EE 486 lecture 18: The Higher Level Functions (HLFs).

M. J. Flynn

The HLF: what are they?

- Also called the Elementary Functions
- Originally the familiar calculator type functions, trig, log, exponential, n-th root, etc.
- With the advent of DSP, most any analytic function representable as a series, a polynomial or a ratio of polynomials. E.g. the incomplete gamma function.

HLF implementations

- Usually a combination of hardware (esp. for DSP) and software.
- Tradeoffs are precision, time and hardware support cost.
- Three types of approaches
  - Tables (interpolation or series based)
  - Generalized (polynomial, rational or continued fractions)
  - Hardware (CORDIC or multiplier PPAs)

Tables

- As with divide, we can
  - Use a Taylor series expansion about X_h, using multiple tables for the terms in the series.
  - Use a form of interpolation based on one or two tables.
  - For reasonable table sizes, either approach is limited to finding f(X) to about 20b of precision.

Polynomial approximations

- Scale the domain to improve precision.
- Selected coefficients to reduce either maximum error or rms error.
- Apply Horner's rule
  - $F(x)= c_n x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$
  - $\quad = (c_n x + c_{n-1})x + \ldots + c_1)x + c_0$
- m th degree polynomial requires m multiply adds.

Polynomial approximations

- Many different polynomial forms are possible:
  - Taylor series (not the best error term)
  - Chebyshev Polynomial (best max error in [-1,1))
  - Interpolation: n point fit to n th (or lower) degree polynomial. Tradeoff between order (number of multiplies) and points (storage of constants or domain points).

Polynomial approximations

- For common elementary functions the IEEE single precision result requires 3-5 terms; double precision requires 6-10 terms. Less if a table is used to start.
- This are some pre scaling overhead operations, not included.

Rational approximations

- Take the ratio of two polynomials, both of lower degree than required of a polynomial approximation for the same precision.
- $R_{m,n} = P_m(x)/ Q_n(x)$, where m,n are the degrees of the polynomials.
- $R_{m,0} = P_m(x)$, the polynomial approximation.
- Select coefficients and order to give the minimum maximum error.

Numerical coprocessor support

- Hardware supported rational approximation really supports the word size.
- Target mantissa is 67 bits, so as to support the 80b IEEE extended precision.
- Such coprocessors are unusual.

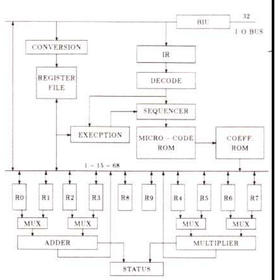


Fig. 1. Block diagram of the numerical coprocessor

Pre scaling of operands

- To improve the precision it's important to scale the operands into a limited range.

1) For sin, cos, and tan, $[a, b] = \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$.
2) For ln, $[a, b] = \left[\frac{1}{\sqrt{2}}, \sqrt{2}\right]$.
3) For $\sin^{-1}$, $\cos^{-1}$, $[a, b] = \left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$.
4) For $\tan^{-1}$, $[a, b] = [-1, 1]$.
5) For 2^x, $[a, b] = \left[0, \frac{1}{2}\right]$.

Rational approximations (RA)

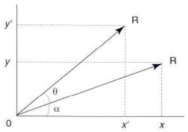
- To reduce the computational order many approximations are of the form
- $P_{m,n} = x\, P_m(x^2)/ Q_n(x^2)$,
- Typically $P_{4,4}$ is sufficient to give 67b precision, altho the arc functions require $P_{7,7}$
- The RA is computationally superior for double or extended precision, for single precision or less CORDIC is preferable.

CORDIC

- A bit by bit algorithm that can compute all the trig functions with a small amount of hardware support.
- Basic algorithm gives sine, cosine and tan; generalization gives arc and hyperbolic trig.
- Developed in '59 by J. Volder
- Broadly used in calculators and sometimes in processors.

