23rd IEEE Symposium on Computer Arithmetic
Santa Clara, CA, USA, 10-13 July 2016

Automated design of floating-point logarithm functions on integer processors

Guillaume Revy
(presented by Florent de Dinechin)

Univ. Perpignan Via Domitia, DALI project-team
Univ. Montpellier 2, LIRMM, UMR 5506
CNRS, LIRMM, UMR 5506

Summary

Context and objectives

- Automated synthesis of floating-point function implementation
  - particular case of logarithm functions $\log_b(x)$
  - work done within the french ANR MetaLibm project (`http://www.metalibm.org`)
- FLIP software library
  - low latency and correctly-rounded implementation
  - VLIW integer processor of the ST200 family
  - no exception handling

Achievements

1. Unified range reduction for $\log_b(x)$ implementation
2. Correctly-rounded $\log_2(x)$, $\log(x)$, and $\log_{10}(x)$ for the binary32 format
   - $\approx$ 200 cycles on the ST231

Problem statement

$b \in \mathbb{R} > 1$

$(p, e_{\min}, e_{\max})$

$\circ \in \{\mathrm{RN}, \mathrm{RU}, \mathrm{RD}, \mathrm{RZ}\}$

Problem statement

$x = m \cdot 2^e,\ x > 0$

$b \in \mathbb{R} > 1$

$(p, e_{\min}, e_{\max})$

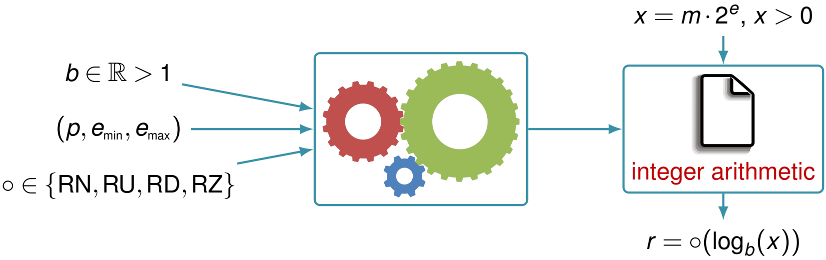
$\circ \in \{\mathrm{RN}, \mathrm{RU}, \mathrm{RD}, \mathrm{RZ}\}$

integer arithmetic

$r = \circ(\log_b(x))$

- input/output: ± 0, $\pm\infty$, NaN, subnormal and normal numbers

Problem statement



- input/output: ± 0, $\pm\infty$, NaN, subnormal and normal numbers

- Our focus: IEEE-754 binary32 implementations on the ST231
  - $b \in \{2, \exp(1), 10\}$
  - $(p, e_{\min}, e_{\max}) = (24, -126, 127)$
  - $\circ = \mathrm{RN}$
  - no underflow nor overflow
  - 4-issue VLIW 32-bit integer processor
  - 1-cycle ALU / 3-cycle MUL
  - 64-bit arithmetic emulated in software
  - high latency memory accesses

Outline of the talk

1. Unified evaluation scheme for $\log_b(x)$
2. Error analysis
3. Implementation and results
4. Concluding remarks and future work

Outline of the talk

1. Unified evaluation scheme for $\log_b(x)$

2. Error analysis

3. Implementation and results

4. Concluding remarks and future work

Logarithm range reduction

- Let x be a non-negative normal floating-point number, with $x \neq 1$

$$\log_b(x) = 2^d \cdot u \quad \rightarrow \quad \mathrm{RN}(\log_b(x)) = (-1)^{\mathrm{sign}(u)} \cdot 2^d \cdot \mathrm{RN}(|u|)$$

with $|u| \in [1,2)$ and $d \in \{e_{\min}, \cdots, e_{\max}\}$

Logarithm range reduction

- Let x be a non-negative normal floating-point number, with $x \neq 1$

$$\log_b(x) = 2^d \cdot u \quad \rightarrow \quad \mathrm{RN}(\log_b(x)) = (-1)^{\mathrm{sign}(u)} \cdot 2^d \cdot \mathrm{RN}(|u|)$$

with $|u| \in [1,2)$ and $d \in \{e_{\min}, \cdots, e_{\max}\}$

- $x = 2^e \cdot m \quad \rightarrow \quad \log_b(x) = \log_b(2) \cdot e + \log_b(m), \quad \text{with} \quad m \in [1,2]$
  - catastrophic cancellation when $e = -1$

