New Floating-Point Programming Opportunities with HP-UX on Itanium™

James W Thomas

Hewlett-Packard Company

Cupertino, CA

PH: 408 447 5781

FAX: 408 447 4924

jwthomas@cup.hp.com

Peter Markstein

Hewlett-Packard Company

Palo Alto, CA

PH: 650 857 6662

FAX: 650 857 5542

peter_markstein@hp.com

New Features in HP-UX 11iV1.5 for Itanium

- C99/IEEE 754 floating-point model
- 4 fully supported floating-point types
- Expanded function Library
- Better performance, accuracy, robustness [1]
- Wide expression evaluation
- Intelligible user controls for FP behavior and performance
- Complex and imaginary types in C

New Programming Opportunities

- Use libm functions for building blocks
- Use wide types for robustness
- Use FMA for accuracy and performance
- Use controls for to trade off FP behavior and performance
- Use IEEE 754 features for simplicity and efficiency
- Use C complex for natural coding style and efficiency

C/C++ Math Library Functions [1], [2]

- 4 fully supported floating types (32, 64, 80, 128 bits)

math

C89…

acos asin atan atan2 cos sin tan cosh sinh tanh exp frexp ldexp log log10 modf pow sqrt ceil fabs floor fmod

Unix standard…

erf erfc gamma lgamma hypot isnan acosh asinh atanh cbrt expm1 ilogb log1p logb nextafter remainder rint scalb j0 j1 jn y0 y1 yn

C99..

isnan isinf signbit isfinite isnormal fpclassify isunordered isgreater isgreaterequal isless islessequal islessgreater copysign log2 exp2 fdim fmax fmin nan scalbn scalbln nearbyint round trunc remquo lrint lround llrint llround fma nexttoward

HP-UX…

annuity compound lgamma_r exp10 cosd sind tand acosd asind atand atan2d

complex (C99)

cacos casin catan ccos csin ctan cacosh casinh catanh ccosh csinh ctanh cexp clog csqrt cabs cpow carg conj cimag cproj creal

fenv

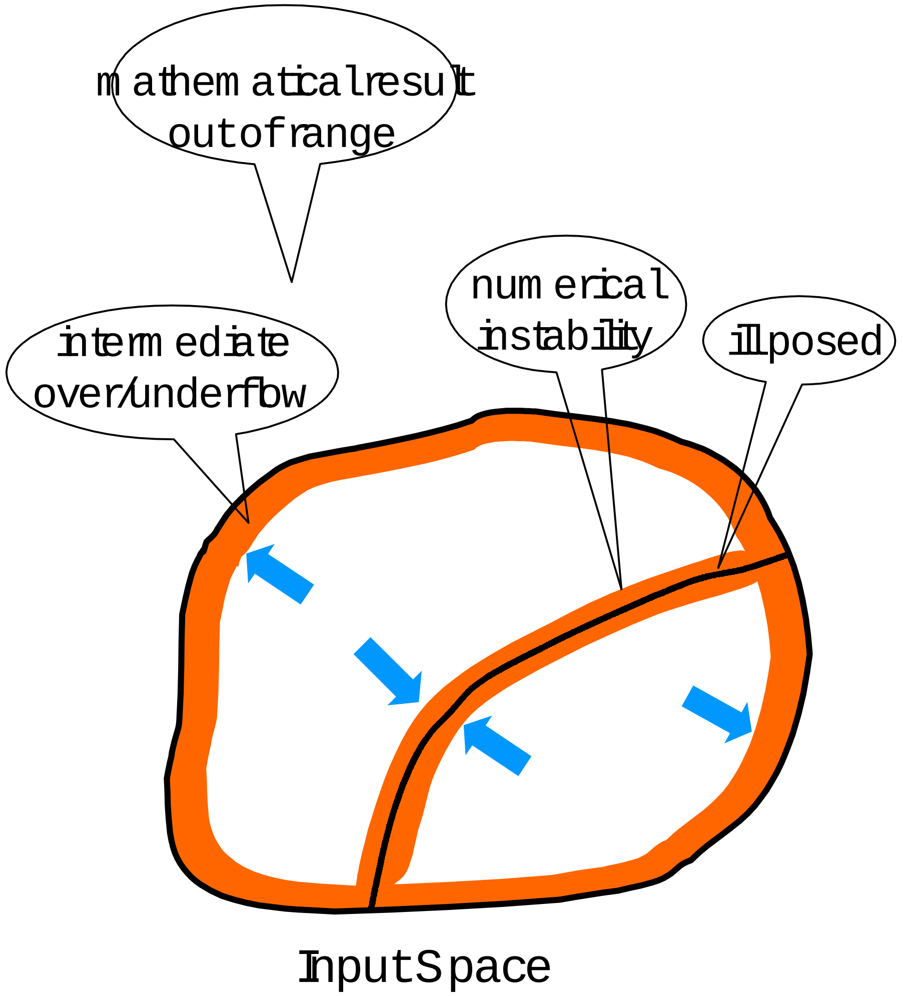
C99…

feclearexcept fegetexcepflag feraiseexcept fesetexceptflag fetestexcept fegetround fesetround fegetenv feholdexcpet fesetenv feupdateenv

