Accurate Math Functions on the Intel IA-32 Architecture: A Performance-Driven Design

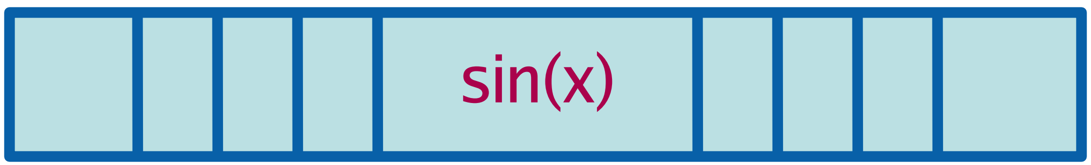
Cristina Anderson, Nikita Astafiev and Shane Story
Intel Corporation
July 2006

Agenda

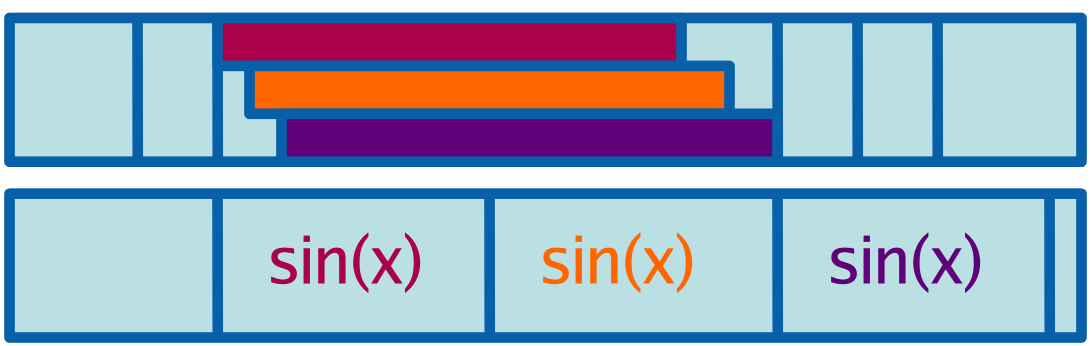
- Intel LIBM design
  - Goals
  - Techniques
  - Verification
- Implementation tricks for gaining
  - Performance
  - Accuracy
- Current results
  - Performance and Accuracy tables
- Future development
  - Challenges of coding in C
  - Correctly rounded

Intel LIBM design

- Goals
  - Performance: Latency optimized routines
    - Given the maximum resources function should finish execution as soon as possible

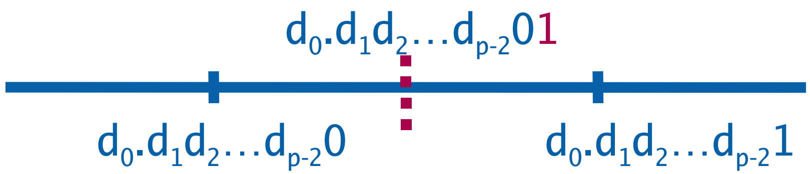


    - Opposed to Through-put optimization: maximum number of independent functions run in parallel – each is constrained in resources



Intel LIBM design

- Goals
  - Performance: Latency optimized routines
  - Accuracy: 0.55 ulps (units in the last place) or better in round-to-nearest



$$\text{Rounding error} = |b^{exp}\cdot \underbrace{d_0.d_1d_2\ldots d_{p-1}}_{\text{p-digits format}} - b^{exp}\cdot d_0.d_1d_2\ldots \underbrace{d_{p-1}dddddddd\ldots}_{\text{Exact base b representation}}|\cdot b^{p-1-exp}$$

Intel LIBM design

- Goals
  - Performance: Latency optimized routines
  - Accuracy: 0.55 ulps or better in round-to-nearest
  - Correctly rounded in IEEE-754 mandated cases
  - Exception flags
  - C99, F90 conformance
  - Structured Exception Handling