Logarithm range reduction

- Let x be a non-negative normal floating-point number, with $x \neq 1$

$$\log_b(x) = 2^d \cdot u \quad \rightarrow \quad \mathrm{RN}(\log_b(x)) = (-1)^{\mathrm{sign}(u)} \cdot 2^d \cdot \mathrm{RN}(|u|)$$

with $|u| \in [1,2)$ and $d \in \{e_{\min}, \cdots, e_{\max}\}$

- $x = 2^e \cdot m \quad \rightarrow \quad \log_b(x) = \log_b(2) \cdot e + \log_b(m), \quad$ with $\quad m \in [1,2]$
  - catastrophic cancellation when $e = -1$

- Range reduction: define $\tau = 1$ if $m \geq 1.5, \;$ 0 otherwise

$$\log_b(x) = \log_b(2) \cdot (e + \tau) + \log_b(m/2^\tau), \quad \text{with} \quad m/2^\tau \in [0.75, 1.5]$$

  - no branch instruction

Logarithm range reduction

- Let x be a non-negative normal floating-point number, with $x \neq 1$

$$\log_b(x) = 2^d \cdot u \quad \to \quad \mathrm{RN}(\log_b(x)) = (-1)^{\mathrm{sign}(u)} \cdot 2^d \cdot \mathrm{RN}(|u|)$$

  with $|u| \in [1,2)$ and $d \in \{e_{\min}, \cdots, e_{\max}\}$

- $x = 2^e \cdot m \quad \to \quad \log_b(x) = \log_b(2) \cdot e + \log_b(m), \quad \text{with} \quad m \in [1,2]$
  - catastrophic cancellation when $e = -1$

- Range reduction: define $\tau = 1$ if $m \geq 1.5$, 0 otherwise

$$\log_b(x) = \underbrace{\log_b(2) \cdot (e+\tau) + \log_b(m/2^\tau)}_{|u| \cdot 2^d}, \quad \text{with} \quad m/2^\tau \in [0.75, 1.5]$$

  - no branch instruction

Evaluation scheme: $\log_b(2) \cdot (e + \tau)$

- Let $t = m/2^\tau - 1 \in [-0.25, 0.5]$, the objective is to evaluate

$$\log_b(x) = \log_b(2) \cdot (e + \tau) + \log_b(1 + t)$$

Evaluation scheme: $\log_b(2) \cdot (e + \tau)$

- Let $t = m/2^\tau - 1 \in [-0.25, 0.5]$, the objective is to evaluate

$$\log_b(x) = \log_b(2) \cdot (e + \tau) + \log_b(1 + t)$$

- $\log_b(2)$ known at compile time: for example $\log_{10}(2) = 0.3010\cdots$

$$\log_{10}(2) = 0.0100110100010000010011010100001001 1\cdots$$

Evaluation scheme: $\log_b(2) \cdot (e + \tau)$

- Let $t = m/2^\tau - 1 \in [-0.25, 0.5]$, the objective is to evaluate

$$\log_b(x) = \log_b(2) \cdot (e + \tau) + \log_b(1 + t)$$

- $\log_b(2)$ known at compile time: for example $\log_{10}(2) = 0.3010 \cdots$

$$\log_{10}(2) = 0.0100110100010000010011010100010011 \cdots$$

  - stored as a signed integer $(00010011010001000001001101010000)_2$ on k bits, associated with an implicit scaling factor = 2^{-30}