HP-UX…

fegetflushtozero fesetflushtozero fegettrapenable fesettrapenable

More robust code delivers useful results for a greater range of inputs



- "Eliminate" intermediate overflow and underflow
- Shrink regions of numerical instability
- error ~ (precision roundoff) / (distance from ill-posed)

HP-UX/Itanium Wide FP Types [3]

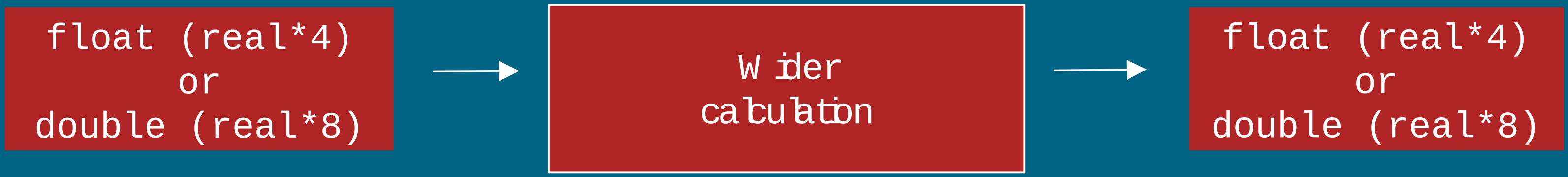
float (real*4)
- full HW support
- precision: 24 bits
- range: 8 bit exponent

double (real*8)
- full HW support
- precision: 53 bits
- range: 11 bit exponent

extended
- full HW support
- precision: 64 bits
- range: 15 bit exponent
- flop speed: = double
- lib func speed: ~0.7X double
- C, C++ compiler/library support

long double (quad, real*16)
- SW implementation utilizes IPF features
- precision: 113 bits
- range: 15 bit exponent
- lib func speed: 0.25X extended
- compiler/library support

Use Wide Types Inside



- 11 extra bits of precision means round-off problems are 2000 times less likely
- 4 extra exponent bits usually eliminates intermediate overflow and underflow

HP-UX Support for Wide Types

- Nomenclature for wide types

```
printf("%hLe\n", logw(EXT_MAX)); // extended
printf("%lLe\n", logq(QUAD_MAX)); // quad
```

- Option for wide expression evaluation (C, C++)

  `-fpeval=float|double|extended`

  type used to evaluate narrower binary operations & constants

- Evaluation-type names

  `float_t` and `double_t` (per C99)

- C type-generic math functions, C++ overloading, Fortran intrinsics

Example

Compute log(ab + cd), where ab + cd ≥ 0

```
cc ...
#include <math.h>
double a, b, c, d, res;
double s;
s = a * b + c * d;
res = log(s);
```

risks:
- cancellation in ab+cd
- log instability where ab+cd inexact and near 1
- premature over/underflow

```
cc -fpeval=extended ...

#include <math.h>
double a, b, c, d, res;
double s;
s = a * b + c * d;
res = log(s);
```

- ab+cd calculated to extended
- reduces risk of cancellation

```
cc -fpeval=extended \
  -fpwidetypes ...
#include <tgmath.h>
double a, b, c, d, res;
double_t s;
s = a * b + c * d;
res = log(s);
```

- log(s) computed to extended
- reduces risk of precision problems 1000X
- eliminates premature over/underflow
- **C99-portable code**

Using Fused Floating Multiply-Add (FMA)

- *Fused* means multiply and add with just one rounding
- Compiler synthesis
- C99 fma function
  - Use the fma function ***to be certain*** of using FMA instead of multiplication followed by addition.
  - Inlined as one Itanium instruction
- Allows low-order product bits to easily be obtained
- Smooth path for implementing higher precision floating-point arithmetic

Using FMA

Example: Computing exp(xy)

Consider this program fragment:

```
extended x, y, r;
r = expw(x*y);
```

- expw is the extended precision exponential function

The relative error in the exponential function is proportional to the absolute error in its argument.

Rounded result of x*y can be in error by as much as 2^{-50}.

- Largest argument for which expw won't overflow is slightly less than 2^{14}.

