [0,\pi/4]$  still large for polynomial evaluation?

- need degree 12 polynomials for 60-bit accuracy
- but these polynomials can be odd

Use sin(a + y) = sin(a) cos(y) + cos(a) sin(y)

- either split again the previously reduced  $\alpha \in [0, \pi/4]$ :  $\alpha = a + y$  with a being the few leading bits of  $\alpha$
- or directly reduce to  $\alpha \in [0, \pi/512]$ , and define *a* as the fractional bits of *K*.
- In both cases, read tabulated values of sin(a) and cos(a)
- and "sum up cleverly"

### Logarithms

Trigonometric functions

#### Logarithms

Gal's accurate table method

Florent de Dinechin, Arénaire Project, ENS-Lyon Elementary functions III

#### Several functions

- log: log(x)
- log2: log<sub>2</sub>(x)
- log1p:  $\log(1 + x)$



#### First range reduction

First decompose x as

$$x = 2^{E'} \cdot m$$

with  $1 \le m < 2$  even if x is subnormal so

$$\log\left(x\right) = E' \cdot \log\left(2\right) + \log\left(m\right)$$

Using this term directly would lead to catastrophic cancellation in the case where E' = -1 and  $m \approx 2$ . To overcome this difficulty, a second adjustment is done as follows:

$$E = \begin{cases} E' & \text{if } m \le \sqrt{2} \\ E' + 1 & \text{if } m > \sqrt{2} \end{cases} \qquad y = \begin{cases} m & \text{if } m \le \sqrt{2} \\ \frac{m}{2} & \text{if } m > \sqrt{2} \end{cases}$$

for some value of  $\sqrt{2}$  (can be grossly approximate, we use  $\sqrt{2}\approx 1.5$  in hardware)

Florent de Dinechin, Arénaire Project, ENS-Lyon

Elementary functions III

$$\log\left(x\right) = E \cdot \log\left(2\right) + \log\left(y\right)$$

and y is in an interval of size  $\approx$  1, still large for polynomial approximation

- use the k high order bits of y as an index i,
- read from a table  $r_i \approx 1/y$
- define the remainder  $z = y \cdot r_i 1$
- z can be computed exactly as a doubled-FP
  - actually it almost fits a double if  $r_i$  has a k + 1-bit mantissa
- SO

$$\log\left(y\right) = \log\left(1+z\right) - \log\left(r_i\right)$$

- Also tabulate the  $log(r_i)$  as double-doubles
- Now z is small: evaluate  $\log(1 + z)$  thanks to a polynomial approximation

Previous reduction: first appears in 1990 a TOMS paper by Gal and Bachelis, popularized by Intel papers about Itanium (using frcpa instruction).

Before that,

- Article by Tang in 1990 in Transactions on Mathematical Software
  - but involves a division
- I don't remember the one in Markstein's book
  - but Markstein agreed later that Intel's was better

### Gal's accurate table method

Trigonometric functions

Logarithms

Gal's accurate table method

Florent de Dinechin, Arénaire Project, ENS-Lyon Elementary functions III

#### In one slide

We just tabulated pairs  $(r_i, \log(r_i))$ 

- we have a certain freedom of choice for  $r_i \approx 1/i$
- (*i* is itself a gross approximation to y)
- we may as well look for a r<sub>i</sub> such that log(r<sub>i</sub>) has 8 zeroes after its 53rd bit of mantissa
- (statistically, one of the 256 neighbours of  $\circ(1/i)$  has this property)
- Then,  $\log(r_i)$ , tabulated as a double, is actually accurate to  $2^{-60}$

- it may be worth to tabulate (δx<sub>i</sub>, y<sub>i</sub>) with y<sub>i</sub> accurate instead of a double-binary64 (y<sub>i</sub><sup>h</sup>, y<sub>i</sub><sup>l</sup>)
  - same memory consumption,
  - but avoids double-binary64 computing
- In CRLibm, we use it to tabulate double-binary64 accurate to  $2^{-120}$ .

Θ ...