Evaluation scheme: $\log_b(2) \cdot (e + \tau)$

- Let $t = m/2^\tau - 1 \in [-0.25, 0.5]$, the objective is to evaluate

$$\log_b(x) = \log_b(2) \cdot (e + \tau) + \log_b(1 + t)$$

- $\log_b(2)$ known at compile time: for example $\log_{10}(2) = 0.3010\cdots$

$$\log_{10}(2) = 0.010011010001000001001101010000 10011\cdots$$

  - stored as a signed integer $(0\,00100110100010000010011010100 00)_2$ on k bits, associated with an implicit scaling factor = 2^{-30}

- Hence, rewrite $\log_b(2)$ as

$$\log_b(2) = 2^\mu \cdot \varphi, \quad \text{with} \quad 1 \leq \varphi < 2$$

  - scaling done statically at synthesis-time

Evaluation scheme: $\log_b(2) \cdot (e+\tau)$

- Let $t = m/2^{\tau} - 1 \in [-0.25, 0.5]$, the objective is to evaluate

$$\log_b(x) = \log_b(2) \cdot (e+\tau) + \log_b(1+t)$$

- $\log_b(2)$ known at compile time: for example $\log_{10}(2) = 0.3010\cdots$

$$\log_{10}(2) = 0.0100110100010000010011010100010011\cdots$$

  - stored as a signed integer $(000100110100010000010011010100 00)_2$ on k bits, associated with an implicit scaling factor = 2^{-30}

- Hence, rewrite $\log_b(2)$ as

$$\log_b(2) = 2^{\mu} \cdot \varphi, \quad \text{with} \quad 1 \leq \varphi < 2$$

  - scaling done statically at synthesis-time

- The formula becomes

$$\log_b(x) = 2^{\mu} \cdot \left(\varphi \cdot (e+\tau) + \log_b(1+t)/2^{\mu}\right)$$

Evaluation scheme: $\log_b(m/2^\tau)$

- Table lookup-based method
  - Tang (1990) and Gal (1991)
  - tabulated values of $\log(x)$ combined with small degree polynomial evaluation
  - requires storing tabulated values on memory

- Polynomial evaluation-based method
  - Schulte and Swartzlander (1993)
  - univariate polynomial to compte $\log_2(x)$
  - our approach: particular bivariate polynomial

Polynomial approximation

- The formula so far

$$\log_b(x) = 2^{\mu} \cdot \left(\varphi \cdot (e + \tau) + \log_b(1+t)/2^{\mu}\right)$$

Polynomial approximation

- The formula so far

$$\log_b(x) = 2^{\mu} \cdot \left(\varphi \cdot (e + \tau) + \log_b(1+t)/2^{\mu}\right)$$

- For example, when $(b, \mu) = (10, -2)$

$$\begin{aligned}
\log_{10}(1 + (0.5 - 2^{-23}))/2^{\mu} &= 0.10110100010100010100010001011011000001011011\cdots \\
\log_{10}(1 + 2^{-24})/2^{\mu} &= 0.00000000000000000000000110111100101101111011\cdots
\end{aligned}$$

Polynomial approximation

- The formula so far

$$\log_b(x) = 2^{\mu} \cdot \left(\varphi \cdot (e + \tau) + \log_b(1+t)/2^{\mu} \right)$$

- For example, when $(b, \mu) = (10, -2)$

$$\begin{aligned} \log_{10}(1 + (0.5 - 2^{-23}))/2^{\mu} &= 0.10110100010100010100010001011011000001011011\cdots \\ \log_{10}(1 + 2^{-24})/2^{\mu} &= 0.00000000000000000000000110111100101101111011\cdots \end{aligned}$$

  - when $e = 0$, we need to compute $\log_{10}(1+t)/2^{\mu}$ on a large number of bits to get sufficient accuracy after renormalization

