EE 486 lecture 17: What's new in Divide.

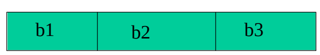
M. J. Flynn

Two approaches

- Bipartite tables ….very useful in short precision divide, as with 3D graphics.
- Higher order series … for long precision and extended precision.
- Note that both of these approaches are also useful in implementing the various HLF (higher level functions: trig, log, sqrt, etc)

Bipartite tables

- Implement first 2 terms of Taylor series for 1/b in 2 tables.
- First term is an approximation, the second term approximates the derivative, (1/b)'
- Then b

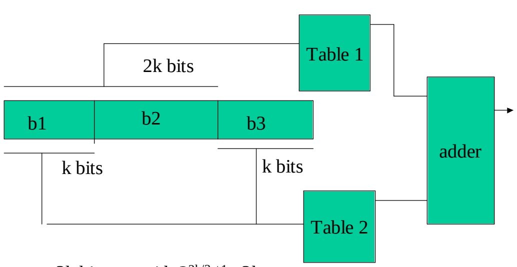


- First table index is b1+b2; second table index is b1+b3 (b3 defines the derivative in the region of b1).

Bipartite Tables to find (1/b)

- Based on first two terms of a Taylor series expanded about the leading bits of b, called b_h. So
- Reciprocal $=(1/b_h) - \Delta b(1/b_h)^2 + (\Delta b)^2(1/b_h)^3$ –note that all terms are positive since Δb is negative.
- Use two tables, one to find the first term and one to find the second… error is approx. by the third term.

Bipartite Tables to find (1/b)



3k bits out with $2^{2k/3+1}$ x3k

Interpolation tables

- Similar approach is to use linear (or higher order interpolation.
- Reciprocal $= (1/b_h) + b_l[(1/b_h)-(1/b_h+ulp)]$
- Now needs one table lookup then a multiply –add.

Interpolation tables

- Can be a more general approach using a multiply and an add.
- Needs a smaller table $2^{k/2}$ x (2k +3 or so).

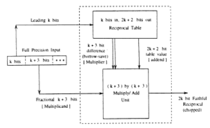


Figure 1. A 2k+3-bits-in 2k-bits-out faithful interpolated reciprocal

Higher order divide (a/b)

- As with the NR on the term exam, we can use multiple terms (say t terms) of the Taylor series as an iteration. So
- Reciprocal $=(1/b_h) - \Delta b(1/b_h)^2 + (\Delta b)^2(1/b_h)^3$
- $\Delta b = |b - b_h| = b_l$, all terms positive
- So look up $(1/b_h)$, $(1/b_h)^2$, $(1/b_h)^3$; compute Δb and $(\Delta b)^2$

Higher order divide: #1

- Now compute new dividend, a' as
- $a' = a - a_h \times (1/b_h) \times b$ and quotient
- $q' = q + a_h' \times (1/b_h)$ (shifted)
- Can use redundant, s +c form to speed things up.
- Precision (m bit lookup) m-2 bits per iteration

Higher order divide: #2

- $B = (1/b_h) - \Delta b(1/b_h)^2 + (\Delta b)^2(1/b_h)^3 \ldots$;t terms
- Look up m bits of $(1/b_h)$, $(1/b_h)^2$, $(1/b_h)^3$
- Now compute new dividend, a' as
- $a' = a - a_h \times B \times b$ and quotient
- $q' = q + a_h' \times B$ (shifted)
- Precision (m bit lookup) mt - t-1 bits per iteration

Higher order divide: #3

- Let $b = b_H + b_L$
- Factor $1/ b_H - b_L / b_H^2 + (b_L)^2(1/ b_H)^3 \ldots$;
- $a/(b_H + b_L) = a/ b_H (1- b_L / b_H + b_L^2 / b_H^2)$
- First 2 terms $a/b = a (b_H - b_L) / b_H^2$
- Look up b_H^2
- Precision (m bit lookup) 2m - 3 bits per iteration… can be 2m with compensation

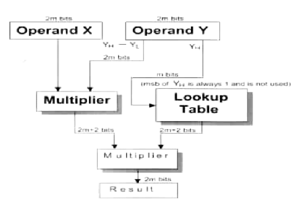


Figure 8.1 Algorithm C

Liddicoat's General Purpose Divide and Elementary Function(HLF) Unit

- Higher order series expansion can be used for really high-performance (low latency) divide and HLF units.
- Up till now we mostly used 1st-order iteration with quadratic convergence.
- Higher-order iterations converge more rapidly BUT have hardware requirements.
- The parallel computation of the square, cube, and powers of an operand reduce the latency of the higher-order iteration.

Reciprocal and the Elementary Functions Represented by Taylor Series Expansions

$$1/b = 1 - x + x^2 - x^3 + x^4 - ...$$

$$\sqrt{b} = 1 + {}^1/_2\,x - {}^1/_8\,x^2 + {}^1/_{16}\,x^3 - {}^{15}/_{128}\,x^4 + ...$$

$$1/\sqrt{b} = 1 - {}^1/_2\,x + {}^3/_8\,x^2 - {}^5/_{16}\,x^3 + {}^{35}/_{128}\,x^4 - ...$$

$$e^x = 1 + x + {}^1/_2\,x^2 + {}^1/_6\,x^3 + {}^1/_{24}\,x^4 + ...$$

$$\ln(x+1) = x - {}^1/_2\,x^2 + {}^1/_3\,x^3 - {}^1/_4\,x^4 + ...$$

$$\cos(x) = 1 - {}^1/_2\,x^2 + {}^1/_{24}\,x^4 - ...$$

$$\sin(x) = x - {}^1/_6\,x^3 + {}^1/_{120}\,x^5 - ...$$

$$\arctan(x) = x - {}^1/_3\,x^3 + {}^1/_5\,x^5 - ...$$

Reciprocal, Square Root, and Inverse Square Root as Series Expansion

Prescaled by $d=(1-bX_0)$ with $X_0 \approx 1/b$, $Y_0 \approx 1/\sqrt{b}$, and $Z_0 \approx \sqrt{b}$

