FP Accuracy & Reproducibility
Intel® C++/Fortran Compiler, Intel® Math Kernel Library and Intel® Threading Building Blocks

Presenter: Georg Zitzlsberger
Date: 17-09-2014
Agenda

• **FP Accuracy & Reproducibility Problems**
• **Compiler FP Controls**
• **Intel® MKL CNR**
• **Intel® TBB Deterministic Reduction**
• **Summary**
The Problem

Numerical (FP) results change on run-to-run:

```
C:\Users\me>test.exe
4.012345678901111

C:\Users\me>test.exe
4.012345678902222
```

Numerical results change between different systems:

**Intel® Xeon® Processor E5540**

```
C:\Users\me>test.exe
4.012345678901111

C:\Users\me>test.exe
4.012345678901111
```

**Intel® Xeon® Processor E3-1275**

```
C:\Users\me>test.exe
4.012345678902222

C:\Users\me>test.exe
4.012345678902222
```
Why Reproducible FP Results?

**Technical/legacy**
Software correctness is determined by comparison to previous (baseline) results.

**Debugging/porting**
When developing and debugging, a higher degree of run-to-run stability is required to find potential problems.

**Legal**
Accreditation or approval of software might require exact reproduction of previously defined results.

**Customer perception**
Developers may understand the technical issues with reproducibility but still require reproducible results since end users or customers will be disconcerted by the inconsistencies.
Why Results Vary I

Basic problem:
- FP numbers have finite resolution and
- Rounding is done for each (intermediate) result

Caused by algorithm:
Conditional numerical computation for different systems and/or input data can have unexpected results

Non-deterministic task/thread scheduler:
Asynchronous task/thread scheduling has best performance but reruns use different threads

Alignment (heap & stack):
If alignment is not guaranteed and changes between reruns the data sets could be computed differently (e.g. vector loop prologue & epilogue of unaligned data)

⇒ User controls those (direct or indirect)
Why Results Vary II

Order of FP operations has impact on rounded result, e.g.

\[(a+b)+c \neq a+(b+c)\]

\[2^{-63} + 1 + -1 = 2^{-63}\] (mathematical result)

\[(2^{-63} + 1) + -1 \approx 0\] (correct IEEE result)

\[2^{-63} + (1 + -1) \approx 2^{-63}\] (correct IEEE result)

Constant folding: \(X + 0 \Rightarrow X\) or \(X * 1 \Rightarrow X\)

Multiply by reciprocal: \(A/B \Rightarrow A * (1/B)\)

Approximated transcendental functions (e.g. \(\text{sqrt}(\ldots), \text{sin}(\ldots), \ldots\))

Flush-to-zero (for SIMD instructions)

Contractions (e.g. FMA)

Different code paths (e.g. SIMD & non-SIMD or Intel AVX vs. SSE)

\(\Rightarrow\) Subject of Optimizations by Compiler & Libraries

9/17/2014
Agenda

- FP Accuracy & Reproducibility Problems
- Compiler FP Controls
- Intel® MKL CNR
- Intel® TBB Deterministic Reduction
- Summary
Compiler Optimizations

Why compiler optimizations:

- Provide best performance
- Make use of processor features like SIMD (vectorization)
- In most cases performance is more important than FP precision and reproducibility
- Use faster FP operations (not legacy x87 coprocessor)

FP model of compiler limits optimizations and provides control about FP precision and reproducibility:

Default is “fast”

Controlled via:
Linux®, OS X*: -fp-model
Windows*: /fp:
FP Model I

FP model does more:

- Value safety
- Floating-point expression evaluation
- Precise floating-point exceptions
- Floating-point contractions
- Floating-point unit (FPU) environment access
FP Model II

FP model settings:

- **precise**: allows value-safe optimizations only
- **source/double/extended**: intermediate precision for FP expression eval.
- **except**: enables strict floating point exception semantics
- **strict**: enables access to the FPU environment disables floating point contractions such as fused multiply-add (fma) instructions implies “precise” and “except”

