Floating-point control in the Intel C/C++ compiler and libraries
or
Why doesn't my application always give the same answer?

Martyn Corden
Developer Products Division
Software Solutions Group
Intel Corporation
February 2012

Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2012. Intel Corporation.

http://intel.com/software/products

Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Agenda

- Overview
- Floating Point (FP) Model
  - Comparisons with gcc
- Performance impact
- Runtime math libraries

Overview

- The finite precision of floating-point operations leads to an inherent uncertainty in the results of a floating-point computation
  - Results may vary within this uncertainty

- Nevertheless, may need reproducibility beyond this uncertainty
  - For reasons of Quality Assurance, e.g. when porting, optimizing, etc

- The right compiler options can deliver consistent, closely reproducible results whilst preserving good performance
  - Across IA-32, Intel® 64 and other IEEE-compliant platforms
  - Across optimization levels
  - -fp-model is the recommended high level control for the Intel Compiler

Floating Point (FP) Programming Objectives

- **Accuracy**
  - Produce results that are “close” to the correct value
    - Measured in relative error, possibly in ulp
- **Reproducibility**
  - Produce consistent results
    - From one run to the next
    - From one set of build options to another
    - From one compiler to another
    - From one platform to another
- **Performance**
  - Produce the most efficient code possible

These options usually conflict!
Judicious use of compiler options lets you control the tradeoffs.
Different compilers have different defaults.

Agenda

- Overview
- Floating Point (FP) Model
- Performance impact
- Runtime math libraries

Floating Point Semantics

- The –fp-model (/fp:) switch lets you choose the floating point semantics at a coarse granularity. It lets you specify the compiler rules for:
    - **Value safety** (main focus)
    - FP expression evaluation
    - FPU environment access
    - Precise FP exceptions
    - FP contractions (fused multiply-add)
- Also pragmas in C99 standard
    - #pragma STDC FENV_ACCESS etc
- Old switches such as –mp now deprecated
    - Less consistent and incomplete; don't use

The –fp-model switch for icc

- **-fp-model**

– fast [=1]	allows value-unsafe optimizations (default)
– fast=2	allows additional approximations
– precise	value-safe optimizations only (also source, double, extended)
– except	enable floating point exception semantics
– strict	precise + except + disable fma + don't assume default floating-point environment

- Replaces old switches –mp, -fp-port, etc (don't use!)

- **-fp-model precise -fp-model source**
  - recommended for ANSI/ IEEE standards compliance, C++ & Fortran
  - "source" is default with "precise" on Intel 64 Linux

GCC option

- -f[no-]fast-math is high level option
  - It is off by default (different from icc)
- Components control similar features:
  - Value safety (-funsafe-math-optimizations)
    - includes reassociation
  - Reproducibility of exceptions
  - assumptions about floating-point environment
  - Assumptions about exceptional values
- also sets abrupt/gradual underflow (FTZ)
- For more detail, check http://gcc.gnu.org/wiki/FloatingPointMath

Value Safety

- In SAFE mode, the compiler may not make any transformations that could affect the result, e.g. all the following are prohibited.

x / x ⇔ 1.0	x could be 0.0, ∞, or NaN
x – y ⇔ - (y – x)	If x equals y, x – y is +0.0 while – (y – x) is -0.0
x – x ⇔ 0.0	x could be ∞ or NaN
x * 0.0 ⇔ 0.0	x could be -0.0, ∞, or NaN
x + 0.0 ⇔ x	x could be -0.0
(x + y) + z ⇔ x + (y + z)	General reassociation is not value safe
(x == x) ⇔ true	x could be NaN

- UNSAFE (fast) mode is the icc default
- VERY UNSAFE mode enables riskier transformations

Value Safety

Affected Optimizations include:

- Reassociation
- Flush-to-zero
- Expression Evaluation, various mathematical simplifications
- Approximate divide and sqrt
- Math library approximations

Reassociation

- Addition & multiplication are “associative” (& distributive)
  - $a+b+c = (a+b) + c = a + (b+c)$
  - $a*b + a*c = a * (b+c)$
- These transformations are equivalent ***mathematically***
  - but ***not*** in finite precision arithmetic
- Reassociation can be disabled in its entirety
  - ⇒ for standards conformance (C left-to-right)
  - Use **-fp-model precise**
  - May carry a significant performance penalty (other optimizations also disabled)

