Advancements in `(cr)libm` development

Presentation at Intel - Portland

Christoph Quirin Lauter

Arénaire team
Laboratoire de l'Informatique et du Parallélisme
École Normale Supérieure de Lyon

Portland, 10 october 2007

Introduction

Introduction

Correct rounding of x^y

Automatic implementation of `libm functions`

Conclusion

Correctly rounded elementary functions - crlibm

crlibm[1]: correctly rounded elementary function library

[1] http://lipforge.ens-lyon.fr/www/crlibm/

Correctly rounded elementary functions - crlibm

crlibm[1]: correctly rounded elementary function library

- Elementary functions as in an usual libm:
  - exp
  - sin
  - cos
  - ...

[1] http://lipforge.ens-lyon.fr/www/crlibm/

Correctly rounded elementary functions - `crlibm`

`crlibm`[1]: correctly rounded elementary function library

- Elementary functions as in an usual `libm`:
  - `exp`
  - `sin`
  - `cos`
  - ...
- Bit-exact, correctly rounded results `f(x)`$= \circ(f(x))$

[1] http://lipforge.ens-lyon.fr/www/crlibm/

Correctly rounded elementary functions - `crlibm`

`crlibm`[1]: correctly rounded elementary function library

- Elementary functions as in an usual `libm`:
  - `exp`
  - `sin`
  - `cos`
  - ...
- Bit-exact, correctly rounded results `f(x)`$= \circ(f(x))$
- No important impact on average performance

[1] `http://lipforge.ens-lyon.fr/www/crlibm/`

Correctly rounded elementary functions - crlibm

crlibm[1]: correctly rounded elementary function library

- Elementary functions as in an usual libm:
  - exp
  - sin
  - cos
  - ...
- Bit-exact, correctly rounded results $\mathtt{f(x)} = \circ(f(x))$
- No important impact on average performance
- Guaranteed worst case performance

[1] http://lipforge.ens-lyon.fr/www/crlibm/

Correctly rounded elementary functions - crlibm

crlibm[1]: correctly rounded elementary function library

- Elementary functions as in an usual libm:
  - exp
  - sin
  - cos
  - ...
- Bit-exact, correctly rounded results f(x)$= \circ(f(x))$
- No important impact on average performance
- Guaranteed worst case performance
- Challenge: Correct rounding requires high accuracy and complete proofs

[1] http://lipforge.ens-lyon.fr/www/crlibm/

This talk goes on...

- Advancements in the correct rounding of x^y
- Techniques for automatic implementation of `libm` functions.

Correct rounding of x^y

Introduction

Correct rounding of x^y

Automatic implementation of `libm functions`

Conclusion

Worst-case computations

- Correct rounding must overcome the Table Maker's Dilemma

$$\circ(f(x) \cdot (1 + \varepsilon)) \stackrel{?}{=} \circ(f(x))$$

Worst-case computations

- Correct rounding must overcome the Table Maker's Dilemma

$$\circ(f(x)\cdot(1+\varepsilon)) \stackrel{?}{=} \circ(f(x))$$

- Finite domain, x is FP number, $x \in \mathbb{F}$: worst-case $\overline{\varepsilon}$ exists

$$\exists\overline{\varepsilon} > 0 \,.\, \forall\varepsilon, |\varepsilon| \leq \overline{\varepsilon} \,.\, \forall x \in \mathbb{F}.\, \circ\left(f(x)\cdot(1+\varepsilon)\right) = f(x)$$

Worst-case computations

- Correct rounding must overcome the Table Maker's Dilemma

$$\circ(f(x) \cdot (1+\varepsilon)) \stackrel{?}{=} \circ(f(x))$$

- Finite domain, x is FP number, $x \in \mathbb{F}$: worst-case $\overline{\varepsilon}$ exists

$$\exists \overline{\varepsilon} > 0 \,.\, \forall \varepsilon, |\varepsilon| \leq \overline{\varepsilon} \,.\, \forall x \in \mathbb{F} .\, \circ\left(f(x) \cdot (1+\varepsilon)\right) = f(x)$$

- Univariate functions implemented in double precision:
  - Computation of $\overline{\varepsilon}$ actually possible (Lefèvre, Stehlé et al.)