Intel LIBM design

- Algorithms
  - Argument reduction
  - Table-lookup
  - Polynomial approximation
  - Reconstruction

$$\log(2^n \cdot M) = n\cdot\log 2 + \log M = n\cdot\log 2 - \log(B) + \log(B\cdot M)$$

$$= n\cdot\log 2 - \log(B) + \log(1 + (B\cdot M - 1)) \qquad B \approx 1/M$$

$$\approx n\cdot T_{\log 2} - T_{\log(1/M)} + \mathrm{Poly}(r)$$

$$= n\cdot(T_{\log 2\_hi} + T_{\log 2\_lo}) - T(\mathrm{index}_{\log(1/M)}) + r\cdot(P_1 + r\cdot(P_2\ldots)\ldots)$$

Intel LIBM design

- Architecture
  - IA-32 instructions set
    - General Purpose
      - 32-bit GP registers
    - x87 FPU
      - 80-bit FP registers

General Purpose	
Arithmetic	ADD, SUB
Logical	OR, AND, XOR
Shifts	SHL, SHR, SAR
Tests	CMP, TEST
Loads	MOV

x87 FPU	
Load	FLD
Arithmetic	FADD, FMUL
Remainder	FPREM1

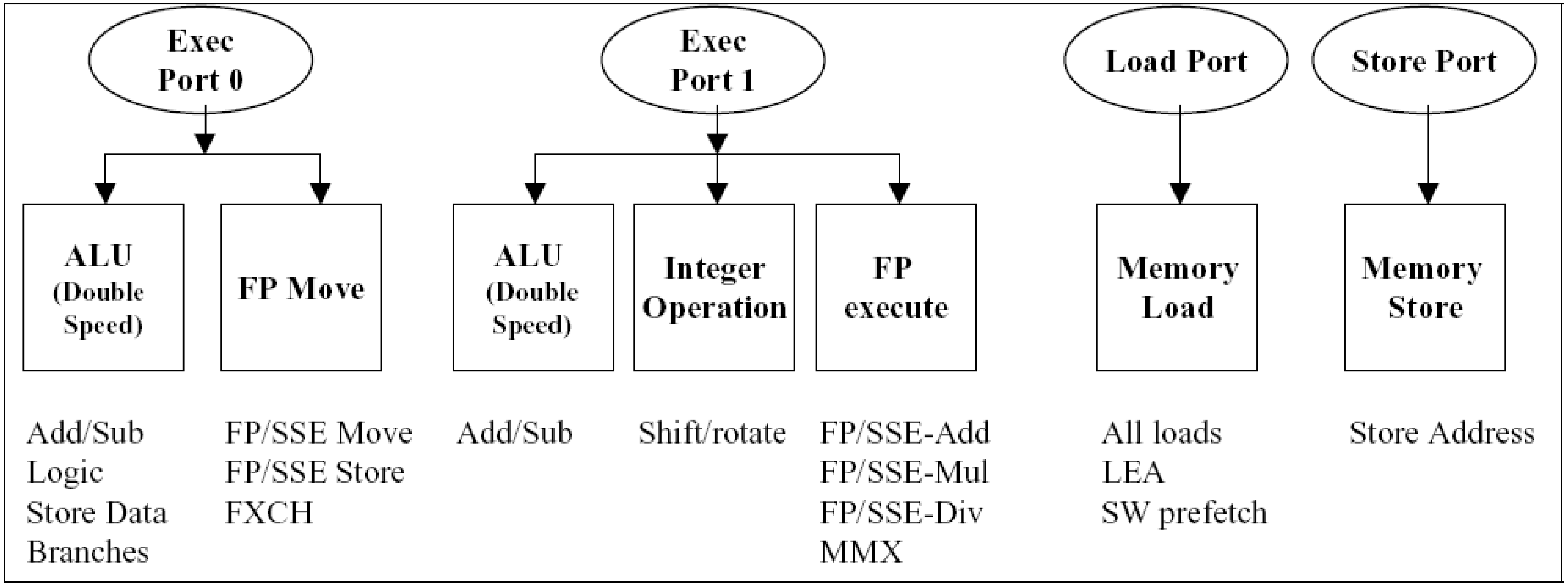
Intel LIBM design

- Architecture
  - IA-32 instructions set
    - General Purpose
    - x87 FPU
    - SSE/SSE2/SSE3…
      - 128-bit registers
      - 2 double FP
      - 4 single FP