Polynomial approximation

- The formula so far

$$\log_b(x) = 2^\mu \cdot \left(\varphi \cdot (e+\tau) + \log_b(1+t)/2^\mu\right)$$

- For example, when $(b,\mu) = (10,-2)$

$$\begin{aligned} \log_{10}(1+(0.5-2^{-23}))/2^\mu &= 0.1011010001010001010001000101101100000101 1011\cdots \\ \log_{10}(1+2^{-24})/2^\mu &= 0.000000000000000000000001101111001011011 11011\cdots \end{aligned}$$

  - when $e=0$, we need to compute $\log_{10}(1+t)/2^\mu$ on a large number of bits to get sufficient accuracy after renormalization

- Hence rewrite $t = 2^\delta h$, where $|h| \geq 0.5$

  (δ is the leading zero count on t, and h is t shifted by δ bits)

$$\frac{\log_b(1+t)}{2^\mu} = 2^\delta \cdot h \cdot \frac{\log_b(1+t)}{2^\mu \cdot t} \quad \text{with} \quad \frac{\log_b(1+t)}{2^\mu \cdot t} \in (1,4)$$

  - scaling done dynamically at evaluation-time

Evaluation scheme

- Objective: compute $\mathrm{RN}(\log_b(x))$, with

$$\log_b(x) = 2^{\mu} \cdot \left(\varphi \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t}\right)$$

Evaluation scheme

- Objective: compute $\mathrm{RN}(\log_b(x))$, with

$$\log_b(x) = 2^{\mu} \cdot \left(\varphi \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t} \right)$$

- Main step

$$\underbrace{\varphi \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t}}_{|u| \cdot 2^{c}}$$

  - and $d = c + \mu$

Evaluation scheme

- Objective: compute $\mathrm{RN}(\log_b(x))$, with

$$\log_b(x) = 2^{\mu} \cdot \left(\varphi \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t} \right)$$

- Main step

$$\underbrace{\varphi \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t}}_{|u| \cdot 2^{c}}$$

  - and $d = c + \mu$

- The value u is approximated by a finite-precision value $\hat{u}$, such that

$$|u - \hat{u}| < \varepsilon.$$

Outline of the talk

1. Unified evaluation scheme for $\log_b(x)$

2. Error analysis

3. Implementation and results

4. Concluding remarks and future work

Three sources of error

- Objective: approximate u by a finite precision value $\hat{u}$

$$u = 2^{-c} \cdot \left(\varphi \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t} \right)$$

Three sources of error

- Objective: approximate u by a finite precision value $\hat{u}$

$$u = 2^{-c} \cdot \left(\varphi \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t} \right)$$

$$\Big\downarrow \quad |\varphi - \hat{\varphi}| \leq 2^{2-k}$$

$$2^{-c} \cdot \left(\hat{\varphi} \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t} \right)$$

Three sources of error

- Objective: approximate u by a finite precision value $\hat{u}$

$$u = 2^{-c} \cdot \left(\varphi \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t} \right)$$

$$\Big\downarrow \quad |\varphi - \hat{\varphi}| \le 2^{2-k}$$

$$2^{-c} \cdot \left(\hat{\varphi} \cdot (e + \tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t} \right)$$

$$\Big\downarrow \quad \alpha(a)$$

$$2^{-c} \cdot \Big(\hat{\varphi} \cdot (e + \tau) + 2^{\delta} \cdot h \cdot a(t) \Big)$$

