Improving Numerical Reproducibility in C/C++/Fortran

Steve Lionel

Intel Corporation

steve.lionel@intel.com

Code the Future

The Three Objectives

- Accuracy
- Reproducibility
- Performance

Pick two

Code the Future

Reproducibility

Consistent results:

- From one run to the next
- From one set of build options to another
- From one compiler (or compiler version) to another
- From one processor or operating system to another

Factors that affect reproducibility

- Floating-point semantics
- Use of higher-precision intermediate results
- Differences in math libraries
- Optimization choices
- Parallelism changing operation order
- Data alignment changing vectorization
- Implementation differences between processors

Code the Future

Floating-Point Semantics

Compiled code transformations that may not be value-safe

- Reassociation – (x+y)+z to x+(y+z)
- Divide using multiply by reciprocal
- Underflow to zero
- Constant-folding
- Keeping higher precision

Code the Future

Reassociation

- Addition and multiplication are mathematically associative, but not computationally associative
- C and C++ disallow reassociation, specify left-to-right order
- Fortran allows reordering as long as parentheses are honored
- Compiler may not obey these by default

Code the Future

Optimization and Instruction Sets

- Affects order of operations and grouping of intermediate results
- Newer instruction sets enable more vectorization
- Fused multiply-add different from separate
- Automatic CPU dispatch in math libraries can change results

Code the Future

Data Alignment

- Misaligned data requires prefix and postfix loop plus main vectorized loop body
- Can change results run-to-run!
  - OS stack alignment
  - Address Space Layout Randomization

Code the Future

CAM (Community Atmospheric Model) example

A(I) + B + TOL

where

- TOL was very small and positive
- A(I) and B could be large

Compiler evaluated this as A(I)+(B+TOL)

Hoisted constant B+TOL out of the loop

TOL got rounded away….

Code the Future

Example

```
do i=1,10000
      sum = sum + sqrt(values(i))
 end do
```

No optimization: 0.2919665E+08

With optimization: 0.2919677E+08

With advanced instructions: 0.2919678E+08

Misaligned data: 0.2919677E+08

Double precision: 0.291967817899O2E+08

Code the Future

What can you do?

- Examples from Intel Fortran
- Other compilers may have similar options

Code the Future

What can you do?

- Align data to vector register width (512 bits for current Intel architectures)
  - !DIR$ ATTRIBUTES ALIGN
  - !DIR$ VECTOR ALIGNED
  - !DIR$ ASSUME_ALIGNED
  - -align
- Use parentheses liberally
  - Intel Fortran – add –assume protect_parens

Code the Future

-fp-model

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
- FP expression evaluation
- FPU environment access
- Precise FP exceptions
- FP contractions

Code the Future

-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

Code the Future

FP Expression Evaluation

- 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)
- The expression evaluation method can significantly impact performance, accuracy, and portability!

Code the Future

Value Safety

- In SAFE (precise) mode, the compiler may not make any transformations that could affect the result, e.g. the following is prohibited:
  (x + y) + z ⇨ x + (y + z)
  general reassociation is not value safe
- UNSAFE (fast) mode is the default
  - The variations implied by "unsafe" are usually very tiny
- VERY UNSAFE (fast=2) mode enables riskier transformations

Code the Future

Reductions

- Parallel implementations imply reassociation (partial sums)
  - Not value safe
- -fp-model precise
  - disables vectorization of reductions
  - does not affect OpenMP* or MPI* reductions
  - These remain value-unsafe (programmer's responsibility)

What can you do?

- -fp-model *keyword* (fast, precise, except, strict, source)
- -fimf-arch-precision=(high, medium, low) – controls accuracy of math library functions
- -fimf-arch-consistency=true – Math library gives same results across processors
- KMP_DETERMINISTIC_REDUCTION=1 for OpenMP reductions

Code the Future

Summary

- Reproducibility is a tradeoff against accuracy and performance
- Decide which kinds of reproducibility are important to you
- There are things you can do to help reduce differences

Code the Future

References

Consistency of Floating-Point Results using the Intel® Compiler (Martyn Corden and David Kreitzer) http://intel.ly/1eiQxua

Differences in Floating-Point Arithmetic Between Intel® Xeon® Processors and the Intel® Xeon Phi™ Coprocessor (Corden) http://intel.ly/1b8Qrq6

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