The low order 14 bits of expw may be corrupted because of the rounding error in x*y.

Using FMA

Example: Computing exp(xy)

Analysis

We can write xy as the exact sum high+low where high is the computed x*y and low is the error which an FMA can determine.

Then exp(high+low) =

exp(high)*exp(low)

But exp(low) = 1 + low + … , and $|low| < 2^{-50}$

So exp(high+low) is very nearly exp(high)*(1 + low), or

exp(high) + low*exp(high)

Computation of exp(high) produces at most .5+ ulp error.

The multiply-add will produce at most .5 ulp error.

Maximum error in computing exp(xy) will be slightly over 1 ulp.

Using FMA

Example: Computing exp(xy)

Final Code

```
cc -fpwidetypes …
```

```
#include <math.h>
extended x, y, r, high, low,
  rt;
high = x * y;
low = fmaw(x, y, -high);
//xy = high+low exactly
rt = expw(high);
r = fmaw(rt, low, rt);
```

fmaw is the extended precision FMA function

Intelligible Tradeoffs between FP Behavior and Performance

- General optimization control
- Controls for special FP functionality
- Controls to trade-off FP model for speed

General Optimization Controls

+O2, +O3, profiling, binding options (-Bprotected), user assertions (+Onoptrs_to_globals), etc.

+O2

- very effective for FP performance
- optimizes math function calls like FP ops, and inlines sqrt

+O3

- inlines key math functions (e.g. log, exp)
- very effective in some loop contexts, e.g. throughput of an inlined, software pipelined exp can approach one value per 6 cycles, vs about 50 cycles if a closed routine is called

No negative effect on specified FP behavior

Controls for Special FP Functionality

Using FP control modes and exception flags

- requires one of

```
+Ofenvaccess
#pragma STDC FENV_ACCESS ON  //C99 feature
```

- else optimization might undermine expected behavior, e.g. in

```
#include <fenv.h>
{
  #pragma STDC FENV_ACCESS ON
  fesetround(FE_UPWARD);
  a = b * c;
}
```

without the pragma, b*c might be moved before the fesetround call

- compiler still optimizes, honoring constraints
- for best performance use pragma on smallest block enclosing sensitive code

Controls for Special FP Functionality

Using errno for math functions

- requires +Olibm errno compile option
- incurs substantial performance penalty
- seriously consider rewriting code to use FP exception flags

Controls to Trade-off FP Behavior for Speed

+Ofltacc=strict|default|limited|relaxed

strict	disallows value changing optimizations
default	like strict, except allows contractions (e.g. FMA)
limited	like default, except NaNs, infinities, and sign of zero may not be per spec
relaxed	allows transformations based on mathematical identities (even if numerical results are changed) – compiler might invoke slightly less accurate math functions

Controls to Trade-off FP Behavior for Speed

+FPD

- installs flush-to-zero underflow mode at startup
- dramatically speeds up some 32-bit float codes on Itanium
- effect on subsequent implementations of Itanium Processor Family architecture may not be so large
- the default IEEE gradual underflow mode makes underflow less likely to affect program robustness

+O fast (or -fast) implies "+O fltacc=relaxed +FPD"

and other performance options not specific to FP

Consider speed vs quality controls for performance hungry code that is known to be tolerant of less rigorous FP behavior or can be thoroughly tested

IEEE 754 (IEC 60559) and Related Features Status

IEEE 754 features in HW well before 1985 when standard became official

Some features now taken for granted by programmers

- single and double data types
- “correctly rounded” arithmetic

Standard doesn’t specify programming language/library bindings

Some features still not widely available and practical for serious use by programmers

- infinities, NaNs, signed zeros
- rounding modes
- exception flags

IEEE 754 and Related Features Status

Anticipated standard-related features still not widely available and practical for serious use

- predictable expression evaluation
- consistent wide evaluation
- compatible elementary functions
- compatible complex arithmetic

Deficiencies addressed in C99 …

C99 Support for IEEE 754 in HP-UX/Itanium

- NAN and INFINITY constants, usable in static and aggregate initialization
- I/O for infinities, NaNs, and sign of zero
- Infinities, NaNs, and signed zero respected
- API for manipulating rounding modes and exception flags
- Pragmas to guarantee reliable rounding modes and exception flags and to limit performance impact
- Pragmas to optionally disallow contractions (e.g. fma synthesis)
- Specification of wide evaluation methods, with auxiliary features, including type-generic math functions

C99 Support for IEEE 754 in HP-UX/Itanium

- Specification of compatible math functions (C99 Annex F)
- Specification of compatible complex arithmetic and functions (C99 Annex G)
- Specification of correctly-rounded binary-decimal conversion

