Towards the automatic implementation of `libm` functions

Presentation at Intel - Nizhniy Novgorod

Florent de Dinechin
Christoph Quirin Lauter

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

Nizhniy Novgorod, 30 july 2007

History of `libm` function development

History of `libm` function development

Automatization of the implementation process

Let's try it out...

Conclusions

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

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...

Automatization of the implementation process

History of `libm` function development

Automatization of the implementation process

Let's try it out...

Conclusions

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 Remez 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 Remez 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 Remez 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 reduction (trivial cases)

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range reduction (trivial cases)
- Compute an approximation polynomial p^*

A prototype toolchain – 2

Automatic handling of the following sub-problems:

- Find an appropriate range reduction (trivial cases)
- 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 reduction (trivial cases)
- 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 reduction (trivial cases)
- 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 reduction (trivial cases)
- 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 reduction (trivial cases)
- 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 reduction (trivial cases)
- 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 reduction (trivial cases)
- 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 reduction (trivial cases)
- 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.
- Integrate everything

Let's try it out...

History of `libm` function development

Automatization of the implementation process

Let's try it out...

Conclusions

Demonstration

Task: Implement

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

in the interval

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

with at least 62 bits of accuracy

Demonstration

Task: Implement

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

in the interval

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

with at least 62 bits of accuracy

Let' try it out...

Conclusions

History of `libm` function development

Automatization of the implementation process

Let's try it out...

Conclusions

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 $e^{\cos x^2+1}$?

Thank you!

Thank you for your attention !

Questions ?