Worst-case computations

- Correct rounding must overcome the Table Maker's Dilemma

$$\circ(f(x) \cdot (1+\varepsilon)) \stackrel{?}{=} \circ(f(x))$$

- Finite domain, x is FP number, $x \in \mathbb{F}$: worst-case $\overline{\varepsilon}$ exists

$$\exists \overline{\varepsilon} > 0 \,.\, \forall \varepsilon, |\varepsilon| \leq \overline{\varepsilon} \,.\, \forall x \in \mathbb{F}. \circ (f(x) \cdot (1+\varepsilon)) = f(x)$$

- Univariate functions implemented in double precision:
  - Computation of $\overline{\varepsilon}$ actually possible (Lefèvre, Stehlé et al.)
  - Computation of $\overline{\varepsilon}$ is a smart exhaustive search

Worst-case computations

- Correct rounding must overcome the Table Maker's Dilemma

$$\circ(f(x) \cdot (1+\varepsilon)) \stackrel{?}{=} \circ(f(x))$$

- Finite domain, x is FP number, $x \in \mathbb{F}$: worst-case $\overline{\varepsilon}$ exists

$$\exists \overline{\varepsilon} > 0 \,.\, \forall \varepsilon, |\varepsilon| \leq \overline{\varepsilon} \,.\, \forall x \in \mathbb{F}. \circ (f(x) \cdot (1+\varepsilon)) = f(x)$$

- Univariate functions implemented in double precision:
  - Computation of $\overline{\varepsilon}$ actually possible (Lefèvre, Stehlé et al.)
  - Computation of $\overline{\varepsilon}$ is a smart exhaustive search
- Bivariate function $x^y : \mathbb{F}^2 \rightarrow \mathbb{F}$

Worst-case computations

- Correct rounding must overcome the Table Maker's Dilemma

$$\circ(f(x) \cdot (1+\varepsilon)) \stackrel{?}{=} \circ(f(x))$$

- Finite domain, x is FP number, $x \in \mathbb{F}$: worst-case $\overline{\varepsilon}$ exists

$$\exists \overline{\varepsilon} > 0 \,.\, \forall \varepsilon, |\varepsilon| \leq \overline{\varepsilon} \,.\, \forall x \in \mathbb{F}. \circ (f(x) \cdot (1+\varepsilon)) = f(x)$$

- Univariate functions implemented in double precision:
  - Computation of $\overline{\varepsilon}$ actually possible (Lefèvre, Stehlé et al.)
  - Computation of $\overline{\varepsilon}$ is a smart exhaustive search
- Bivariate function $x^y : \mathbb{F}^2 \rightarrow \mathbb{F}$
  - roughly 2^{112} valid inputs

Worst-case computations

- Correct rounding must overcome the Table Maker's Dilemma

$$\circ(f(x)\cdot(1+\varepsilon)) \overset{?}{=} \circ(f(x))$$

- Finite domain, x is FP number, $x \in \mathbb{F}$: worst-case $\overline{\varepsilon}$ exists

$$\exists \overline{\varepsilon} > 0 \,.\, \forall \varepsilon, |\varepsilon| \leq \overline{\varepsilon} \,.\, \forall x \in \mathbb{F}. \circ (f(x)\cdot(1+\varepsilon)) = f(x)$$

- Univariate functions implemented in double precision:
  - Computation of $\overline{\varepsilon}$ actually possible (Lefèvre, Stehlé et al.)
  - Computation of $\overline{\varepsilon}$ is a smart exhaustive search
- Bivariate function $x^y : \mathbb{F}^2 \rightarrow \mathbb{F}$
  - roughly 2^{112} valid inputs
  - Worst-case search of $\overline{\varepsilon}$ currently untractable

Correct rounding of x^n

- Consider x^n, $x \in \mathbb{F}$, $n \in \mathbb{N}$, n small
- Lefèvre: traditional worst-case search is possible
  - Consider each n separately
  - Current range achieved: $n \leq 255$
  - Worst case $\overline{\varepsilon} = 2^{-117}$ comparable to other double precision functions
- Correctly rounded `power(x,n)`$=\circ(x^n)$

