Speeding Up Floating-Point Division With In-lined Iterative Algorithms

Robert Enenkel, Allan Martin

IBM® Toronto Lab

Outline

- Hardware floating-point division
- The case for software division
- Software division algorithms
- Special cases/tradeoffs
- Performance results
- Automatic generation

Hardware Division

- PPC fdiv, fdivs
- Advantages
  - accurate (correctly rounded)
  - handles exceptional cases (Inf, NaN)
  - lower latency than SW
- Disadvantages
  - occupies FPU completely
  - inhibits parallelism

Alternatives to HW division

- Vector libraries
  - MASS
  - higher overhead, greater speedup
- In-lined software division
  - low overhead, medium speedup

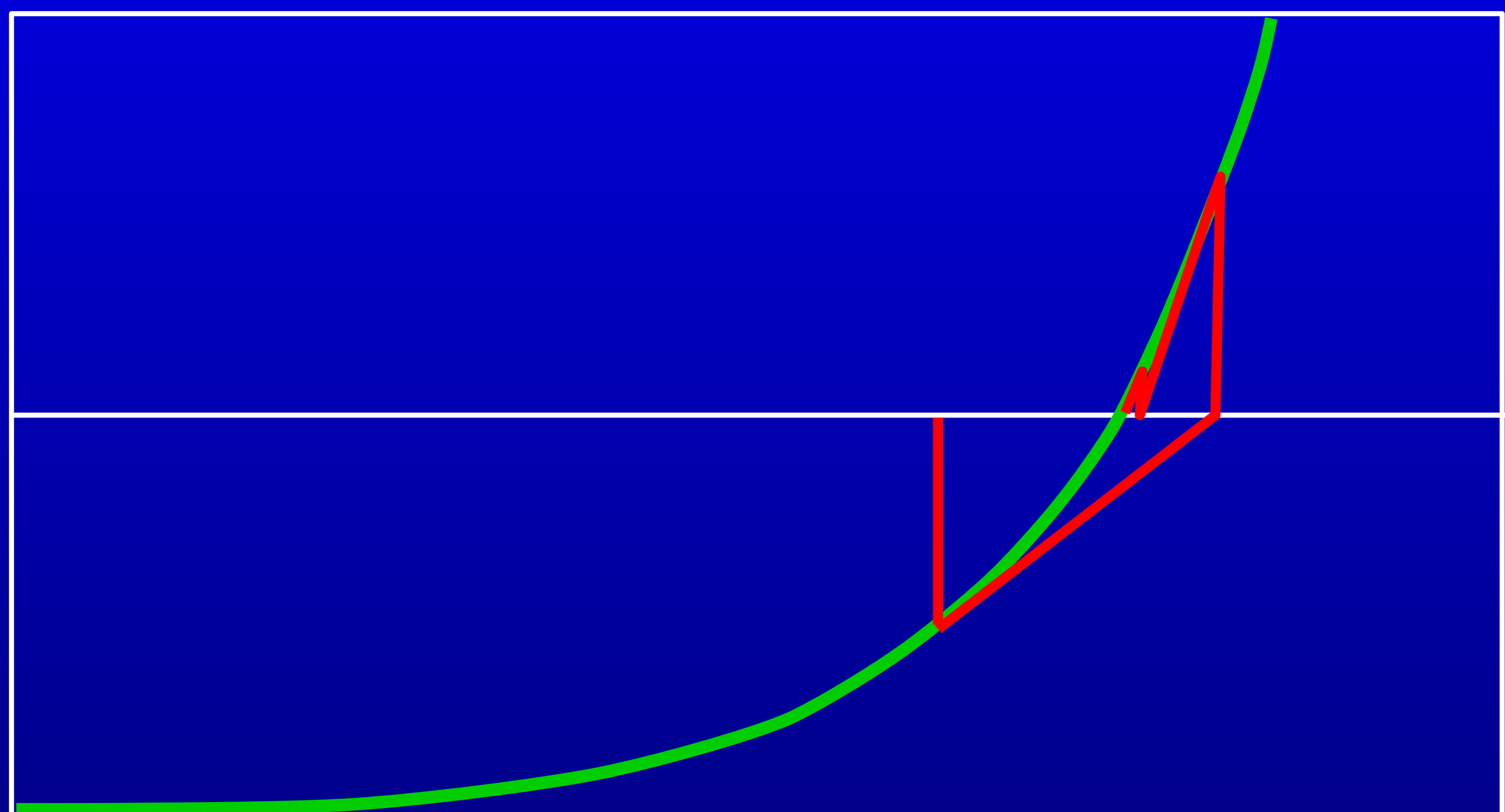
Rationale for Software Division

- Write SW division algorithm in terms of HW arithmetic instructions
  - Newton's method or Taylor series
- Latency will be higher than HW division
- But...SW instructions can be interleaved, so throughput may be better
- Requires enough independent instructions to interleave
  - loop of divisions
  - other work