Example (see exercises)

“tiny” is intended to keep a[i]>0

but… optimizer hoists constant expression (c+tiny) out of loop

tiny gets “rounded away” wrt c

```
icc –O1 reassoc.cpp; ./a.out
a = 0   b = inf
icc -fp-model precise reassoc.cpp;./a.out
a = 1e-20   b = 1e+20
```

```
g++ reassoc.cpp; ./a.out
a = 1e-20   b = 1e+20
g++ -O3 -ffast-math reassoc.cpp; ./a.out
a = 0   b = inf
```

```
#include <iostream>
#define N 100

int main()   {
  float a[N], b[N];
  float c = -1., tiny = 1.e-20F;

  for (int i=0; i<N; i++) a[i]=1.0;

  for (int i=0; i<N; i++)  {
    a[i] = a[i] + c + tiny;
    b[i] = 1/a[i];
  }
  std::cout << "a = " << a[0] <<
  "   b = " << b[0] << "\n";
}
```

Denormalized numbers and Flush-to-Zero (FTZ)

- Denormals extend the (lower) range of IEEE floating-point values, at the cost of:
  - Reduced precision
  - Reduced performance (can be 100 X for ops with denormals)

- If your application creates but does not depend on denormal values, setting these to zero may improve performance (“abrupt underflow”, or “flush-to-zero”,)
  - Done in SSE or AVX hardware, so fast
  - Happens by default at –O1 or higher (for icc, not gcc)
  - -no-ftz or –fp-model precise will prevent
    - Must compile main with this switch to have an effect
    - -fp-model precise –ftz to get “precise” without denormals
  - Not available for x87, denormals always generated
    - (unless trapped and set to zero in software – very slow)
- For gcc, -ffast-math sets abrupt underflow (FTZ)
  - But –O3 -ffast-math reverts to gradual underflow

Reductions

- Parallel implementations imply reassociation (partial sums)
  - Not value safe, but can give substantial performance advantage
  - -fp-model precise
    - disables vectorization of reductions
    - does not affect OpenMP* or MPI* reductions

      These remain value-unsafe (programmer's responsibility)

```
float Sum(const float A[], int n )
{
    float sum=0;
    for (int i=0; i<n; i++)
        sum = sum + A[i];
    return sum;
}
```

```
float Sum( const float A[], int n )
{
  int i, n4 = n-n%4;
  float sum=0,sum1=0,sum2=0,sum3=0;
  for (i=0; i<n4; i+=4) {
      sum  = sum  + A[i];
      sum1 = sum1 + A[i+1];
      sum2 = sum2 + A[i+2];
      sum3 = sum3 + A[i+3];
  }
  sum = sum + sum1 + sum2 + sum3;
  for (; i<n; i++) sum = sum + A[i];
   return sum;    }
```

Reproducibility of Reductions in OpenMP*

- Each thread has its own partial sum
  - Breakdown, & hence results, depend on number of threads
  - Partial sums are summed at end of loop
  - Order of partial sums is undefined (OpenMP standard)
    - First come, first served
    - Result may vary from run to run (even for same # of threads)
    - For both gcc and icc
    - Can be more accurate than serial sum
  - For icc, option to define the order of partial sums (tree)
    - Makes results reproducible from run to run
    - export KMP_FORCE_REDUCTION=tree
      - May also help accuracy
      - Possible slight performance impact, depends on context
      - Requires static scheduling, fixed number of threads
      - currently undocumented
      - See example

FP Expression Evaluation

- In the following expression, what if a, b, c, and d are mixed data types (single and double for example)

a = (b + c) + d

Four possibilities for intermediate rounding, (corresponding to C99 FLT_EVAL_METHOD)

Indeterminate		(-fp-model fast)
Use precision specified in source		(-fp-model source)
Use double precision	(C/C++ only)	(-fp-model double)
Use long double precision	(C/C++ only)	(-fp-model extended)

- Or platform-dependent default (-fp-model precise)
  - Defaults to –fp-model source on Intel64
  - Recommended for most purposes
- The expression evaluation method can significantly impact performance, accuracy, and portability

The Floating Point Unit (FPU) Environment

- FP Control Word Settings
  - Rounding mode (nearest, toward +∞, toward -∞, toward 0)
  - Exception masks, status flags
    (inexact, underflow, overflow, divide by zero, denormal, invalid)
  - Flush-to-zero (FTZ), Denormals-are-zero (DAZ)
  - x87 precision control (single, double, extended) [don't mess!]