CORDIC

- $x' = R\cos(\alpha+\theta)$
- $y' = R\sin(\alpha+\theta)$
- R is unit length vector.
- By a series of rotations by α_i make $\Sigma\alpha_i = \theta$ and then determine x and y and hence the trig functions.



Given θ (in radians) find sin θ, cos θ, tan θ, etc

CORDIC

- Expand $x' = R\cos(\alpha+\theta)$; $y' = R\sin(\alpha+\theta)$
- $x' = R[\cos(\theta)\cos(\alpha) - \sin(\theta)\sin(\alpha)]$
- $y' = R[\sin(\theta)\cos(\alpha) + \cos(\theta)\sin(\alpha)]$
- $x' = (R\cos(\alpha))\cos(\theta) - y\sin(\theta)$
- Since $x = R\cos(\alpha)$. And divide by $\cos(\theta)$
- $x'/\cos(\theta) = x - y\tan(\theta)$, similarly
- $y'/\cos(\theta) = y + x\tan(\theta)$,

CORDIC

- $\tan(\theta) \sim \theta = \alpha_0 +/- \alpha_1 +/- \alpha_2 +/- \ldots$ where $\alpha_t = \tan^{-1} 2^{-i}$ for α_t, choose + or – so as to bring $(\theta +/- \Sigma\, \alpha_j) = z_i$ to zero,
- Iterations are $x_{i+1} = x_i -/+ (y_i\, 2^{-i})$, similarly
- $y_{i+1} = y_i +/- (x_i\, 2^{-i})$,
- $z_{i+1} = z_i -/+ (2^{-i})$, select sign so that $z_m = 0$

CORDIC

- Rotations by a are actually pseudo rotations since for each iteration R is being lengthened by $1/\cos(\alpha)$. But since $\cos(\alpha) = \cos(-\alpha)$, we can pre compute the product of this lengthening as K= 1.646760…

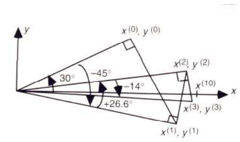
Converting $\tan(2^{-i})$ into degrees

iteration	0	1	2	3	4	5	6	7	8	9
degrees	45	26 .6	14	7.1	3.6	1.8	0.9	0.4	0.2	0.1

An example, $\theta = 30^0$

TABLE 22.2
Choosing the signs of the rotation angles to force z to 0

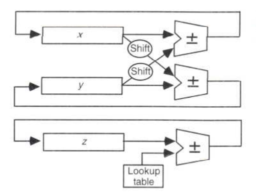
i	$z^{(i)}$	$-$	$\alpha^{(i)}$	$=$	$z^{(i+1)}$
0	+30.0	−	45.0	=	−15.0
1	−15.0	+	26.6	=	+11.6
2	+11.6	−	14.0	=	−2.4
3	−2.4	+	7.1	=	+4.7
4	+4.7	−	3.6	=	+1.1
5	+1.1	−	1.8	=	−0.7
6	−0.7	+	0.9	=	+0.2
7	+0.2	−	0.4	=	−0.2
8	−0.2	+	0.2	=	+0.0
9	+0.0	−	0.1	=	−0.1



CORDIC

- Produces all trig functions at a bit of precision per iteration
- Each iteration takes 3 adds and a compare.

CORDIC hardware support



Using PPA arrays for function evaluation (Schwartz)

- If a multiplier provides a solution to AxB=C by adding pp's then maybe we can use something similar to back solve A op Q =B (for divide) or S op S = A for sqrt.
- Solve any problem expressible as a multiplication (even the trig functions)
- Goal: use a FP multiplier's pp array to give an HLF approximation in one multiply time.