- **fast=[=1] (default)**:
  Allows value-unsafe optimizations compiler chooses precision for expression evaluation
  Floating-point exception semantics not enforced
  Access to the FPU environment not allowed
  Floating-point contractions are allowed

- **fast=2**: some additional approximations allowed
FP Model III

Using `-fast`, translates to:

```
-ipo -O3 -no-prec-div -static -xHost -fp-model fast=2
```

- **On Intel® Xeon® & Intel® Core™:**
  Can speed up COMPLEX operations and is similar to GNU* GCC's (default) FP optimizations

- **On Intel® Xeon Phi™:**
  Implies `-fimf-domain-exclusion=15` which results in faster executions of math library routines; excludes extremes, nans, infinities and denormals
## FP Model - Comparison

<table>
<thead>
<tr>
<th>Key</th>
<th>Value Safety</th>
<th>Expression Evaluation</th>
<th>FPU Environ. Access</th>
<th>Precise FP Exceptions</th>
<th>FP contract</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>precise source double extended</strong></td>
<td>Safe</td>
<td>Varies Source Double Extended</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>strict</strong></td>
<td>Safe</td>
<td>Varies</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>fast=1 (default)</strong></td>
<td>Unsafe</td>
<td>Unknown</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>fast=2</strong></td>
<td>Very Unsafe</td>
<td>Unknown</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>except</strong></td>
<td>/*/**</td>
<td>*</td>
<td>*</td>
<td>Yes</td>
<td>*</td>
</tr>
<tr>
<td><strong>except-</strong></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>No</td>
<td>*</td>
</tr>
</tbody>
</table>

* These modes are unaffected. `-fp-model except[-]` only affects the precise FP exceptions mode.

** It is illegal to specify `-fp-model except` in an unsafe value safety mode.
FP Model - Example

Using \texttt{-fp-model [precise|strict]}:

- Disables reassociation
- Enforces standard conformance (left-to-right)
- May carry a significant performance penalty

Disabling of reassociation also impacts vectorization (e.g. partial sums)!

```cpp
#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";
}
```
Other FP Options I

- **Linux**, OS X*: `[no-] ftz`, Windows*: `/Qftz [-]`
  Flush denormal results to zero

- **Linux**, OS X*: `[no-] prec-div`, Windows*: `/Qprec-div [-]`
  Improves precision of floating point divides

- **Linux**, OS X*: `[no-] prec-sqrt`, Windows*: `/Qprec-sqrt [-]`
  Improves precision of square root calculations

- **Linux**, OS X*: `-fimf-precision=name`, Windows*: `/Qimf-precision:name`
  `high, medium, low`: Controls accuracy of math library functions

- **Linux**, OS X*: `-fimf-arch-consistency=true`, Windows*: `/Qimf-arch-consistency:true`
  Math library functions produce consistent results on different processor types of the same architecture
Other FP Options II

- Linux*, OS X*: `-fpe0`, Windows*: `/fpe:0`
  Unmask floating point exceptions (Fortran only) and disable generation of denormalized numbers

- Linux*, OS X*: `-fp-trap=common`, Windows*: `/Qfp-trap:common`
  Unmask common floating point exceptions (C/C++ only)

- Linux*, OS X*: `-[no-]fast-transcendentals`, Windows*: `/Qfast-transcendentals [-]`
  Enable/disable fast math functions

- ...
Pragmas (C/C++ only)

- \#pragma fenv_access
  Informs about possibly changed FP environment; requires strict FP model

- Block-wise control:
  \#pragma float_control(...,[on|off])
  Turn on/off FP model settings

Examples:

- \#pragma float_control(except,[on|off])
  Compiler has to expect/handle FP exceptions
  Alternative: use strict or except FP model

- \#pragma float_control(fma,[on|off])
  FP contractions are allowed/disallowed
  Alternative: use strict FP model; -no-fma or /Qfma-
**FP Expression Evaluation**