- Affected Optimizations, e.g.
  - Constant folding
  - FP speculation
  - Partial redundancy elimination
  - Common subexpression elimination
  - Dead code elimination
  - Conditional transform, e.g.

    if (c) x = y; else x = z; ➔ x = (c) ? y : z;

FPU Environment Access

- When access disabled (default):
  - compiler assumes default FPU environment
    - Round-to-nearest
    - All exceptions masked
    - No FTZ/DAZ
  - Compiler assumes program will NOT read status flags

- If user might change the default FPU environment, inform compiler by setting FPU environment access mode!!
  - Access may only be enabled in value-safe modes, by:
    - **-fp-model strict** or
    - #pragma STDC FENV_ACCESS ON
  - Compiler treats control settings as unknown
  - Compiler preserves status flags
  - Some optimizations are disabled

Precise FP Exceptions

- When Disabled (default):
  - Code may be reordered by optimization
  - FP exceptions might not occur in the “right” places

- When enabled by
  -fp-model strict
  -fp-model except
  #pragma float_control(except, on)
  - The compiler must account for the possibility that any FP operation might throw an exception
    - Disables optimizations such as FP speculation
    - May only be enabled in value-safe modes
    - (more complicated for x87)
  - Does not unmask exceptions
    - Must do that separately, e.g.

    -fp-trap=common for C

    or functions calls such as feenableexcept()

    -fpe0 or set_halting_mode() for Fortran

Example

```
double  x.,  zero = 0.;
    feenableexcept
(FE_DIVBYZERO);
    for( int i = 0; i < 20;
i++ )
        x = zero ? (1./zero) :
zero;
        …
```

Problem: F-P exception from (1./zero) despite explicit protection

- The invariant (1./zero) gets speculatively hoisted out of loop by optimizer, but the "?" alternative does not
- exception occurs before the protection can kick in
- NOTE: does not occur for AVX due to masked vector operations

Solution: Disable optimizations that lead to the premature exception

- icc –fp-model precise –fp-model except (or icc –fp-model strict)
  disables all optimizations that could affect FP exception semantics
- icc –fp-speculation safe
  disables just speculation where this could cause an exception
- #pragma float_control around the affected code block (see doc)

Floating Point Contractions

- affects the generation of FMA instructions on Intel® MIC architecture and Intel® AVX2 (-xcore-avx2)
  - Enabled by default or -fma, disable with –no-fma
  - Disabled by –fp-model strict or C/C++ #pragma
  - -[no-]fma switch overrides –fp-model setting
  - Intel compiler does NOT support 4-operand AMD*-specific fma instruction)

- When enabled:
- The compiler may generate FMA for combined multiply/add
  - Faster, more accurate calculations
  - Results may differ in last bit from separate multiply/add
- When disabled:

  -fp-model strict, #pragma fp_contract(off) or –no-fma
  - The compiler must generate separate multiply/add with intermediate rounding

Agenda

- – Overview
- – Floating Point (FP) Model
- – Performance impact
- – Runtime math libraries

Typical Performance Impact of -fp-model source

- Measured on SPECCPU2006fp benchmark suite:
- -O2 or –O3
- Geomean reduction due to
  –fp-model precise –fp-model source
  in range 12% - 15%

- Intel Compiler XE 2011 (12.0)
- Measured on Intel Xeon® 5650 system with dual, 6-core processors at 2.67Ghz, 24GB memory, 12MB cache, SLES* 10 x64 SP2

Use -fp-model source (/fp:source) to improve floating point reproducibility whilst limiting performance impact

