Formal verification of IA-64 division algorithms

John Harrison

Intel Corporation

- IA-64 overview
- HOL Light overview
- IEEE correctness
- Division on IA-64
- Theory of division algorithms
- Improved theorems and faster algorithm
- Conclusions

IA-64 overview

IA-64 is a new 64-bit computer architecture jointly developed by Hewlett-Packard and Intel, and the Itanium^{TM} chip from Intel will be its first silicon implementation. Among the special features of IA-64 are:

- An instruction format encoding parallelism explicitly
- Instruction predication
- Speculative and advanced loads
- Upward compatibility with IA-32 (x86).

The IA-64 Applications Developer's Architecture Guide is now available from Intel in printed form and online:

`http://developer.intel.com/design/ia64/downloads/adag.htm`

Quick introduction to HOL Light

HOL Light is a member of the family of HOL theorem provers.

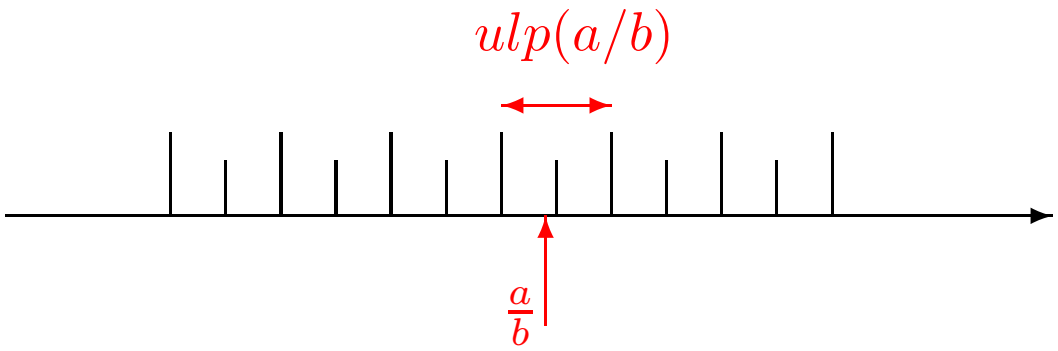
- An LCF-style programmable proof checker written in CAML Light, which also serves as the interaction language.
- Supports classical higher order logic based on polymorphic simply typed lambda-calculus.
- Extremely simple logical core: 10 basic logical inference rules plus 2 definition mechanisms.
- More powerful proof procedures programmed on top, inheriting their reliability from the logical core. Fully programmable by the user.
- Well-developed mathematical theories including basic real analysis.

HOL Light is available for download from:

`http://www.cl.cam.ac.uk/users/jrh/hol-light`

IEEE correctness

The IEEE standard states that all the algebraic operations, including division, should give the closest floating point number to the true answer, or the closest number up, down, or towards zero in other rounding modes.



In addition, all the flags need to be set correctly, e.g. inexact, underflow, ...

IA-64 features an IEEE-correct fused multiple add, which can compute $xy + z$ with a single rounding error. However it has no instruction for division.

Division on IA-64

Instead, approximation instructions are provided, e.g. the floating point reciprocal approximation instruction.

$$\texttt{frcpa}.sf\ f_1, p_2 = f_3$$

In normal cases, this returns in f_1 an approximation to $\frac{1}{f_3}$. The approximation has a worst-case relative error of about $2^{-8.86}$. The particular approximation is specified in the IA-64 architecture.

Software is intended to start from this approximation and refine it to an IEEE-correct quotient. Surprisingly, quite short sequences of straight-line code suffice to do so. We will concentrate on round-to-nearest mode, since the other modes are much easier.

Markstein's main theorem

Markstein (IBM Journal of Research and Development, vol. 34, 1990) proves the following general theorem. Suppose we have a quotient approximation $q_0 \approx \frac{a}{b}$ and a reciprocal approximation $y_0 \approx \frac{1}{b}$. Provided:

- The approximation q_0 is within 1 ulp of $\frac{a}{b}$.
- The reciprocal approximation y_0 is $\frac{1}{b}$ rounded to the nearest floating point number

then if we execute the following two `fma` (fused multiply add) operations:

$$\begin{aligned} r &= a - bq_0 \\ q &= q_0 + ry_0 \end{aligned}$$

the value r is calculated exactly and q is the correctly rounded quotient, whatever the current rounding mode.

Markstein's reciprocal theorem

The problem is that we need a perfectly rounded y_0 first, for which Markstein proves the following variant theorem.

If y_0 is within $1ulp$ of the exact $\frac{1}{b}$, then if we execute the following `fma` operations in round-to-nearest mode:

$$\begin{aligned} e &= 1 - by_0 \\ y &= y_0 + ey_0 \end{aligned}$$

then e is calculated exactly and y is the correctly rounded reciprocal, *except possibly when the mantissa of b is all 1s*.

