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

Automatization of the implementation process

Let’s try it out...

Conclusions
First function in crlibm
First function in crlibm

- \( \exp(x) \) by David Defour
First function in crlibm

- \( \exp(x) \) by David Defour
- correctly rounded in two approximation steps
First function in crlibm

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

- $\exp(x)$ by myself
An alternative implementation

- $\exp(x)$ by myself
- correctly rounded in one approximation step
An alternative implementation

- \( \exp(x) \) by myself
- correctly rounded in one approximation step
- usage of Itanium specific features through assembler
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
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 internship at Intel
Further functions in crlibm: \texttt{atan(x)}, \texttt{log(x)}...
Function development by Arénaire members – 3

Further functions in crlibm: \( \text{atan}(x) \), \( \text{log}(x) \)...

- Maple scripts generating header files
Further functions in crlibm: \( \text{atan}(x), \text{log}(x) \)...

- Maple scripts generating header files
- Computation of infinite norms in Maple
Further functions in crlibm: $\text{atan}(x)$, $\log(x)$...

- Maple scripts generating header files
- Computation of infinite norms in Maple
- Hand-written Gappa proofs
Further functions in crlibm: \( \text{atan}(x), \log(x) \)...  
- Maple scripts generating header files  
- Computation of infinite norms in Maple  
- Hand-written Gappa proofs  
- **duration: about 1 month per function**
And at Intel?

How many man-hours are accounted per \texttt{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 $[a, b]$
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: $p$

Towards automatic implementation of libm functions - Intel Nizhny Novgorod - 30 July 2007
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, automatic toolchain for the implementation process
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, 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:

- Find an appropriate range reduction (trivial cases)
- Compute an approximation polynomial $p$
- Bring the coefficients of $p$ into floating-point form
- Implement $p$ in floating-point arithmetic
- Bound round-off errors, write a proof
- Check the proof for errors
- Bound and proof the approximation error: $\|p - f\|_\infty$

Missing parts:
- Analyze the behaviour of $f$ in $[a, b]$
- Find a range reduction using tables etc.
- Integrate everything
Automatic handling of the following sub-problems:

- Find an appropriate range reduction (trivial cases)
Automatic handling of the following sub-problems:

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

Missing parts:

- Analyze the behaviour of $f$ in $[a, b]$
- Find a range reduction using tables etc.
- Integrate everything
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: $p$
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
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
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
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: $\| \frac{p-f}{f} \|_{\infty}$
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 $[a, b]$
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.
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
Task: Implement

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

in the interval

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

with at least 62 bits of accuracy
**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

- \( \sin \pi(x), \cos \pi(x), \tan \pi(x) \)
Results on new functions

Last functions in crlibm

- \textit{sinpi(x)}, \textit{cospi(x)}, \textit{tanpi(x)}
- correctly rounded in two approximation steps
Results on new functions

Last functions in crlibm

- \( \sin \pi(x), \cos \pi(x), \tan \pi(x) \)
- correctly rounded in two approximation steps
- both evaluation codes generated automatically
Results on new functions

Last functions in crlibm

- \texttt{sinpi(x)}, \texttt{cospi(x)}, \texttt{tanpi(x)}
- correctly rounded in two approximation steps
- both evaluation codes generated automatically
- \textit{duration: two days}
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 maintainability of some code parts?
Could this be interesting for Intel’s customers?

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

Questions?