Three sources of error

- Objective: approximate u by a finite precision value $\hat{u}$

$$u = 2^{-c} \cdot \left(\varphi \cdot (e+\tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t} \right)$$

$$\Big\downarrow \quad |\varphi - \hat{\varphi}| \leq 2^{2-k}$$

$$2^{-c} \cdot \left(\hat{\varphi} \cdot (e+\tau) + 2^{\delta} \cdot h \cdot \frac{\log_b(1+t)}{2^{\mu} \cdot t} \right)$$

$$\Big\downarrow \quad \alpha(a)$$

$$2^{-c} \cdot \left(\hat{\varphi} \cdot (e+\tau) + 2^{\delta} \cdot h \cdot a(t) \right)$$

$$\Big\downarrow \quad \rho(\mathcal{P})$$

$$\hat{u} = 2^{-c} \cdot \left(\hat{\varphi} \otimes (e+\tau) \oplus 2^{\delta} \cdot \underbrace{h \otimes a(t)}_{\text{bivariate polynomial}} \right)$$

Sufficient error bounds

- By triangular inequality

$$|u - \hat{u}| \leq 2^{6-k} + (2 - 2^{3-p}) \cdot \alpha(a) + \rho(\mathcal{P})$$

- To compute $\hat{u}$ such that $|u - \hat{u}| < \varepsilon$, we have to ensure

$$2^{6-k} + (2 - 2^{3-p}) \cdot \alpha(a) + \rho(\mathcal{P}) < \varepsilon$$

Sufficient error bounds

- By triangular inequality

$$|u - \hat{u}| \leq 2^{6-k} + (2 - 2^{3-p}) \cdot \alpha(a) + \rho(\mathcal{P})$$

- To compute $\hat{u}$ such that $|u - \hat{u}| < \varepsilon$, we have to ensure

$$2^{6-k} + (2 - 2^{3-p}) \cdot \alpha(a) + \rho(\mathcal{P}) < \varepsilon$$

- More particularly, we have to compute a polynomial approximant $a(t)$ such that

$$\alpha(a) < \frac{\varepsilon - 2^{6-k}}{2 - 2^{3-p}}$$

Sufficient error bounds

- By triangular inequality

$$|u - \hat{u}| \leq 2^{6-k} + (2 - 2^{3-p}) \cdot \alpha(a) + \rho(\mathcal{P})$$

- To compute $\hat{u}$ such that $|u - \hat{u}| < \varepsilon$, we have to ensure

$$2^{6-k} + (2 - 2^{3-p}) \cdot \alpha(a) + \rho(\mathcal{P}) < \varepsilon$$

- More particularly, we have to compute a polynomial approximant $a(t)$ such that

$$\alpha(a) \leq \theta \quad \text{with} \quad \theta < \frac{\varepsilon - 2^{6-k}}{2 - 2^{3-p}}$$

Sufficient error bounds

- By triangular inequality

$$|u - \hat{u}| \leq 2^{6-k} + (2 - 2^{3-p}) \cdot \alpha(a) + \rho(\mathcal{P})$$

- To compute $\hat{u}$ such that $|u - \hat{u}| < \varepsilon$, we have to ensure

$$2^{6-k} + (2 - 2^{3-p}) \cdot \alpha(a) + \rho(\mathcal{P}) < \varepsilon$$

- More particularly, we have to compute a polynomial approximant $a(t)$ such that

$$\alpha(a) \leq \theta \quad \text{with} \quad \theta < \frac{\varepsilon - 2^{6-k}}{2 - 2^{3-p}}$$

- And finally, we have to find a finite-precision evaluation program $\mathcal{P}$

$$\rho(\mathcal{P}) < \varepsilon - 2^{6-k} - (2 - 2^{3-p}) \cdot \theta$$

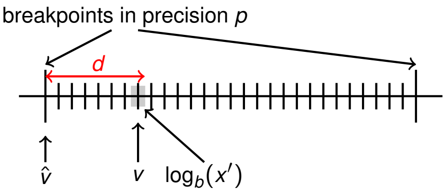
Computation of the required error bound

- Objective: compute $\hat{u}$ such that $|u - \hat{u}| < \varepsilon$
  - to ensure correct rounding
  - Table's Maker Dilemma
  - exhaustive searching feasible for the binary32 floating-point format (with $p = 24$)