SSE/SSE2/SSE3	Double Precision	Single Precision
FP SIMD/scalar add, subtract	ADDPD/SD SUBPD/SD	ADDPS/SS SUBPS/SS
FP SIMD/scalar multiply, divide	MULPD/SD DIVPD/SD	MULPS/SS DIVPS/SS
Logical	ORPD, ANDPD, XORPD	ORPS, ANDPS, XORPS
Load one element Load 128 bit	MOVSD MOVAPD	MOVSS MOVAPS
Pack	MOVDDUP	
Pack / unpack	PSHUFD	
Transfers to / from GPR	MOVD, PEXTRW, PINSRW	
Format conversions	CVTSS2SD, CVTSD2SS	
Logical shifts	PSRLQ, PSLLQ, PSRLD, PSLLD	

Intel LIBM design

- Architecture
  - IA-32 instructions set
    - General Purpose
    - x87 FPU
    - SSE/SSE2/SSE3…
  - Instruction-level parallelism



Intel LIBM design

- Architecture
  - IA-32 instructions set
    - General Purpose
    - x87 FPU
    - SSE/SSE2/SSE3…
  - Instruction-level parallelism
  - SIMD parallelism

$$P(x) = (\ldots((P_9 \cdot x + P_8) \cdot x + P_7) \cdot x + \ldots + P_0$$

$$\begin{cases} P_{odd} = (((P_9 \cdot x^2 + P_7) \cdot x^2 + P_5) \cdot x^2 + P_3) \cdot x^2 + P_1 \\ P_{even} = (((P_8 \cdot x^2 + P_6) \cdot x^2 + P_4) \cdot x^2 + P_2) \cdot x^2 + P_0 \end{cases}$$

$$P(x) = P_{odd} \cdot x + P_{even}$$

Intel LIBM design

- Verification
  - Paper proofs are limited
    - IEEE required cases (sqrt, division, remainder)
  - IMLTS (Intel Math Library Test Suite)
    - Black and White box testing
      - Accuracy / Monotonicity / Symmetry / Special Values / Flags
      - Hard to round cases
    - Errors have been found in the 3 available CR libraries

Implementation tricks
Branch collapsing

Branch collapsing via bit-mask	
if (x > b) $\quad y = y + a$	$m_a = (b - x) \gg 31$ $a_1 = m_a \ \& \ a$ $y = y + a_1$

Implementation tricks
Branch collapsing

Branch collapsing via condition shrinking	
Let a > b	

if (x > a)
        goto overflow
if (x < b)
        goto underflow
…
<main path>
…
	if ((unsigned) (x-b) > a-b)
	        goto special
	…
	<main path>
	…

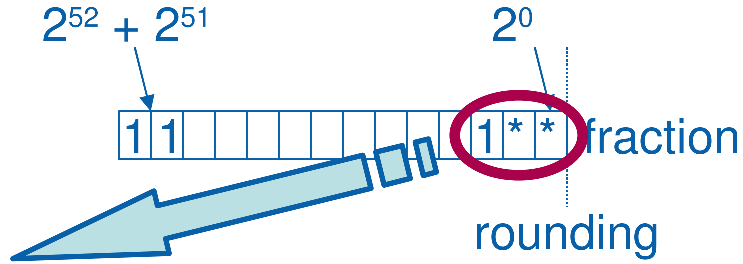
Implementation tricks
Fast conversions

Fast conversion to integer via Right Shifter

Let double $|x| < 2^{51}$

cvtsd2si	y, x	addsd	x, $2^{52} + 2^{51}$
cvtsi2sd	r, y	subsd	x, $2^{52} + 2^{51}$

Result is represented in double FP format



movd index, x

Implementation tricks

Fast Single to Double conversion

Let single $x > 0$

$$x = 2^{(E-127)} \cdot \text{significand}$$

cvtss2sd y, x	psllq x, 52 – 23 paddd x, 0x3800000000000000

$$y = 2^{(E-1023)} \cdot \text{significand}$$

Fast integer ABS

$N = 32$ or 64

$S = x >> (N - 1)$	$S = \begin{cases} 0000\ldots, & \text{if } x >= 0 \\ 1111\ldots, & \text{if } x < 0 \end{cases}$
$y = (x + S)$ xor S	$y = \lvert x \rvert$