An example of deriving a PPA approximation for the reciprocal function is shown. First, the PPA of a multiplication ($B * Q = 0.11\cdots$) is back-solved in terms of the multiplier (Q).

$$
\begin{array}{c|c|c|c|c|c|ccccc||c}
 & & & & 0. & 1 & b_2 & b_3 & b_4 & b_5\cdots & =B \\
 & & & X & q_0. & q_1 & q_2 & q_3 & q_4 & \cdots & =Q \\
\hline
 & & & & & & \vdots & \vdots & \vdots & \vdots\cdots & \\
 & & & & & q_4 & b_2q_4 & b_3q_4 & b_4q_4 & b_5q_4\cdots & \\
 & & & & q_3 & b_2q_3 & b_3q_3 & b_4q_3 & b_5q_3 & \cdots & \\
 & & & q_2 & b_2q_2 & b_3q_2 & b_4q_2 & b_5q_2 & \cdots & & \\
 & & q_1 & b_2q_1 & b_3q_1 & b_4q_1 & b_5q_1 & \cdots & & & \\
 & q_0 & b_2q_0 & b_3q_0 & b_4q_0 & b_5q_0 & \cdots & & & & \\
\hline
0. & 1 & 1 & 1 & 1 & 1 & \cdots & & & & \approx 1.0
\end{array}
$$

By choosing the quotient digits appropriately in a redundant notation, each column of the PPA forms an independent equation.

$$
\begin{aligned}
q_0 &= 1 \\
q_1 + b_2q_0 &= 1 \\
q_2 + b_2q_1 + b_3q_0 &= 1 \\
q_3 + b_2q_2 + b_3q_1 + b_4q_0 &= 1 \\
q_4 + b_2q_3 + b_3q_2 + b_4q_1 + b_5q_0 &= 1.
\end{aligned}
$$

PP arrays

- Back solving the reciprocal's equations involves both positive and negative terms.
- In the place of the (3,2) counter we can use a more general structure called a Petritz counter, where each of the 3 inputs can be either positive or negative.
- But this requires special hardware for each function.

1. Back solve

These boolean equations are solved to yield the following algebraic equations for five digits of the quotient:

$$
\begin{aligned}
q_0 &= 1 \\
q_1 &= 1 - b_2 \\
q_2 &= 1 - b_3 \\
q_3 &= 1 - b_2 + 2b_2b_3 - b_3 - b_4 \\
q_4 &= 1 - b_2b_3 - b_4 + 2b_2b_4 - b_5.
\end{aligned}
$$

2. Put in PPA form

These equations are used to form the PPA of the approximation.

q_0	q_1	q_2	q_3	q_4
			$-b_4$	$-b_5$
			$-b_3$	$2b_2b_4$
			$2b_2b_3$	$-b_4$
	$-b_2$	$-b_3$	$-b_2$	$-b_2b_3$
1	1	1	1	1

Reduction rules for all positive pp's

3. Reduction Rules

1. Algebraic Expansion: $M * a \Rightarrow \sum k_i 2^{-i} * a$, where M is any real number, a and k_i are Boolean, and 2^{-i} is implied by the column weight. This step expands all coefficients in the array into their binary components. An example is:

$$5a \;\Rightarrow\; 4a + a \;\Rightarrow\; (a,\ 0,\ a)\ \textit{by columns}$$

2. Algebraic Reduction: $2a - a \Rightarrow a$,
3. Boolean Reduction: $a + b - ab \Rightarrow a|b$
   or $a + \overline{a}b \Rightarrow a|b$,
4. Boolean Reduction: $a - ab \Rightarrow a(1-b) \Rightarrow a\overline{b}$,
5. Boolean Reduction: $a + b - 2ab \Rightarrow a \oplus b$;

HLF using PPA's

3. Apply reduction rules

Reduced PPA

q_0	q_1	q_2	q_3	q_4
				$\overline{b_5}$
			$-(b_2\|b_4)$	$-b_4$
1	$\overline{b_2}$	$(b_2\|\overline{b_3})$	$\overline{b_3}$	$-b_2b_3$