  HP-UX/Itanium correctly rounds between each FP format and up to 36 decimal digits (sufficient to distinguish all quad values)

Using IEEE 754 Special Values in C99

Example: Find the maximum of partially initialized data, read in from text, where uninitialized data is represented by NaNs.

```
// Sample data: 2.3  nan  -4.5  -inf  nan  -0  2.4  -1e10 …
#include <math.h>
float max = -INFINITY;
…
for (i=0; i<N; i++) { fscanf(stream, "%f", &x[i]); }
…
for (i=0; i<N; i++) { max = fmaxf(max, x[i]); }
```

- printf and scanf support nan, inf (and infinity) for I/O
- math.h defines INFINITY and NAN macros (usable for static and aggregate initialization)
- C99 fmax returns larger numerical argument (even if other argument is NaN)

Using IEEE 754 Exceptions in C99 [4]

Example: Solve a set of equations, with speed *and* robustness

```
#include <fenv.h>
#pragma STDC FENV_ACCESS ON
//Clear the exception flags
feclearexcept(FE_ALL_EXCEPT);
//Try a fast algorithm
fastSolve (coeff, rhs, result);
if (fetestexcept (FE_ALL_EXCEPT & ~FE_INEXACT)) {
      //Oops! The simple algorithm ran into trouble!
      carefulSolve(coeff, rhs, result); //Slow but careful
}
```

- The fast algorithm may be several times faster than the careful one which is typically required only rarely.

C99 Complex Features in HP-UX/Itanium

- Complex types

  float complex double complex ...
- Imaginary types

  float imaginary double imaginary ...
- Imaginary unit
- Infinity properties
- Complex function library
- IEEE 754 compatible special cases

Infinity Properties

For z nonzero and finite

inf*z=inf inf*inf=inf

inf/z=inf inf/0=inf

z/inf=0   0/inf=0

z/0=inf |inf|=inf even for complex z, 0s, and infinities -- where a complex value with at least one infinite part is regarded as infinite (even if the other part is NaN)

- Enhances robustness

  e.g. 1/(z*z) returns 0 when z*z overflows

- Facilitates modeling Riemann sphere
- Performance cost significant in vector contexts

  +0 cx limited range allows faster multiply and divide which don't support infinity properties

Using C99 Complex [5]

Example: Efficiently determine if (z – i) / (z + 2i) lies outside the unit circle, given z = x + yi, for x and y real

```
#include <complex.h>

double x, y;

double complex z;

z = x + y*I;

w = (z – I) / (z + 2*I);

if (cabs(w) > 1) { /* outside unit circle */ }

else { /* not outside unit circle */ }
```

- Natural mathematical-style notation
- C99 avoids promotions among real, complex, and imaginary types, for built-in efficiency: x + y*I requires no FP ops
- Infinity properties assure the code works even if z = -2i, saving additional special-case code

Summary

- HP-UX 11iv1.5 for Itanium provides a substantially enhanced FP model
- Software developers can use
  - high quality, high performance library functions
  - FMA
  - wide FP types
  - IEEE 754-based features

  for robustness and performance
- They can use intelligible and convenient options and pragmas to balance FP behavior and performance needs
- They can use C99 complex for mathematical-style notation and built-in efficiency and consistency

References

- Li, R., Markstein, P., Okada, J., Thomas, J.: "The *Libm* Library and Floating-Point Arithmetic for HP-UX on Itanium™". Describes development goals and strategies, shows accuracy and performance data, and provides details for several topics in this slide set. http://devresource.hp.com/devresource/Docs/TechPapers/IA64/libm.pdf
- Markstein, P.: *IA-64 and Elementary Functions, Speed and Precision*, Prentice Hall PTR, 2000. Primary developer of the HP-UX elementary functions presents algorithms, implementation details, and related aspects of the IPF architecture.
- Intel® IA-64 Architecture, Software Developer's Manual, 2000 http://developer.intel.com/design/ia-64/manuals/index.htm
- Demmel, J., Li, X.: "Faster Numerical Algorithms via Exception Handling". A study of the approach shown in the example using IEEE 754 exceptions. http://sunsite.berkeley.edu/Dienst/UI/2.0/Describe/ncstrl.ucb/CSD-93-728
- Kahan, W., Darcy, J: "How Java's Floating-Point Hurts Everyone Everywhere", pp11-15. Shows other uses of C99 style complex. http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf
- ANSI/IEEE Std 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*
- ANSI/ISO/IEC 9899:1999, *Programming languages – C* (C99)
- JTC1/SC22/WG14: "Rationale for International Standard – Programming Languages – C" http://anubis.dkuug.dk/jtc1/sc22/wg14/