Newton's Method

- To find x such that $f(x) = 0$,
- Initial guess x_0
- $x_{n+1} = x_n - f(x_n)/f'(x_n)$, $n=0, 1, 2,...$
- Provided x_0 is close enough
  - x_n converges to x
  - It converges quadratically $|x_{n+1}-x| < c|x_n-x|^2$
  - Number of bits of accuracy doubles with each iteration

Newton's Method



Newton Iteration for Division

- For 1/b, let $f(x) = 1/x - b$
- For a/b, use $a*(1/b)$ or $f(x) = a/x - b$
- Algorithm for 1/b
  - $x_0 \sim 1/b$ initial guess
  - $e_0 = 1 - b*y_0$
  - $x_1 = x_0 + e_0*x_0$
  - $e_1 = e_0*e_0$
  - $x_2 = x_1 + e_1*x_1$
  - etc...

How Many Iterations Needed?

- **Power5 reciprocal estimate instructions**
  - FRES (single precision), FRE (double prec.)
  - |relative error| <= 2^(-8)
- **Floating-point precision**
  - single: 24 bits
  - double: 53 bits
- **Newton iterations**
  - error: 2^(-16), 2^(-32), 2^(-64), 2^(-128)
  - single: 2 iterations for 1 ulp
  - double: 3 iterations for 1 ulp
  - +1 iteration for correct rounding (0.5 ulps)

Taylor Series for Reciprocal

- $x_0 \sim 1/b$ initial guess
- $e = 1 - b\, x_0$
- $1/b = x_0/(b\, x_0) = x_0\, (1/(1-e))$

  $= x_0\, (1 + e + e^2 + e^3 + e^4 + \ldots)$
- Algorithm (6 terms)
  - $e = 1 - d*x_0$
  - $t_1 = 0.5 + e * e$
  - $q_1 = x_0 + x_0 * e$
  - $t_2 = 0.75 + t_1*t_1$
  - $t_3 = q_1*e$
  - $q_2 = x_0 + t_2*t_3$

Speed/Accuracy tradeoff

- IBM compilers have -qstrict/-qnostrict
- -qstrict: SW result should match HW division exactly
- -qnostrict: SW result may be slightly less accurate for speed

Exceptions

- Even when a/b is representable...
- 1/b may underflow
  - a ~ b ~ huge, a/b ~ 1, 1/b denormalized
  - Causes loss of accuracy
- 1/b may overflow
  - a, b denormalized, a/b ~ 1, 1/b = Inf
  - Causes SW algorithm to produce NaN
- Handle with tests in algorithm
  - Use HW divide for exceptional cases

Algorithm variations

- User callable built-in functions
  - swdiv(a,b): double precision, checking
  - swdivs(a,b): single precision, checking
  - swdiv_nochk(a,b): double, non-checking
  - swdivs_nochk(a,b): single, non-checking
- Accuracy of swdiv, swdiv_nochk depends on -qstrict/-qnostrict
- _nochk versions faster but have argument restrictions

Accuracy and Performance

	Power5 speedup ratio	Power4 speedup ratio	Power5 ulps max error	Power4 ulps max error
swdivs	1.07	1.05	0.5	0.5
swdivs_nochk	1.46	1.28	0.5	0.5
swdiv strict	1.05		0.5	
swdiv nostrict	1.50		1.5	
swdiv_nochk strict	1.51		0.5	
swdiv_nochk nostrict	1.77		1.5	

Automatic Generation of Software Division

- The swdivs and swdiv algorithms can also be automatically generated by the compiler
- Compiler can detect situations where throughput is more important than latency

Automatic Generation of Software Division

- In straight-line code, we use a heuristic that calculates how much FP can be executed in parallel
  - independent instructions are good, especially other divides
  - dependent instructions are bad (they increase latency)

Automatic Generation of Software Division

- In modulo scheduled loops software-divide code can be pipelined, interleaving multiple iterations
- Divides are expanded if divide does not appear in a recurrence (cyclic data-dependence)

Summary

- Software divide algorithms
  - ƒ user callable
  - ƒ compiler generated
- Loops of divides
  - ƒ up to 1.77x speedup
- UMT2K benchmark
  - ƒ 1.19x speedup