Computation of the required error bound

- Objective: compute $\hat{u}$ such that $|u - \hat{u}| < \varepsilon$
  - to ensure correct rounding
  - Table's Maker Dilemma
  - exhaustive searching feasible for the binary32 floating-point format (with $p = 24$)

- Exhaustive search
  - smallest distance between $\log_b(x')$ and the middle of two floating-point numbers in precision p, for all floating-point inputs x'



Computation of the required error bound

- Objective: compute $\hat{u}$ such that $|u - \hat{u}| < \varepsilon$
  - to ensure correct rounding
  - Table's Maker Dilemma
  - exhaustive searching feasible for the binary32 floating-point format (with $p = 24$)

- Exhaustive search
  - smallest distance between $\log_b(x')$ and the middle of two floating-point numbers in precision p, for all floating-point inputs x'

b	2	$\exp(1)$	10
error bound: $-\log_2(\varepsilon)$	51	58	56

  - for example: $x = 127837836949849943048192$

$$\log(x) = \underbrace{\color{green}{1.10101001101000111111000}1000000000000000000000000000000000}_{58\text{ bits}}11\cdots \times 2^5$$

Outline of the talk

1. Unified evaluation scheme for $\log_b(x)$
2. Error analysis
3. Implementation and results
4. Concluding remarks and future work

Determining word-length k and polynomial degree

- Computation of a polynomial approximant $a(t)$, such that

$$\alpha(a) < \frac{\varepsilon - 2^{6-k}}{2 - 2^{3-p}} = \frac{\varepsilon - 2^{6-k}}{2 - 2^{-21}}, \quad \text{since} \quad p = 24$$

- The word-length k must be chosen such that this bound remains non-negative

b	2	$\exp(1)$	10
error bound: $-\log_2(\varepsilon)$	51	58	56
value of k	64	96	64

Determining word-length k and polynomial degree

- Computation of a polynomial approximant $a(t)$, such that

$$\alpha(a) < \frac{\varepsilon - 2^{6-k}}{2 - 2^{3-p}} = \frac{\varepsilon - 2^{6-k}}{2 - 2^{-21}}, \quad \text{since} \quad p = 24$$

- The word-length k must be chosen such that this bound remains non-negative

b	2	$\exp(1)$	10
error bound: $-\log_2(\varepsilon)$	51	$\cancel{58}$ 56^*	56
value of k	64	$\cancel{96}$ 64	64

- For $\log(x)$, only three inputs required more accuracy than 2^{-56}
  - we first ignore these inputs at synthesis-time
  - then we check if the implementation returns the correct answer for these inputs
  - interest: reducing the word-length k to 64

Design space exploration: polynomial evaluator

- Given ε, the synthesis process is the following

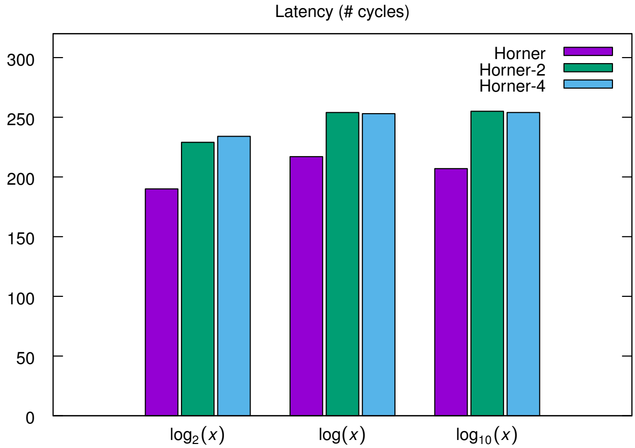
1. determine the smallest degree of $a(t)$ so as the error bound is satisfied

b	2	$\exp(1)$	10
error bound: $-\log_2(\varepsilon)$	51	56^*	56
smallest degree of a	19	21 (22)	21

