Optimizing Scientific Libraries for the Itanium

John Harrison
Intel Corporation

Gelato Federation Meeting, HP Cupertino

May 25, 2005

Quick summary

Intel supplies ‘drop-in replacement’ versions of common libraries optimized for particular (micro-)architectures.

We’ll focus on our version of the standard math library `libm` for the Intel® Itanium® architecture.

Provides superior speed and/or accuracy to typical generic versions (FDLIBM etc.)

Special features of Itanium make for some particularly high-quality implementations.

What makes Itanium special?

- Parallelism, many registers, predication and explicit scheduling — can consider sophisticated algorithms with parallel threads
- Fused multiply-add and extended precision — useful for polynomials and eases many accuracy-preserving techniques
- Non-atomic division and square root — allows optimized integration into more complicated algorithms
- Multiple status fields — intermediate computations can use higher intermediate precision with no performance hit
- Predication and register renaming — can eliminate most control flow and pipeline multiple instances

“Naive” table-driven algorithm

A naive algorithm for e^x illustrates the key steps of the typical table-driven algorithm:

- Reduction: split $x = N + r$ where N is an integer and $|r| \leq 1/2$.
- Approximation: $e^r = 1 + r + r^2/2! + \cdots r^n/n!$
- Reconstruction: $e^x = e^N e^r$ with e^N taken from a table.

Same pattern is used in most real algorithms, but with careful attention to numeric accuracy and the tradeoff between speed and table size.

Real table-driven algorithm

- The input number X is first reduced to r with approximately $|r| \leq \pi/4$ such that $X = r + N\pi/2$ for some integer N. We now need to calculate $\pm tan(r)$ or $\pm cot(r)$ depending on N modulo 4.
- If the reduced argument r is still not small enough, it is separated into its leading few bits B and the trailing part $x = r - B$, and the overall result computed from $tan(x)$ and pre-stored functions of B, e.g.

$$tan(B + x) = tan(B) + \frac{\frac{1}{sin(B)cos(B)}tan(x)}{cot(B) - tan(x)}$$

- Now a polynomial (plus reciprocal) approximation is used for $tan(r)$, $cot(r)$ or $tan(x)$ as appropriate.

Polynomials using `fma`

A traditional approach to computing polynomials is “Horner’s rule”:

$$a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + xa_n) \cdots)))$$

Minimizes the total number of operations and usually has good numerical properties.

Even better on Itanium. The core FP operation is the *fused multiply-add*, which computes $a \cdot b + c$ in one operation.

Horner’s rule is used in most of the *throughput-optimized* algorithms.

Binary splitting of polynomials

If our concern is *latency*, we can do better using binary splitting. This arrangement yields 3 `fma` latencies instead of 6:

$$1 + d + d^2 + \cdots + d^6 = 1 + (1 + d + d^2)(d^4 + d)$$

More generally, we can do most n-degree polynomials in around $log_2(n)$ latencies, thanks to pipelining and two floating-point units.

Using extended precision, there are few accuracy worries with algebraic rearrangement. (Need a little care over monotonicity!)

So our algorithms often use longer polynomials than the norm, given that they can be computed quickly and accurately.

Exploiting non-atomic division

On Itanium, there's no atomic division operation, but `frcpa` returns a reciprocal approximation good to about 8 bits.

Techniques largely due to Markstein allow this to be refined to an IEEE-correct quotient just using standard `fma` operations.

- Automatically inherits pipelining from core operations, giving high throughput and ability to schedule in parallel with other work
- Simpler hardware
- Can more flexibly use specially optimized algorithms, e.g. not force IEEE rounding of intermediate results
- Can exploit the reciprocal approximation itself for various tricks

Using parallelism and non-atomic division

In computing atan(x) for $x > 1$ we actually compute $\pi/2 - atan(1/x)$. It seems we have a division in the critical path.

But eventually we are computing a polynomial in $1/x$, and we can factor out the highest term:

$$p(1/x) = 1/x^{45} q(x)$$

Now let $c =$ frcpa(x) and $b = c \cdot x - 1$. We can actually compute the following, where all three terms in the product can be evaluated in parallel

$$p(1/x) = c^{45} r(b) q(x)$$

where $r(b)$ is a power series expansion of $\frac{1}{(1+b)^{45}}$.

The `frcpa` trick

Though designed for starting a division, `frcpa` can speed up range reduction for some multiplicative functions.

Let $c = \text{frcpa}(x)$ and $r = c \cdot x - 1$. Then:

$$ln(x) = ln((1 + r)/c) = ln(1 + r) - ln(c)$$

We have a precomputed table of values for $ln(c)$, and $ln(1 + r)$ is approximated by a short power series.

Thanks to this, the logarithm is about the fastest of our transcendentals!

Some results

Figures for `libm` latency and `VML` throughput, double precision.

Function	Latency	Throughput	Accuracy (ulps)
$atan$	56	13.3	0.501
$cbrt$	34	10.2	0.501
exp	43	6.2	0.502
log	31	11.2	0.501
sin	49	8.2	0.503
tan	63	11.3	0.507

All figures for double precision “common path”. Some cases may be quicker ($\exp(0)$) or slower ($\sin(2^{1000})$).

Accuracy figures are for `libm`; those for VML may differ slightly.

Conclusions

Using Itanium's architectural features to the full, we can provide a `libm` with industry-leading speed and accuracy.

Almost every special architectural feature of Itanium is fully exploited here, occasionally in unexpected ways.

Some current work:

- Investigate perfectly rounded transcendental functions (with reasonable performance)
- Fill out functionality with high-quality versions of more obscure functions (e.g. Bessel functions $J_n(x)$ and $Y_n(x)$).

Further reading

For quick surveys:

New Algorithms for Improved Transcendental Functions on IA-64, Shane Story and Peter Tang, 14th IEEE Computer Arithmetic Conference, 1999.

The Computation of Transcendental Functions on the IA-64 Architecture, John Harrison, Ted Kubaska, Shane Story and Peter Tang, Intel Technology Journal Q4 1999.

Much more detailed information:

IA-64 and elementary functions: speed and precision, Peter Markstein, Hewlett-Packard Professional Books, 2000.

Scientific Computing on Itanium-based systems, Marius Cornea, John Harrison and Peter Tang, Intel Press 2003.