Verifying Relative Error Bounds Using Symbolic Simulation

Jesse Bingham & Joe Leslie-Hurd

Intel Corporation

CAV
Sunday 20 July 2014

Presented by Alan Hu (U. British Columbia)

Motivating Example

- RCP14 is a floating point instruction that computes an approximate reciprocal.
- The spec requires that for every input x, the relative error of the hardware output $h(x)$ is at most 2^{-14}:

$$\frac{|h(x) - 1/x|}{|1/x|} < 2^{-14}$$

- In general, if f is a mathematical function, we write $h(x) \approx_p f(x)$ to indicate that the hardware result $h(x)$ approximates $f(x)$ within relative error 2^{-p}
- Floating point hardware typically has a deterministic spec – each input has exactly one correct answer. Our spec is not of this form, hence existing verification flows cannot be used.

Our Formal Verification Flow

1. Given symbolic input x, compute $h(x)$ via symbolic simulation.
2. Prove a meta-theorem that reduces $h(x) \approx_p f(x)$ to symbolic integer reasoning of the form

   $$B \;\lessgtr\; \prod_{i=1}^{n} M_i$$

   where $\lessgtr$ is either $<$ or $>$, B is a concrete integer, and $M_1, \ldots, M_n$ are symbolic integers.
3. Use customized algorithms to decide the resulting symbolic integer product inequality.

Talk Plan

1. Reducing to Symbolic Integer Reasoning
2. Deciding Symbolic Product Inequalities
3. Experiments
4. Summary

Approximate Reciprocal Verification

- Given an input x, symbolically simulate the hardware to compute $h(x)$, an approximation to $1/x$.
- x and $h(x)$ are symbolic floating point values.
  - The real number represented by x is

$$\frac{s_x m_x 2^{e_x}}{2^\ell}$$

  - $s_x = \pm 1$ is the sign.
  - $m_x \in [2^\ell, 2^{\ell+1})$ is the mantissa.
  - $e_x \in [e_{min}, e_{max}]$ is the exponent.
- **Verification Task:** Check $h(x) \approx_p 1/x$.
  - *i.e., the relative error of $h(x)$ is within 2^{-p} of $1/x$*

Reducing Approximate Reciprocal to Integer Reasoning

We reduce the verification task to integers like so:

$$
\begin{aligned}
& h(x) \approx_p 1/x \\
\iff & |h(x) - 1/x| \;/\; |1/x| < 2^{-p} \\
\iff & s_x = s_{h(x)} \;\wedge\; (1) \\
& -2 \leq e_x + e_{h(x)} \leq 0 \;\wedge\; (2) \\
& 2^{2\ell+2}(1 - 2^{-p}) < m_x m_{h(x)} 2^{e_x + e_{h(x)} + 2} < 2^{2\ell+2}(1 + 2^{-p}) \; (3)
\end{aligned}
$$

The final conjunction breaks down as follows:

1. The output sign must be the same as the input sign.
2. The output exponent must be equal to or one less than the negated input exponent.
3. The output mantissa must satisfy the two inequalities above.
   - Since $p \leq 2\ell + 2$, these are symbolic integer inequalities.

Reducing Other Approximate Instructions

- The `RSQRT` family of instructions approximate $1/\sqrt{x}$.
  - This reduction is a straightforward modification of the reduction for `RCP`.
- The `EXP2` family of instructions approximate 2^x.
  - This reduction uses a novel technique based on iterated square-roots of the constant 2.
- The details of all our reductions are in the paper, but in each case the result has the form

$$B \;\lessgtr\; \prod_{i=1}^{n} M_i$$

where $\lessgtr$ is either $<$ or $>$, B is a concrete integer, and $M_1, \ldots, M_n$ are symbolic integers.

Hardest Proof Obligation: Product Inequalities

- We have reduced checking $h(x) \approx_{14} 1/x$ to

$$L \; < \; m_x m_{h(x)} 2^{e_x + e_{h(x)} + 2} \; < \; U$$

  where L and U are constant integers
  - For the other instructions, we get similar inequalities.
  - Note $2^{e_x + e_{h(x)} + 2} \in \{1, 2, 4\}$.
- Deciding these inequalities by explicitly performing the symbolic integer multiplication $m_x m_{h(x)}$ is prohibitively expensive.
- We developed a suite of algorithms that avoid this blow-up by only approximating the product.