Agenda

- Overview
- Floating Point (FP) Model
- Performance impact
- Runtime math libraries

Math Library Functions

- Different implementations may not have the same accuracy
  - On Intel 64:
    - libsvml for vectorized loops
    - libimf (libm) elsewhere
    - Processor-dependent code within libraries, selected at runtime
    - Inlining was important for Itanium, to get software pipelining, but unimportant for Intel 64 since can vectorize with libsvml

- No official standard (yet) dictates accuracy or how results should be rounded (except for division & sqrt)

-fp-model precise helps generate consistent math calls

- eg within loops, between kernel & prolog/epilog
- Remove or reduce dependency on alignment
- May prevent vectorization unless use –fast-transcendentals
  - When may differ from non-vectorized loop

New math library features (12.x compiler)

- Select minimum precision
  - Currently for libsvml (vector); scalar libimf normally “high”

  -fimf-precision=<*high*|*medium*|*low*>
    - Default is off (compiler chooses)
    - Typically high for scalar code, medium for vector code
    - “low” typically halves the number of mantissa bits
      - Potential performance improvement
    - “high” ~0.55 ulp; “medium” < 4 ulp (typically 2)

- -fimf-arch-consistency=<*true* | *false*>
  - Will produce consistent results on all microarchitectures or processors within the same architecture
  - Run-time performance may decrease
  - Default is false (even with –fp-model precise !)

Math Libraries – known issues

- Differences could potentially arise between:
    - Different compiler releases, due to algorithm improvements
        - Use –fimf-precision
        - another workaround, use later RTL with both compilers
    - Different platforms, due to different algorithms or different code paths at runtime
        - Libraries detect run-time processor internally
        - Independent of compiler switches
        - use -fimf-arch-consistency=true
    - Expected accuracy is maintained
        - 0.55 ulp for libimf
        - < 4 ulp for libsvml (default for vectorized loops)
- Adherence to an eventual standard for math functions would improve consistency but at a cost in performance.

Intel® Math Kernel Library

- Linear algebra, FFTs, sparse solvers, statistical, …
  - Highly optimized, vectorized
  - Threaded internally using OpenMP*
  - Repeated runs may not give identical results

- Coming soon: **C**onditional **B**it**W**ise **R**eproducibility
  - Repeated runs give identical results under certain conditions:
    - Same number of threads
    - OMP_SCHEDULE=static (the default)
    - Same OS and architecture (e.g. Intel 64)
    - Same microarchitecture, or specify a minimum microarchitecture
    - Consistent data alignment
  - Call **`mkl_bwr_set(…)`**

Further Information

- Microsoft Visual C++* Floating-Point Optimization http://msdn2.microsoft.com/en-us/library/aa289157(vs.71).aspx
- The Intel® C++ and Fortran Compiler Documentation, “Floating Point Operations”
- “Consistency of Floating-Point Results using the Intel® Compiler” http://software.intel.com/en-us/articles/consistency-of-floating-point-results-using-the-intel-compiler/
- Goldberg, David: "What Every Computer Scientist Should Know About Floating-Point Arithmetic“ *Computing Surveys*, March 1991, pg. 203

Quick Overview of Primary Switches

Primary Switches	Description
/fp:*keyword* **-fp-model *keyword***	***fast***[=***1***|*2*]*, precise, source, double, extended, except, strict* Controls floating point semantics
/Qftz[-] -[no-]ftz	*Flushes denormal results to Zero*
Other switches	
/Qfp-speculation *keyword* -fp-speculation *keyword*	***fast****, safe, strict, off* *floating point speculation control*
/Qprec-div[-] -[no-]prec-div	*Improves precision of floating point divides*
/Qprec-sqrt[-] -[no-]prec-sqrt	*Improves precision of square root calculations*
/Qfma[-] -[no-]fma	*Enable[Disable] use of fma instructions*
/Qfp-trap:… -fp-trap=common	*Unmask floating point exceptions (C/C++ only)*
/fpe:0 -fpe0	*Unmask floating point exceptions (Fortran only)*
/Qfp-port -fp-port	*Round floating point results to user precision*
/Qprec -mp1	*More consistent comparisons & transcendentals*
/Op[-] -mp [-nofltconsistency]	*Deprecated; use /fp:source etc instead*