Using the theorems

Using these two theorems together, we can obtain an IEEE-correct division algorithm as follows:

- Calculate approximations y_0 and q_0 accurate to 1 ulp (straightforward). [N `fma` latencies]
- Refine y_0 to a perfectly rounded y_1 by two `fma` operations, and in parallel calculate the remainder $r = a - bq_0$. [2 `fma` latencies]
- Obtain the final quotient by $q = q_0 + ry_0$. [1 `fma` latency].

There remains the task of ensuring that the algorithm works correctly in the special case where b has a mantissa consisting of all 1s.

One can prove this simply by testing whether the final quotient is in fact perfectly rounded. If it isn't, one needs a slightly more complicated proof. Markstein shows that things will still work provided q_0 *overestimates* the true quotient.

Initial algorithm example

Our example is an algorithm for quotients using only single precision computations (hence suitable for SIMD). It is built using the `frcpa` instruction and the (negated) `fma` (fused-multiply-add):

1. $y_0 = \frac{1}{b}(1+\epsilon)$ [`frcpa`]
2. $e_0 = 1 - by_0$
3. $y_1 = y_0 + e_0 y_0$
4. $e_1 = 1 - by_1$ $q_0 = ay_0$
5. $y_2 = y_1 + e_1 y_1$ $r_0 = a - bq_0$
6. $e_2 = 1 - by_2$ $q_1 = q_0 + r_0 y_2$
7. $y_3 = y_2 + e_2 y_2$ $r_1 = a - bq_0$
8. $q = q_1 + r_1 y_3$

This algorithm needs 8 times the basic `fma` latency, i.e. $8 \times 5 = 40$ cycles.

For extreme inputs, underflow and overflow can occur, and the formal proof needs to take account of this.

Improved theorems

In proving Markstein's theorems formally in HOL, we noticed a way to strengthen them. For the main theorem, instead of requiring y_0 to be perfectly rounded, we can require only a relative error:

$$|y_0 - \frac{1}{b}| < |\frac{1}{b}|/2^p$$

where p is the floating point precision. Actually Markstein's original proof only relied on this property, but merely used it as an intermediate consequence of perfect rounding.

The altered precondition looks only trivially different, and in the worst case it is. However it is in general much easier to achieve.

Achieving the relative error bound

Suppose y_0 results from rounding a value y_0^*.

The rounding can contribute as much as $\frac{1}{2}\ ulp(y_0^*)$, which in all significant cases is the same as $\frac{1}{2}\ ulp(\frac{1}{b})$.

Thus the relative error condition after rounding is achieved provided y_0^* is in error by no more than

$$|\frac{1}{b}|/2^p - \frac{1}{2}\ ulp(\frac{1}{b})$$

In the worst case, when b's mantissa is all $1s$, these two terms are almost identical so extremely high accuracy is needed. However at the other end of the scale, when b's mantissa is all 0s, they differ by a factor of two.

Thus we can generalize the way Markstein's reciprocal theorem isolates a single special case.

Stronger reciprocal theorem

We have the following generalization: if y_0 results from rounding a value y_0^* with relative error better than $\frac{d}{2^{2p}}$:

$$|y_0^* - \frac{1}{b}| \leq \frac{d}{2^{2p}}|\frac{1}{b}|$$

then y_0 meets the relative error condition for the main theorem, *except possibly when the mantissa of b is one of the d largest, i.e. when considered as an integer is $2^p - d \leq m \leq 2^p - 1$.*

Hence, we can compute y_0 more 'sloppily', and hence perhaps more efficiently, at the cost of explicitly checking more special cases.

An improved algorithm

The following algorithm can be justified by applying the theorem with $d = 165$, explicitly checking 165 special cases.

1. $y_0 = \frac{1}{b}(1 + \epsilon)$ [`frcpa`]
2. $d = 1 - by_0$ $q_0 = ay_0$
3. $y_1 = y_0 + dy_0$ $d' = d + dd$ $r_0 = a - bq_0$
4. $e = 1 - by_1$ $y_2 = y_0 + d'y_0$ $q_1 = q_0 + r_0y_1$
5. $y_3 = y_1 + ey_2$ $r_1 = a - bq_1$
6. $q = q_1 + r_1y_3$

On a machine capable of issuing three FP operations per cycle, this can be run in 6 FP latencies.

Itanium^{TM} can only issue two FP instructions per cycle, but since it is fully pipelined, this only increases the overall latency by one cycle, not a full FP latency. Thus the whole algorithm runs in 31 cycles.

Conclusions

Because of HOL's mathematical generality, all the reasoning needed is done in a unified way with the customary HOL guarantee of soundness:

- Underlying pure mathematics
- Formalization of floating point operations
- Proof of the special Markstein-type theorems
- Routine relative error computation for the final result before rounding
- Explicit computation with the special cases isolated.

Moreover, because HOL is programmable, many of these parts can be, and have been, automated.

Finally, the detailed examination of the proofs that formal verification requires threw up significant improvements that have led to some faster algorithms.