**FLT_EVAL_METHOD** (C99) to control rounding of intermediate results, e.g.:
\[ a = (b + c) + d \] (float_t & double_t)

<table>
<thead>
<tr>
<th>Evaluation Method</th>
<th>(/fp: (-fp-model))</th>
<th>Language</th>
<th>FLT_EVAL_METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indeterminate</td>
<td>fast</td>
<td>C/C++/Fortran</td>
<td>-1</td>
</tr>
<tr>
<td>Use source precision</td>
<td>source</td>
<td>C/C++/Fortran</td>
<td>0 (default)</td>
</tr>
<tr>
<td>Use double precision</td>
<td>double</td>
<td>C/C++</td>
<td>1</td>
</tr>
<tr>
<td>Use long double precision</td>
<td>extended</td>
<td>C/C++</td>
<td>2</td>
</tr>
</tbody>
</table>

9/17/2014
OpenMP* Deterministic Reduction

**KMP_DETERMINISTIC_REDUCTION:** Enables (1) or disables (0) the use of a specific ordering of the reduction operations for implementing the reduction clause for an OpenMP* parallel region. This has the effect that, for a given number of threads, in a given parallel region, for a given data set and reduction operation, a floating point reduction done for an OpenMP reduction clause will have a consistent floating point result from run to run, since round-off errors will be identical.

Use with `-fp-model [precise|strict]`!
Recommendation

- The **default FP model** is fast but has less precision/reproducibility (vectorization)

- The **strict FP model** has best precision/reproducibility but is slow (no vectorization; x87 legacy)

- For best trade-off between precision, reproducibility & performance use:  
  Linux*, OS X*: `-fp-model precise -fp-model source`  
  Windows*: `/fp:precise /fp:source`  
  Approx. 12-15% slower performance for SPECCPU2006fp

- Don’t mix math libraries from different compiler versions!

- Using different processor types (of same architecture), specify:  
  Linux*, OS X*: `-fimf-arch-consistency=true`  
  Windows*: `/Qimf-arch-consistency:true`

**More information:**

9/17/2014
Agenda

- FP Accuracy & Reproducibility Problems
- Compiler FP Controls
- Intel® MKL CNR
- Intel® TBB Deterministic Reduction
- Summary
Historical Intel® MKL reproducibility

Through MKL 10.3 (Nov. 2011), the recommendation was to:

- Align your input/output arrays using the Intel® MKL memory manager

- Call sequential Intel® MKL

- This meant the user needed to handle threading
Conditional Numerical Reproducibility (CNR):

- **Memory alignment**
  - Align memory — try Intel MKL memory allocation functions
  - 64-byte alignment for processors in the next few years

- **Number of threads**
  - Set the number of threads to a constant number
  - Use sequential libraries

- **Deterministic task scheduling**
  - Ensures that FP operations occur in order to ensure reproducible results

- **Code path control**
  - Maintains consistent code paths across processors
  - Will often mean lower performance on the latest processors

Achieve best performance possible for cases that require reproducibility
New in Intel® MKL 11.1

Conditional Numerical Reproducibility (CNR):

- Data alignment no longer requirement for numerical reproducibility.
- But aligning input data is still a good idea for getting better performance.

<table>
<thead>
<tr>
<th>Pre-requisite: Fixed number of threads</th>
<th>Deterministic task scheduling</th>
<th>Code path control</th>
</tr>
</thead>
</table>
| • Set the number of threads to a constant number (MKL_NUM_THREADS) | • Ensures that FP operations occur in order to ensure reproducible results | • Maintains consistent code paths across processors  
• Will often mean lower performance on the latest processors |