- **Reciprocal**

$$\boxed{1/b = X_0(1 + d + d^2 + d^3 + d^4 + ...)}$$

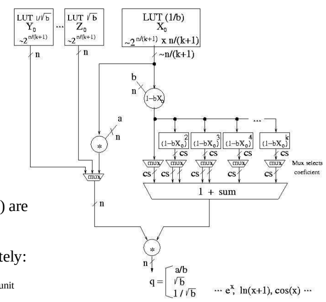
- **Square Root**

$$\boxed{\sqrt{b} = Y_0(1 - 1/2d - 1/8d^2 - 1/16d^3 - 15/128d^4 - ...)}$$

- **Inverse Square Root**

$$\boxed{1/\sqrt{b} = Z_0(1 + 1/2\,d + 3/8\,d^2 + 5/16\,d^3 + 35/128\,d^4 + ...)}$$

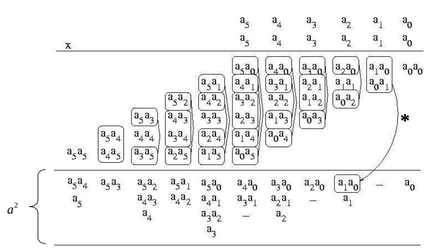
Architecture for the General Purpose Arithmetic Unit



- The powers of $(1-bX_0)$ are computed in parallel.
- Latency is approximately:

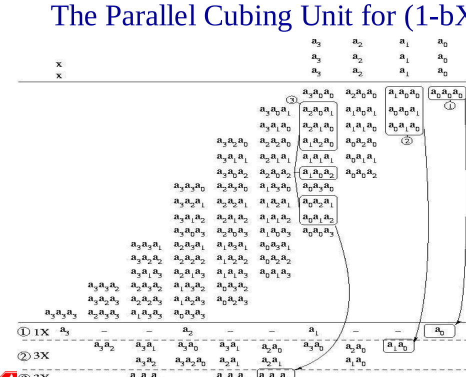
$t = t_{\text{look up table}} + 3\,t_{\text{sub-unit}}$

The Parallel Squaring Unit for $(1-bX_0)^2$

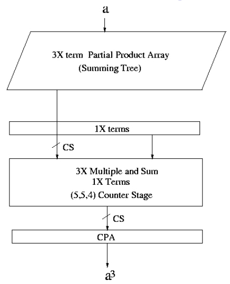


$* \; a_i a_j + a_j a_i = 2a_i a_j$

The Parallel Cubing Unit for $(1-bX_0)^3$

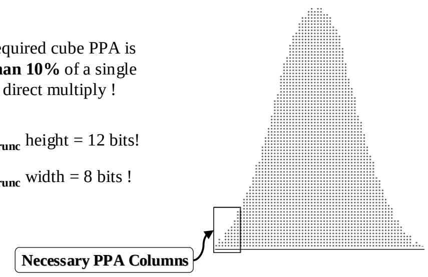


Hardware Structure for the Parallel Cubing Unit



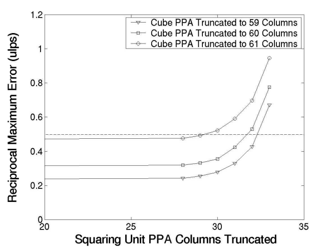
Truncated PPA for 24-bit Cube

- The required cube PPA is **less than 10%** of a single 24-bit direct multiply !
- PPA $_{trunc}$ height = 12 bits!
- PPA $_{trunc}$ width = 8 bits !



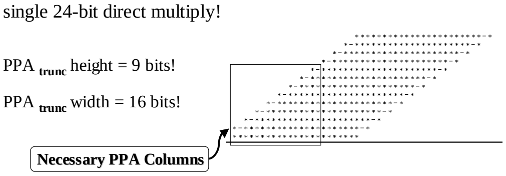
Square PPA Column Truncation

- The divide unit was simulated for various squaring and cubing unit truncations.
- There is a knee in the curve when the squaring unit is truncated by **29** columns.
- **E < 0.5 ulp**
  with the cube PPA_{trunc} = 60 and square PPA_{trunc} = 31.



Truncated Square PPA

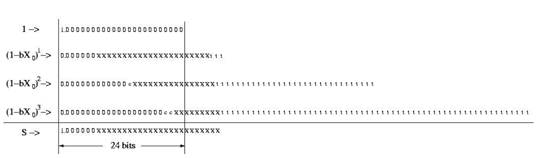
- The required squaring unit PPA is **less than 15%** of a single 24-bit direct multiply!
- PPA $_{trunc}$ height = 9 bits!
- PPA $_{trunc}$ width = 16 bits!



Final Divide Sub-unit Precision

3rd Order Reciprocal Approximation

$$1/b = X_0 * \underbrace{(1 + (1-bX_0)^1 + (1-bX_0)^2 + (1-bX_0)^3)}_{S}$$



Divide and HLF: net

- Can be done in a LUT + (1-2) MPY+ADD.
- Yes, a 4 cycle divide is possible.
- And the hardware cost is probably no more than two multipliers and an 3 way adder and (of course) a LUT.