Correct rounding of x^n

- Consider x^n, $x \in \mathbb{F}$, $n \in \mathbb{N}$, n small
- Lefèvre: traditional worst-case search is possible
  - Consider each n separately
  - Current range achieved: $n \leq 255$
  - Worst case $\overline{\varepsilon} = 2^{-117}$ comparable to other double precision functions
- Correctly rounded `power(x,n)`$=\circ(x^n)$
  - Guaranteed worst-case performance for small n

Correct rounding of x^n

- Consider x^n, $x \in \mathbb{F}$, $n \in \mathbb{N}$, n small
- Lefèvre: traditional worst-case search is possible
  - Consider each n separately
  - Current range achieved: $n \leq 255$
  - Worst case $\overline{\varepsilon} = 2^{-117}$ comparable to other double precision functions
- Correctly rounded `power(x,n)`$=\circ(x^n)$
  - Guaranteed worst-case performance for small n
  - Situation comparable to sin and cos:

Correct rounding of x^n

- Consider x^n, $x \in \mathbb{F}$, $n \in \mathbb{N}$, n small
- Lefèvre: traditional worst-case search is possible
  - Consider each n separately
  - Current range achieved: $n \leq 255$
  - Worst case $\overline{\varepsilon} = 2^{-117}$ comparable to other double precision functions
- Correctly rounded `power(x,n)`$=\circ(x^n)$
  - Guaranteed worst-case performance for small n
  - Situation comparable to sin and cos:
    - small values of n (resp. x for sin) are the most interesting
    - Ziv's rounding technique allows for correct rounding outside the known domain

Correct rounding of x^n

- Consider x^n, $x \in \mathbb{F}$, $n \in \mathbb{N}$, n small
- Lefèvre: traditional worst-case search is possible
  - Consider each n separately
  - Current range achieved: $n \leq 255$
  - Worst case $\overline{\varepsilon} = 2^{-117}$ comparable to other double precision functions
- Correctly rounded `power(x,n)`$=\circ(x^n)$
  - Guaranteed worst-case performance for small n
  - Situation comparable to sin and cos:
    - small values of n (resp. x for sin) are the most interesting
    - Ziv's rounding technique allows for correct rounding outside the known domain
- This research paves the road for x^y

Ziv's rounding technique for x^y

- Ziv's rounding technique:
  Decrease error ε of approximation $x^y \cdot (1 + \varepsilon)$ until rounding becomes possible

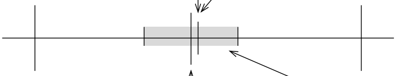
$$\circ(x^y \cdot (1 + \varepsilon)) = \circ(x^y)$$

Ziv's rounding technique for x^y

- Ziv's rounding technique:
  Decrease error ε of approximation $x^y \cdot (1+\varepsilon)$ until rounding becomes possible

$$\circ(x^y \cdot (1+\varepsilon)) = \circ(x^y)$$

- Issue:
  For ensuring termination, rounding boundary cases must be filtered out

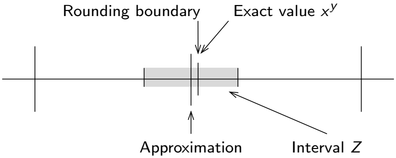


Ziv's rounding technique for x^y

- Ziv's rounding technique:
  Decrease error ε of approximation $x^y \cdot (1+\varepsilon)$ until rounding becomes possible

$$\circ(x^y \cdot (1+\varepsilon)) = \circ(x^y)$$

- Issue:
  For ensuring termination, rounding boundary cases must be filtered out



- Rounding boundary cases:
  Complex set for x^y:

$$RB = \{x^y = z | x, y \in \mathbb{F}_{53}, z \in \mathbb{F}_{54}\}$$

Previous approaches

Previous approaches:

- Rewrite

$$RB = \{x^y = z | x, y \in \mathbb{F}_{53}, z \in \mathbb{F}_{54}\}$$

as

$$x = 2^E \cdot m, \quad y = 2^F \cdot n, \quad z = 2^G \cdot k$$

$$E \cdot 2^F \cdot n = G, \quad (m^n)^{2^F \cdot n} = k$$

Previous approaches

Previous approaches:

- Rewrite

$$RB = \{x^y = z | x, y \in \mathbb{F}_{53}, z \in \mathbb{F}_{54}\}$$

  as

$$x = 2^E \cdot m, \quad y = 2^F \cdot n, \quad z = 2^G \cdot k$$

$$E \cdot 2^F \cdot n = G, \quad (m^n)^{2^F \cdot n} = k$$

- Mainly test whether

$$(m^n)^{2^F} = k$$

Previous approaches

Previous approaches:

- Rewrite

$$RB = \{x^y = z | x, y \in \mathbb{F}_{53}, z \in \mathbb{F}_{54}\}$$

  as

$$x = 2^E \cdot m, \quad y = 2^F \cdot n, \quad z = 2^G \cdot k$$

$$E \cdot 2^F \cdot n = G, \quad \left(m^n\right)^{2^F \cdot n} = k$$

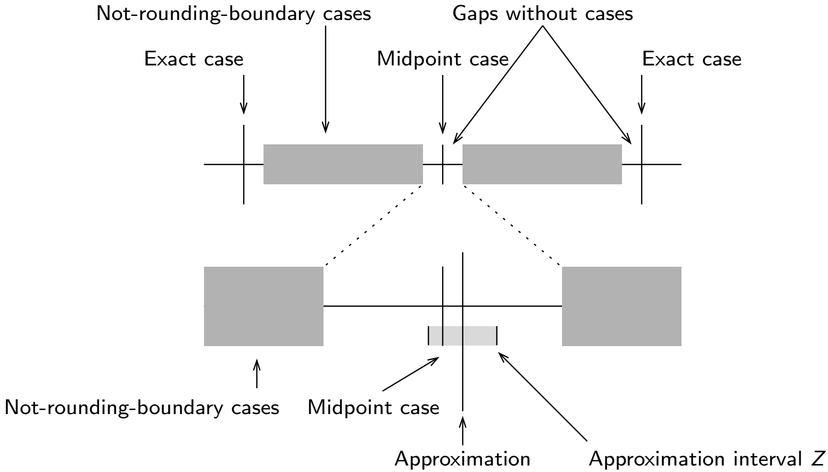
- Mainly test whether

$$\left(m^n\right)^{2^F} = k$$

- Cost of the test in double precision:
  - up to 5 square root extractions
  - up to 10 doubled precision multiplies
  - pipeline broken by many `ifs`

An efficient rounding boundary test for $x^y - 1$

Use worst-case information for rounding boundary testing



An efficient rounding boundary test for x^y – 2

- Worst-case actually unknown for x^y !

An efficient rounding boundary test for x^y – 2

- Worst-case actually unknown for x^y !
- All rounding boundary cases for x^y in double precision lie in a subset

$$\begin{aligned} \mathbb{S} &= \left\{(x, y) \in \mathbb{F}_{53}^2 \mid y \in \mathbb{N},\ 2 \leq y \leq 35\right\} \\ &\cup \Big\{(m, 2^F n) \in \mathbb{F}_{53}^2 \mid F \in \mathbb{Z},\ -5 \leq F < 0,\ n \in 2\mathbb{N}+1, \\ &\quad 3 \leq n \leq 35,\ m \in 2\mathbb{N}+1\} \end{aligned}$$

An efficient rounding boundary test for x^y – 2

- Worst-case actually unknown for x^y !
- All rounding boundary cases for x^y in double precision lie in a subset

$$\begin{aligned} \mathbb{S} &= \left\{(x,y) \in \mathbb{F}_{53}^2 \mid y \in \mathbb{N},\ 2 \leq y \leq 35\right\} \\ &\cup \left\{(m, 2^F n) \in \mathbb{F}_{53}^2 \mid F \in \mathbb{Z},\ -5 \leq F < 0,\ n \in 2\mathbb{N}+1,\right. \\ &\quad \left. 3 \leq n \leq 35,\ m \in 2\mathbb{N}+1\right\} \end{aligned}$$