Implementation tricks
Accuracy and performance

- arcsinh(x) $= \ln(x + \mathrm{sqrt}(x^2+1)) = \ln(1 + (x + \mathrm{sqrt}(x^2+1) - 1))$
  $= \ln(1 + V),\ V >= 0$
- Relative error

$$\frac{dy}{y} = \frac{d(\ln(1+V))}{\ln(1+V)} = \frac{dV}{(1+V)\cdot\ln(1+V)} < \frac{dV}{V} \qquad V > 0$$

- Multiprecision computations
  - Newton iterations for sqrt and argument reduction
  - Polynomial reconstruction of log (lower terms)

A + B != round to double (A + B)
Let |A| > |B|
S_{hi} = round to double (A + B)
$S_{lo} = (A - S_{hi}) + B$
$A + B = S_{hi} + S_{lo}$

Sometimes it is possible to make the lower terms of the polynomial simple – to have less high-low parts computations

Current results: Accuracy

Function	Intel libm	GNU libm	CRlibm (first step)	CRlibm
SIN	0.515082	2.60E+33	0.5	0.5
COS	0.51851	2.60E+33	0.500001	0.5
TAN	0.541852	2.60E+33	0.500001	0.5
ASIN	0.535745	3.440871	0.50157	0.5
ACOS	0.531348	2.15E+05		
ATAN	0.542333	0.500307	0.5	0.5
EXP	0.540348	0.787214	0.500047	0.5
LOG	0.501397	0.500376	0.500583	0.5
POW	0.506688	8.48E+08		
SINF	0.513276	1.13E+15		
COSF	0.509615	3.52E+13		
TANF	0.504488	7.04E+13		
ASINF	0.500003	0.5		
ACOSF	0.500003	0.5		
ATANF	0.520918	0.5		
EXPF	0.506582	0.5		
LOGF	0.502916	0.5		
POWF	0.501025	0.5		

Intel 9.1 compiler LIBM: Intel(R) C++ Compiler for 32-bit applications, Version 9.1 Build 20060505Z
glibc 2.3.6 built on SUSE* SLES9 with gcc 3.3.3
Modified CRLIBM* 0.11beta1, built on SUSE* SLES9 with gcc 3.3.3
Performance tests and ratings are measured using specific computer systems and/or components. Any difference in system design or configuration may affect actual performance.

Current results: Performance (clock cycles)

Function	Intel libm	GNU libm	CRlibm (first step)	CRlibm
SIN	164	258	460	21406
COS	164	256	455	20468
TAN	270	304	732	47194
ASIN	203	498	1244	3542
ACOS	206	498		
ATAN	155	338	730	25907
EXP	145	444	528	2548
LOG	159	342	365	1505
POW	249	665		
SINF	66	231		
COSF	65	232		
TANF	86	283		
ASINF	82	357		
ACOSF	84	357		
ATANF	69	302		
EXPF	48	354		
LOGF	57	197		
POWF	115	594		

Intel 9.1 compiler LIBM: Intel(R) C++ Compiler for 32-bit applications, Version 9.1 Build 20060505Z
glibc 2.3.6 built on SUSE* SLES9 with gcc 3.3.3
Modified CRLIBM* 0.11beta1, built on SUSE* SLES9 with gcc 3.3.3
Performance tests and ratings are measured using specific computer systems and/or components. Any difference in system design or configuration may affect actual performance.

Future Development

- Code in C
  - Ease of use and support
    - In-lining
    - Portability
  - Some architectural features are inaccessible from the high-level programming language (e.g. logical operations on FP numbers)
    - Compiler built-ins (extensions to language) can help
  - Dependency on the compiler
- Correctly rounded
  - Currently out of scope
  - Challenge: accuracy (consistency) vs performance
  - Proof is the main problem: e.g. 3 publicly available CR libraries have had accuracy issues
    - IBM Ultimate LIBM*. 1999, 2002.
    - Sun LIBMCR*. Ver. 0.9; December 2004.
    - ENS-Lyon CRLIBM*. Ver. 0.8beta; January 2005

Thank You!