4. Error Compensate to Improve Accuracy

q_0	q_1	q_2	q_3	q_4	q_5
				$\overline{b_5}$	
				$\boxed{\overline{b_2}b_3b_4}$	
			$-(b_2\|b_4)$	$-b_4$	
1	$\overline{b_2}$	$(b_2\|\overline{b_3})$	$\overline{b_3}$	$-b_2b_3$	$\boxed{-\overline{b_2}\,\overline{b_3}b_5}$

HLFs using PPA's

5. Complement Negative Elements and Subtract One

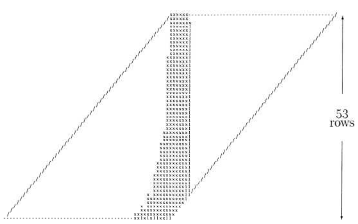
	q_0	q_1	q_2	q_3	q_4	q_5
Before :					$\overline{b_5}$	
					$\overline{b_2}b_3b_4$	
				$-(b_2\|b_4)$	$-b_4$	
	1	$\overline{b_2}$	$(b_2\|\overline{b_3})$	$\overline{b_3}$	$-b_2b_3$	$-\overline{b_2}\,\overline{b_3}b_5$
After :					$\overline{b_5}$	
					$\overline{b_2}b_3b_4$	
				$\overline{b_2}\,\overline{b_4}$	$\overline{b_4}$	
	1	$\overline{b_2}$	$(b_2\|\overline{b_3})$	$\overline{b_3}$	$(\overline{b_2}\|\overline{b_3})$	$(b_2\|b_3\|\overline{b_5})$
				-1	-2	-1

HLFs using PPA's

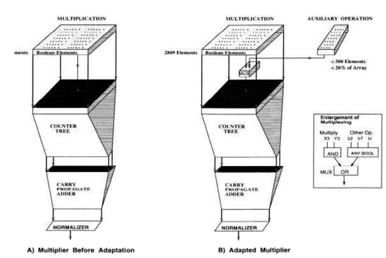
6. Reduce Constants

q_0	q_1	q_2	q_3	q_4	q_5
				1	
				$\overline{b_5}$	
			1	$\overline{b_2}b_3b_4$	
	1		$\overline{b_2}\,\overline{b_4}$	$\overline{b_4}$	1
	$\overline{b_2}$	$(b_2\|\overline{b_3})$	$\overline{b_3}$	$(\overline{b_2}\|\overline{b_3})$	$(b_2\|b_3\|\overline{b_5})$

A 53 pp array provides a 17b reciprocal



Modifying the multiplier for HLFs



Other HLFs

For functions such as

$$\begin{aligned} q &= a/b & (qb = a) \\ r &= 1/b & (r \cdot b = 1) \\ s &= \sqrt{z} & (x \cdot x = z) \end{aligned}$$

the pp equations are straightforward.

In extending this technique to functions such as log, exponent, and the trig functions, the rearrangement of the function into multiplicative form is not obvious. It usually involves two substeps:

1. Expressing the operands as polynomials (or other suitable analytic functions)
2. Differentiating the function.

Other HLFs

$$W(x) = f(V(x))$$

is

$$1.x_1x_2x_3\ldots = f(1.v_1v_2v_3\ldots)$$

W, V are polynomials with (0,1) coefficient.
If f is expressable as a series expansion, then $f(V(x))$ is also expressable as a series.
Collect terms in v_i to solve for x_j.

(2.)
Some transcendental functions have analytic derivatives:

$$W(x) = \ln(V(x))$$

$$W(x)' \cdot V(x) = V(x)'$$

$$\frac{d(\sin^{-1}(x))}{dx} = \frac{1}{\sqrt{1-x^2}}$$

Use identities to find $\sin x$, e^x.

PPA conclusions

1. Almost all of the known elementary functions can be computed (usually to within 12–20 bits of precision) in the same time it takes to do one multiplication.
2. The hardware cost of this adaptation consists merely of a few multiplex and logic gates which are input to certain of the partial product array rows.
3. The resulting adaptive multiplier/function unit can be dynamically reconfigured in less than one cycle to perform any of the basic math functions.