- Worst-case search is tractable for $(x,y) \in \mathbb{S}$

An efficient rounding boundary test for x^y – 2

- Worst-case actually unknown for x^y !
- All rounding boundary cases for x^y in double precision lie in a subset

$$\begin{aligned}\mathbb{S} &= \left\{(x, y) \in \mathbb{F}_{53}^2 \mid y \in \mathbb{N},\ 2 \leq y \leq 35\right\} \\ &\cup \left\{(m, 2^F n) \in \mathbb{F}_{53}^2 \mid F \in \mathbb{Z},\ -5 \leq F < 0,\ n \in 2\mathbb{N}+1,\right. \\ &\qquad \left. 3 \leq n \leq 35,\ m \in 2\mathbb{N}+1\right\}\end{aligned}$$

- Worst-case search is tractable for $(x, y) \in \mathbb{S}$
- Testing if $(x, y) \in \mathbb{S}$ is easy: straightforward comparisons

An efficient rounding boundary test for x^y – 2

- Worst-case actually unknown for x^y !
- All rounding boundary cases for x^y in double precision lie in a subset

$$\begin{aligned}\mathbb{S} &= \left\{(x,y) \in \mathbb{F}_{53}^2 \mid y \in \mathbb{N},\ 2 \leq y \leq 35\right\} \\ &\cup \left\{(m, 2^F n) \in \mathbb{F}_{53}^2 \mid F \in \mathbb{Z},\ -5 \leq F < 0,\ n \in 2\mathbb{N}+1,\right. \\ &\quad \left. 3 \leq n \leq 35,\ m \in 2\mathbb{N}+1\right\}\end{aligned}$$

- Worst-case search is tractable for $(x,y) \in \mathbb{S}$
- Testing if $(x,y) \in \mathbb{S}$ is easy: straightforward comparisons
- Experimental results:

An efficient rounding boundary test for $x^y - 2$

- Worst-case actually unknown for x^y !
- All rounding boundary cases for x^y in double precision lie in a subset

$$\begin{aligned}\mathbb{S} &= \left\{(x,y) \in \mathbb{F}_{53}^2 \mid y \in \mathbb{N},\ 2 \leq y \leq 35\right\} \\ &\cup \Big\{(m, 2^F n) \in \mathbb{F}_{53}^2 \mid F \in \mathbb{Z},\ -5 \leq F < 0,\ n \in 2\mathbb{N}+1, \\ &\qquad 3 \leq n \leq 35,\ m \in 2\mathbb{N}+1\}\end{aligned}$$

- Worst-case search is tractable for $(x,y) \in \mathbb{S}$
- Testing if $(x,y) \in \mathbb{S}$ is easy: straightforward comparisons
- Experimental results:
  - 39% speed-up on average w.r.t. previous implementations

An efficient rounding boundary test for $x^y - 2$

- Worst-case actually unknown for x^y !
- All rounding boundary cases for x^y in double precision lie in a subset

$$\begin{aligned}\mathbb{S} &= \left\{(x,y) \in \mathbb{F}_{53}^2 \mid y \in \mathbb{N},\ 2 \leq y \leq 35\right\} \\ &\cup \Big\{(m, 2^F n) \in \mathbb{F}_{53}^2 \mid F \in \mathbb{Z},\ -5 \leq F < 0,\ n \in 2\mathbb{N}+1, \\ &\quad 3 \leq n \leq 35,\ m \in 2\mathbb{N}+1\}\end{aligned}$$

- Worst-case search is tractable for $(x,y) \in \mathbb{S}$
- Testing if $(x,y) \in \mathbb{S}$ is easy: straightforward comparisons
- Experimental results:
  - 39% speed-up on average w.r.t. previous implementations
  - Overhead of RB detection decreased from 50% to 9%

An efficient rounding boundary test for $x^y - 2$

- Worst-case actually unknown for x^y !
- All rounding boundary cases for x^y in double precision lie in a subset

$$\begin{aligned}\mathbb{S} &= \left\{(x,y) \in \mathbb{F}_{53}^2 \mid y \in \mathbb{N},\ 2 \le y \le 35\right\} \\ &\cup \Big\{(m, 2^F n) \in \mathbb{F}_{53}^2 \mid F \in \mathbb{Z},\ -5 \le F < 0,\ n \in 2\mathbb{N}+1, \\ &\quad 3 \le n \le 35,\ m \in 2\mathbb{N}+1\}\end{aligned}$$