Algorithmic Framework

- Let $\Pi = \prod_{i=1}^{n} M_i$; we wish to establish $L < \Pi$
- Common theme: compute sequence of approximations $a_0, a_1, \ldots$ to Π, where $a_i \leq a_{i+1} \leq \Pi$ for all $i \geq 0$.
- If we reach an i such that $L < a_i$, we have clearly established $L < \Pi$
- Two optimizations
  - Replace each a_i with $\mathbf{ite}(L < a_i, 0, a_i)$; BDD complexity is restricted to "space" wherein the inequality is yet-to-be proven
  - Replace each a_i with $truncL_t(a) = 2^t \lfloor a2^{-t} \rfloor$; e.g. zero out t lower order "bits" (t guessed by user)

The "Partial Product Summation" algorithm for $B < xy$

function PP_BOUND_LOWER(B, x, y)
  $a := 0$ ▷ B, x, y and a are symbolic naturals
  $sat := false$ ▷ sat is a BDD
  for $i := r$ **downto** 0 **do** ▷ r is "bit width" of y
    $a := a + y_i x 2^i$
    $sat := sat \vee B < a$
    if $sat = True$ **then** ▷ if sat is tautological we're done
      return $True$
    end if
    $a :=$ **ite**($sat, 0, a$) ▷ Sat Space Restriction
    $a := truncL_t(a)$ ▷ Truncation
  end for
  return sat
end function

Case Studies

- 12 distinct instructions verified
- RCP and RSQRT: 5 flavours each, involving single (SP, 32 bit) or double (DP, 64 bit) -precision floats, with relative errors 2^{-11}, 2^{-14}, and 2^{-28}
  - 2^{-14} flavors support denormal inputs and outputs; handling these was simple in our framework (see paper for details)
- EXP2: two flavors that output SP or DP, both take 32-bit *fixed* point input; relative error is 2^{-23}
- Case-splitting required for more challenging instructions
  - Designs all based on Look-up-tables (LUT); case splits held constant some or all input bits used in LUT index.
  - *embarrassingly parallelizable*, ran many cases concurrently to reduce wall clock time

Results

Op.	Tot. Time	Spec. Time.	Mem.	Alg.	Case split
RCP 11S	58	3	1.8	P	No
RCP 14S	103	49	1.8	P	No
RCP 14D	135	51	1.8	P	No
RCP 28S	14,972	7,038	17.4	E	No
RCP 28D	2.7 days	1.3 days	3.6	E	512-way
RSQRT 11S	68	4	1.8	P	No
RSQRT 14S	124	69	1.8	P	No
RSQRT 14D	139	55	1.8	P	No
RSQRT 28S	18,301	13,173	6.0	E	16-way
RSQRT 28D	22.7 days[1]	16.7 days	9.0	E	1,024-way
EXP2 23S	72,759	63,428	2.9	B	128-way
EXP2 23D	59,706	51,152	2.8	B	128-way

[1] Wall clock just over 2 days

Summary

- A new technique for verifying approximate floating point instructions, integrated with symbolic simulation of RTL.
- Based on reducing the approximate spec to an integer problem which can be solved by symbolic computation techniques.
- Used to formally verify the `RCP`, `RSQRT` and `EXP2` family of instructions for a next-generation Intel® processor.
- `EXP2` never done before; [Sawada 2010] handles some `RCP` and `RSQRT` instructions using ACL2, our proofs run faster however.
- Please get in touch if you are interested in finding out more

`jesse.d.bingham@intel.com`
`joe.leslie-hurd@intel.com`

Backup

Reducing Floating Point Problems to Integers

- Reducing floating point problems to integers is a well-known technique.
- Our goal is to make it easier to perform a symbolic computation.
- Another application is to generate hard examples to test rounding modes of floating point units:[2]
  - $\sqrt{16777210 \times 2^{24}} = 16777212.9999997\ldots$
  - $\sqrt{10873622 \times 2^{23}} = 9550631.0000007\ldots$

[2] Michael Parks. Number-theoretic test generation for directed rounding. *IEEE Transactions on Computers*, 49(7):651–658, 2000.