9/17/2014
Controls for CNR Features

<table>
<thead>
<tr>
<th>Maximum Compatibility</th>
<th>Function Call</th>
<th>Environment Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>For consistent results ...</td>
<td>mkl_cbwr_set( ... )</td>
<td>MKL_CBWR_COMPATIBLE</td>
</tr>
<tr>
<td>on Intel® or Intel®-compatible CPUs supporting SSE2 instructions or later</td>
<td>COMPATIBLE</td>
<td></td>
</tr>
<tr>
<td>on Intel® processors supporting SSE2 instructions or later</td>
<td>MKL_CBWR_SSE2</td>
<td>SSE2</td>
</tr>
<tr>
<td>on Intel processors supporting SSE4.2 instructions or later</td>
<td>MKL_CBWR_SSE4_2</td>
<td>SSE4_2</td>
</tr>
<tr>
<td>on Intel processors supporting Intel® AVX or later</td>
<td>MKL_CBWR_AVX</td>
<td>AVX</td>
</tr>
<tr>
<td>from run to run (but not processor-to-processor)</td>
<td>MKL_CBWR_AUTO</td>
<td>AUTO</td>
</tr>
</tbody>
</table>

Supports up to Intel® AVX2!
### Impact of CNR on Performance

<table>
<thead>
<tr>
<th>CNR Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNR Off</td>
<td>Maximum performance with CNR off</td>
</tr>
<tr>
<td>AUTO</td>
<td>AUTO - Deterministic task scheduling</td>
</tr>
<tr>
<td>AVX</td>
<td>AVX - Best performing code path on for Intel AVX</td>
</tr>
<tr>
<td>SSE4_2</td>
<td>SSE4_2 - Code path supported on both processors</td>
</tr>
<tr>
<td>COMPATIBLE</td>
<td>COMPATIBLE - Getting reproducible results on IA and IA-compatible processors</td>
</tr>
</tbody>
</table>

#### Gflops (Peak performance)

- Intel® Xeon® E5-2690 (supporting Intel AVX)
- Intel® Xeon® X5680 (supporting SSE4.2)

Configuration Info:
- Versions: Intel® Math Kernel Library (Intel® MKL) 11.0
- Hardware: Intel® Xeon® Processor E5-2690, 2 Eight-Core CPUs (20MB LLC, 2.9GHz), 32GB of RA
- Operating System: RHEL 6 64-bit, Linux® 3.10.0-226.2.1.el6.x86_64
- Benchmark Source: Intel Corporation
- Test environment: 64-bit executable, Matrix 40x40, OMP_NUM_THREADS=12

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, refer to [www.intel.com/performance/resources/benchmark_limitations.htm](http://www.intel.com/performance/resources/benchmark_limitations.htm).

* Other brands and names are the property of their respective owners.

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

Example: LINPACK benchmark

9/17/2014
Agenda

- FP Accuracy & Reproducibility Problems
- Compiler FP Controls
- Intel® MKL CNR
- Intel® TBB Deterministic Reduction
- Summary
For non-associative operations, parallel_reduce does not guarantee deterministic results

Re-association of operations done differently

Depends on the number of threads, the partitioner used, and on which ranges are stolen.

Solution: **parallel_deterministic_reduce(...)**

Uses deterministic reduction tree.

Generates deterministic result even for floating-point (but different from serial execution)

Partitioners are disallowed

Specification of grainsize is highly recommended.
Example

Replaces `parallel_reduce(...)`: 

```cpp
sum = parallel_deterministic_reduce(
    blocked_range<int>(0,n,10000),
    0.f,
    [&](blocked_range<int> r, T s) -> float
    {
        for( int i=r.begin(); i!=r.end(); ++i )
            s += a[i];
        return s;
    },
    std::plus<T>()
);
```
Agenda

• FP Accuracy & Reproducibility Problems
• Compiler FP Controls
• Intel® MKL CNR
• Intel® TBB Deterministic Reduction
• Summary
Summary

• Both Intel® C/C++ and Fortran Compilers provide options for finding best trade-off between FP precision & reproducibility and performance

• FP control even possible for selected code blocks (not entire object or application)

• Intel® MKL 11.0 introduced CNR to control code paths executed to guarantee same results even if different processors are used

• Use Intel® MKL task scheduling for determinism

• Intel® TBB 4.1 offers parallel deterministic reduction
Legal Disclaimer & Optimization Notice

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.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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