- Worst-case search is tractable for $(x,y) \in \mathbb{S}$
- Testing if $(x,y) \in \mathbb{S}$ is easy: straightforward comparisons
- Experimental results:
  - 39% speed-up on average w.r.t. previous implementations
  - Overhead of RB detection decreased from 50% to 9%
  - Still more optimization: 99.1% of RB cases imply $y = \frac{3}{2}$

An efficient rounding boundary test for x^y – 3

Details can be found at

`http://prunel.ccsd.cnrs.fr/ensl-00169409/`

Automatic implementation of `libm functions`

Introduction

Correct rounding of x^y

Automatic implementation of `libm functions`

Conclusion

Function development by Arénaire members – 1

First function in `crlibm`

Function development by Arénaire members – 1

First function in `crlibm`

- `exp(x)` by David Defour

Function development by Arénaire members – 1

First function in `crlibm`

- `exp(x)` by David Defour
- correctly rounded in two approximation steps

Function development by Arénaire members – 1

First function in `crlibm`

- `exp(x)` by David Defour
- correctly rounded in two approximation steps
- portable `C` code
- integer library for second step

Function development by Arénaire members – 1

First function in `crlibm`

- `exp(x)` by David Defour
- correctly rounded in two approximation steps
- portable `C` code
- integer library for second step
- complex, hand-written proof

Function development by Arénaire members – 1

First function in `crlibm`

- `exp(x)` by David Defour
- correctly rounded in two approximation steps
- portable `C` code
- integer library for second step
- complex, hand-written proof
- duration: a Ph.D. thesis

Function development by Arénaire members – 2

An alternative implementation

Function development by Arénaire members – 2

An alternative implementation

- `exp(x)` by myself

Function development by Arénaire members – 2

An alternative implementation

- `exp(x)` by myself
- correctly rounded in one approximation step

Function development by Arénaire members – 2

An alternative implementation

- `exp(x)` by myself
- correctly rounded in one approximation step
- usage of Itanium specific features through assembler

Function development by Arénaire members – 2

An alternative implementation

- `exp(x)` by myself
- correctly rounded in one approximation step
- usage of Itanium specific features through assembler
- complex, hand-written, wrong proof

Function development by Arénaire members – 2

An alternative implementation

- `exp(x)` by myself
- correctly rounded in one approximation step
- usage of Itanium specific features through assembler
- complex, hand-written, wrong proof
- duration: a summer intern-ship at Intel Nizhny Novgorod

Function development by Arénaire members – 3

Further functions in `crlibm`: `atan(x)`, `log(x)`...

Function development by Arénaire members – 3

Further functions in `crlibm`: `atan(x)`, `log(x)`...

- Maple scripts generating header files

Function development by Arénaire members – 3

Further functions in `crlibm`: `atan(x)`, `log(x)`...

- Maple scripts generating header files
- Computation of infinite norms in Maple

Function development by Arénaire members – 3

Further functions in `crlibm`: `atan(x)`, `log(x)`...

- Maple scripts generating header files
- Computation of infinite norms in Maple
- Hand-written Gappa proofs

Function development by Arénaire members – 3

Further functions in `crlibm`: `atan(x)`, `log(x)`...

- Maple scripts generating header files
- Computation of infinite norms in Maple
- Hand-written Gappa proofs
- duration: about 1 month per function

Function development at Intel

And at Intel?

How many man-hours are accounted per `libm` function?

What is the issue?

Why is the Arénaire development process so slow?

What is the issue?

Why is the Arénaire development process so slow?

Actually, I thought we were always doing the same things...

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

- Analyze the behaviour of f in $[a, b]$

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

- Analyze the behaviour of f in $[a, b]$
- Find an appropriate range reduction

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

- Analyze the behaviour of f in $[a, b]$
- Find an appropriate range reduction
- Compute an approximation polynomial p^*

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

- Analyze the behaviour of f in $[a, b]$
- Find an appropriate range reduction
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

- Analyze the behaviour of f in $[a, b]$
- Find an appropriate range reduction
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

- Analyze the behaviour of f in $[a, b]$
- Find an appropriate range reduction
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

- Analyze the behaviour of f in $[a, b]$
- Find an appropriate range reduction
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof
- Check the proof for mistakes