Floating-point representations

Parameter	Single	Double	Quad or Extended Precision (IEEE_X)*
Format width in bits	32	64	128
Sign width in bits	1	1	1
mantissa	23 (24 implied)	52 (53 implied)	112 (113 implied)
Exponent width in bits	8	11	15
Max binary exponent	+127	+1023	+16383
Min binary exponent	- 126	- 1022	-16382
Exponent bias	+127	+1023	+16383
Max value	~3.4 x 10^{38}	~1.8 x 10^{-308}	~1.2 x 10^{-4932}
Value (Min normalized)	~1.2 x 10^{-38}	~2.2 x 10^{-308}	~3.4 x 10^{-4932}
Value (Min denormalized)	~1.4 x 10^{-45}	~4.9 x 10^{-324}	~6.5 x 10^{-4966}

Special FP number representations

- Single precision representations

	1 Sign bit	8 Exponent bits	(1)+23 Significand bits
zero	0 or 1	0	0
denormalized	0 or 1	0	(0.)xxxxx…
normalized	0 or 1	1-254	(1.)xxxxx…
infinity	0 or 1	255	0
Signalling NaN (SNaN)	No meaning	255	(1.)0xxxx…
Quiet Nan (QNaN)	No Meaning	255	(1.)1xxxx…

Flush-To-Zero and Denormal FP Values

- A ***normalized*** FP number has leading binary bit and an exponent in the range accommodated by number of bits in the exponent.
- example:

$0.171865_{10} = 1/8 + 1/32 + 1/64$

$= 0.001011_2$

normalized $= 1.011_2 \times 2^{-3}$

- Exponent is stored in 8 bits single or 11 bits double: mantissa in 23 bits single, 52 bits double
- exponent biased by 127 (single precision)
- leading sign bit – normalized “1.” bit implied, not physically stored (1.011 stored as 011)

0 01111100 01100000000000000000000

Flush-To-Zero and Denormal FP Values

- What happens if the number is close to zero BUT exponent X in the 2^{-x} won't fit in 8 or 11 bits?
- 2^{-128} for example in single precision
- Cannot represent in a NORMALIZED fashion:
- $1/2^{127} = 0.00\ldots001_2$ (126 zeros after the binary point and a binary 1)
- $= 1.0_2 \times 2^{-128}$
- But -128 won't fit in a 127 biased 8-bit exponent value!
- Solution: DENORMAL representation
- Exponent is -126 (all zeros), NO implied leading 1.
- 0 00000000 10000000000000000000000

Flush-To-Zero and Denormal FP Values

- “Underflow” is when a very small number is created that cannot be represented. “gradual underflow” is when values are created that can be represented as denormal
- Denormals do not include as many significant digits
- Gradual loss of precision as denormal values get closer to zero

- *OK, fine, I like these denormal numbers, they carry some precision – why are denormals an issue?*
  - **UNFORTUNATELY denormals can cause 100x loss of performance**
- Solution: set any denormal to zero: FLUSH TO ZERO
  - Keeps performance up, tradeoff is some loss of precision

–prec-div and –prec-sqrt Options

- Both override the –fp-model settings
- Default is –no-prec-sqrt, and somewhere between –prec-div and –no-prec-div

[-no]-prec-div /Qprec-div[-]

- Enables[disables] various divide optimizations
  - x / y ⇔ x * (1.0 / y)
  - Approximate divide and reciprocal

[-no]-prec-sqrt /Qprec-sqrt[-]

- Enables[disables] approximate sqrt and reciprocal sqrt

-[no-]fast-transcendentals

The compiler frequently optimizes calls of math library functions (like exp, sinf) in loops

- Uses SVML (short vector math library) to vectorize loops
- Uses the XMM direct call routines,

  e.g. exp → ___libm_sse2_exp (IA-32 only)
  - Uses fast in-lined implementations

This switch “-[no]fast-transcendental can be used to overwrite default behavior

- Behavior related to settings of fp-model and other switches – see reference manual !!