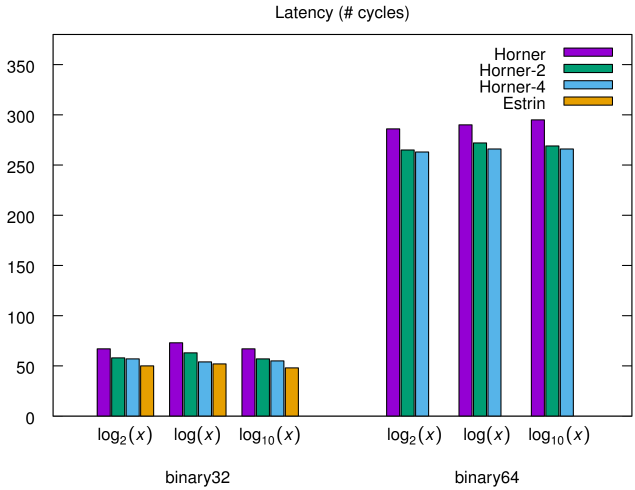
2. build polynomial approximant and compute the approximation error bound
3. build evaluation program (CGPE), and check its evaluation error (Gappa)
   - explore Horner, Estrin, and other parallel schemes

   $\longrightarrow$ if no program can be found, increase the polynomial degree

Correctly-rounded binary32 implementations

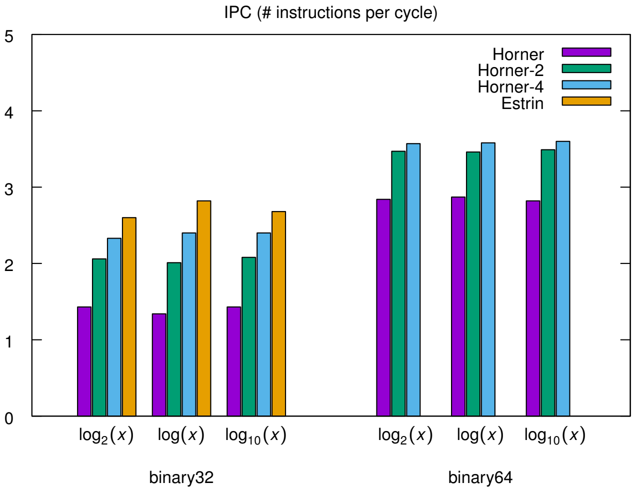


Faithful binary32/64 implementations



- Estrin rule exposes much more ILP than the other schemes $\rightarrow$ register spilling in binary64

Faithful binary32/64 implementations



- Estrin rule exposes much more ILP than the other schemes $\rightarrow$ register spilling in binary64

Outline of the talk

1. Unified evaluation scheme for $\log_b(x)$

2. Error analysis

3. Implementation and results

4. Concluding remarks and future work

Conclusion remarks and future work

Work done so far

- Automated design of floating-point implementation of logarithm functions
  - unified range reduction
  - polynomial evaluation-based algorithm
- Correctly-rounded binary32 implementation in $\approx$ 200 cycles on the ST231
  - FDLibm over SoftFloat was 1400+ cycles
  - FDLibm over FLIP was about 1000 cycles
- Extension to faithful binary32/64 implementations

Future work is threefold

- Evaluate the performance of this approach on other architectures
- Extend this approach to other transcendental functions, like $\exp_b(x)$
- Study the impact on performances of the polynomial evaluation scheme

23rd IEEE Symposium on Computer Arithmetic
Santa Clara, CA, USA, 10-13 July 2016

Automated design of floating-point logarithm functions on integer processors

Guillaume Revy
(presented by Florent de Dinechin)

Univ. Perpignan Via Domitia, DALI project-team
Univ. Montpellier 2, LIRMM, UMR 5506
CNRS, LIRMM, UMR 5506