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

- Analyze the behaviour of f in $[a, b]$
- Find an appropriate range reduction
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof
- Check the proof for mistakes
- Bound and proof the approximation error: $\| \frac{p-f}{f} \|_\infty$

Steps in the implementation of a function

Task: implement f in a domain $[a, b]$ with an accuracy of k bits

- Analyze the behaviour of f in $[a, b]$
- Find an appropriate range reduction
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof
- Check the proof for mistakes
- Bound and proof the approximation error: $\|\frac{p-f}{f}\|_\infty$
- Integrate everything

A prototype toolchain – 1

A prototype, automatic toolchain for the implementation process

A prototype toolchain – 1

A prototype, automatic toolchain for the implementation process

- Joint work by
  - S. Chevillard (floating-point polynomial approximation part)
  - Ch. Lauter (implementation and proof part)
  - G. Melquiond (Gappa)
  - and other Arénaire members

A prototype toolchain – 1

A prototype, automatic toolchain for the implementation process

- Joint work by
  - S. Chevillard (floating-point polynomial approximation part)
  - Ch. Lauter (implementation and proof part)
  - G. Melquiond (Gappa)
  - and other Arénaire members
- Written in
  - Pari/GP
  - `C`, `C++`
  - Shell scripts
  - an internal language: `arenaireplot`

A prototype toolchain – 1

A prototype, automatic toolchain for the implementation process

- Joint work by
  - S. Chevillard (floating-point polynomial approximation part)
  - Ch. Lauter (implementation and proof part)
  - G. Melquiond (Gappa)
  - and other Arénaire members
- Written in
  - Pari/GP
  - `C`, `C++`
  - Shell scripts
  - an internal language: `arenaireplot`
- Targetted to
  - portable `C` implementations
  - using double, double-double and triple-double arithmetic
  - with easy-to-handle Horner evaluation

A prototype toolchain – 2

Automatic handling of the following sub-problems:

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation
- Compute an approximation polynomial p^*

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof
- Check the proof for errors

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof
- Check the proof for errors
- Bound and proof the approximation error: $\|\frac{p-f}{f}\|_\infty$

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof
- Check the proof for errors
- Bound and proof the approximation error: $\|\frac{p-f}{f}\|_\infty$

Missing parts:

- Analyze the behaviour of f in $[a, b]$

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof
- Check the proof for errors
- Bound and proof the approximation error: $\| \frac{p-f}{f} \|_\infty$

Missing parts:

- Analyze the behaviour of f in $[a, b]$
- Find a range reduction using tables etc.

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range translation
- Compute an approximation polynomial p^*
- Bring the coefficients of p^* into floating-point form: p
- Implement p in floating-point arithmetic
- Bound round-off errors, write a proof
- Check the proof for errors
- Bound and proof the approximation error: $\left\| \frac{p-f}{f} \right\|_\infty$

Missing parts:

- Analyze the behaviour of f in $[a, b]$
- Find a range reduction using tables etc.
- Integrate everything

Demonstration

Task: Implement

$$f(x) = e^{\cos x^2 + 1}$$

in the interval

$$I = [-2^{-8}; 2^{-5}]$$

with at least 66 bits of accuracy

Demonstration

Task: Implement

$$f(x) = e^{\cos x^2 + 1}$$

in the interval

$$I = [-2^{-8}; 2^{-5}]$$

with at least 66 bits of accuracy

Let' try it out...

Results on new functions

Last functions in `crlibm`

Results on new functions

Last functions in `crlibm`

- `sinpi(x)`, `cospi(x)`, `tanpi(x)`

Results on new functions

Last functions in `crlibm`

- `sinpi(x)`, `cospi(x)`, `tanpi(x)`
- correctly rounded in two approximation steps

Results on new functions

Last functions in `crlibm`

- `sinpi(x)`, `cospi(x)`, `tanpi(x)`
- correctly rounded in two approximation steps
- both evaluation codes generated automatically

Results on new functions

Last functions in `crlibm`

- `sinpi(x)`, `cospi(x)`, `tanpi(x)`
- correctly rounded in two approximation steps
- both evaluation codes generated automatically
- duration: two days

And Intel's customers ?

Could this be interesting for Intel's customers?

- Faster-to-market and cheaper implementations ?

And Intel's customers ?

Could this be interesting for Intel's customers?

- Faster-to-market and cheaper implementations ?
- Easier approach to Gappa usage ?

And Intel's customers ?

Could this be interesting for Intel's customers?

- Faster-to-market and cheaper implementations ?
- Easier approach to Gappa usage ?
- Better maintainablity of some code parts ?

And Intel's customers ?

Could this be interesting for Intel's customers?

- Faster-to-market and cheaper implementations ?
- Easier approach to Gappa usage ?
- Better maintainablity of some code parts ?
- Compilers that inline composite functions like $e^{\cos x^2+1}$?

Conclusion

Introduction

Correct rounding of x^y

Automatic implementation of `libm functions`

Conclusion

Conclusion and outlooks...

- More correctly rounded functions:

Conclusion and outlooks...

- More correctly rounded functions:
  - High performance on average can be achieved for $\circ(x^y)$

Conclusion and outlooks...

- More correctly rounded functions:
  - High performance on average can be achieved for $\circ(x^y)$
  - Worst case bounding might become feasible for x^y:
    a certificate that 2500 bits suffice for double seems to cost about 500 machine-years

Conclusion and outlooks...

- More correctly rounded functions:
  - High performance on average can be achieved for $\circ(x^y)$
  - Worst case bounding might become feasible for x^y:
    a certificate that 2500 bits suffice for double seems to cost about 500 machine-years
- Attacking double-extended precision:

Conclusion and outlooks...

- More correctly rounded functions:
  - High performance on average can be achieved for $\circ(x^y)$
  - Worst case bounding might become feasible for x^y:
    a certificate that 2500 bits suffice for double seems to cost about 500 machine-years
- Attacking double-extended precision:
  - Worst-case search would be possible for univariate functions

Conclusion and outlooks...

- More correctly rounded functions:
  - High performance on average can be achieved for $\circ(x^y)$
  - Worst case bounding might become feasible for x^y:
    a certificate that 2500 bits suffice for double seems to cost about 500 machine-years
- Attacking double-extended precision:
  - Worst-case search would be possible for univariate functions
  - We have tools for simplifying the implementation process

Conclusion and outlooks...

- More correctly rounded functions:
  - High performance on average can be achieved for $\circ(x^y)$
  - Worst case bounding might become feasible for x^y: a certificate that 2500 bits suffice for double seems to cost about 500 machine-years
- Attacking double-extended precision:
  - Worst-case search would be possible for univariate functions
  - We have tools for simplifying the implementation process
- More numerical knowlegde inside high-level compilers

Conclusion and outlooks...

- More correctly rounded functions:
  - High performance on average can be achieved for $\circ(x^y)$
  - Worst case bounding might become feasible for x^y: a certificate that 2500 bits suffice for double seems to cost about 500 machine-years
- Attacking double-extended precision:
  - Worst-case search would be possible for univariate functions
  - We have tools for simplifying the implementation process
- More numerical knowlegde inside high-level compilers
  - Remove the numerical burden from low-level `C/Fortran`

Conclusion and outlooks...

- More correctly rounded functions:
  - High performance on average can be achieved for $\circ(x^y)$
  - Worst case bounding might become feasible for x^y: a certificate that 2500 bits suffice for double seems to cost about 500 machine-years
- Attacking double-extended precision:
  - Worst-case search would be possible for univariate functions
  - We have tools for simplifying the implementation process
- More numerical knowlegde inside high-level compilers
  - Remove the numerical burden from low-level `C/Fortran`
  - Numerical algorithms described in a high-level language

Conclusion and outlooks...

- More correctly rounded functions:
  - High performance on average can be achieved for $\circ(x^y)$
  - Worst case bounding might become feasible for x^y: a certificate that 2500 bits suffice for double seems to cost about 500 machine-years
- Attacking double-extended precision:
  - Worst-case search would be possible for univariate functions
  - We have tools for simplifying the implementation process
- More numerical knowlegde inside high-level compilers
  - Remove the numerical burden from low-level `C/Fortran`
  - Numerical algorithms described in a high-level language
  - Highly investigated by Arénaire
- Need: more and more computational power

Thank you!

Thank you for your attention !

Questions ?