Run-to-run Numerical Reproducibility with the Intel® Math Kernel Library and Intel® Composer XE 2013

Todd Rosenquist
Agenda

• Why do floating point results vary?
• Reproducibility in the Intel compilers
• New reproducibility features in Intel MKL
Ever seen something like this?

C:\Users\me>test.exe
4.012345678901111

C:\Users\me>test.exe
4.012345678902222

≠

C:\Users\me>test.exe
4.012345678902222

C:\Users\me>test.exe
4.012345678901111

C:\Users\me>test.exe
4.012345678902222
...or this on different processors?

Intel® Xeon® Processor E5540  Intel® Xeon® Processor E3-1275

C:\Users\me>test.exe 4.012345678901111
C:\Users\me>test.exe 4.012345678901111
C:\Users\me>test.exe 4.012345678901111
C:\Users\me>test.exe 4.012345678901111
C:\Users\me>test.exe 4.012345678902222
C:\Users\me>test.exe 4.012345678902222
C:\Users\me>test.exe 4.012345678902222
C:\Users\me>test.exe 4.012345678902222
Why do results vary?

Root cause for variations in results in Intel MKL

- floating-point numbers and rounding
- double precision example where \((a+b)+c \neq a+(b+c)\)

\[
2^{-63} + 1 + -1 = 2^{-63} \quad \text{(mathematical result)} \\
(2^{-63} + 1) + -1 \approx 0 \quad \text{(correct IEEE result)} \\
2^{-63} + (1 + -1) \approx 2^{-63} \quad \text{(correct IEEE result)}
\]
**Why might the order of operations change in a computer program**

<table>
<thead>
<tr>
<th>Optimizations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction sets</td>
<td>memory alignment affects grouping of data in registers</td>
</tr>
<tr>
<td>multiple cores / multiple processors</td>
<td>most functions are threaded to use as many cores as will give good scalability</td>
</tr>
<tr>
<td>Non-deterministic task scheduling</td>
<td>some algorithms use asynchronous task scheduling for optimal performance</td>
</tr>
<tr>
<td>code path</td>
<td>optimized to use all the processor features available on the system where the program is run</td>
</tr>
</tbody>
</table>

Many optimizations require a change in order of operations.
Why are reproducible results important for Intel MKL users?

**Technical / legacy**
Software correctness is determined by comparison to previous ‘gold’ 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.

Source: Email correspondence with Kai Diethelm of GNS. see his whitepaper:
What are the ingredients for reproducibility

Source code

Tools

• Compilers
• Libraries
Floating Point Semantics

The -fp-model (/fp:) compiler switch lets you choose the floating point semantics at a coarse granularity. It lets you specify the compiler rules for:

- **Value safety** (our main focus)
- FP expression evaluation
- FPU environment access
- Precise FP exceptions
- FP contractions (fused multiply-add)
## The -fp-model & /fp: switches

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast[=1] (default)</td>
<td>allows value-unsafe optimizations</td>
</tr>
<tr>
<td>fast=2</td>
<td>allows additional optimizations</td>
</tr>
<tr>
<td><strong>precise</strong></td>
<td>value-safe optimizations only</td>
</tr>
<tr>
<td>except</td>
<td>enables floating point exception semantics</td>
</tr>
<tr>
<td>strict</td>
<td>precise + except + disable fma + don’t assume default floating-point environment</td>
</tr>
</tbody>
</table>

- **Recommendation for reproducibility: -fp-model precise**
  - for reproducible result and for ANSI/IEEE standards compliance, C++ & Fortran
Reassociation

- **fp-model precise**
  - disables reassociation
  - enforces C std conformance (left-to-right)
  - may carry a significant performance penalty

```c
#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";
}
```

*Parentheses are respected only in value-safe mode!*
Reductions

Parallel implementations imply reassociation (partial sums)

- Not value safe, but can give substantial performance advantage
- `-fp-model precise`
  - disables vectorization of reductions, makes value safe
  - does not affect OpenMP* or MPI* or TBB reductions

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

```c
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;
}
```
Run-to-Run Variations (single-threaded)

Data alignment may vary from run to run, due to changes in the external environment

• E.g. malloc of a string to contain date, time, user name or directory:
  size of allocation affects alignment of subsequent malloc’s

• Compiler may “peel” scalar iterations off the start of the loop until subsequent memory accesses are aligned, so that the main loop kernel can be vectorized efficiently

• For reduction loops, this changes the composition of the partial sums, hence changes rounding and the final result

• Occurs for both gcc and icc, when compiling for Intel® AVX

To avoid, align data:

  _mm_malloc(size, 32)  (icc only)
  mkl_malloc(size, 32)  (Intel MKL)

• or compile with –fp-model precise (icc) or without –ffast-math (larger performance impact)
Reproducibility of Reductions in OpenMP* & TBB

Each thread has its own partial sum
- Partial sums are summed at end of loop
- Breakdown, & hence results, depend on number of threads
- 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
- For OpenMP* threading in icc & ifort, option to define the order of partial sums
  - Makes results reproducible from run to run
  - export KMP_DETERMINISTIC_REDUCTION=yes     (XE 2013)
    - May also help accuracy
    - Possible slight performance impact, depends on context
    - Requires static scheduling, fixed number of threads
    - Default for large numbers of threads
- For Threading Building Blocks (TBB):
  - Use the template function: parallel_deterministic_reduce
**Typical Performance Impact**

- SPECCPU2006fp benchmark suite compiled with -O2 or -O3
- Geomean performance reduction due to -fp-model precise and -fp-model source: 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

- Performance impact can vary between applications

---

**Use -fp-model precise to improve floating point reproducibility while limiting performance impact**
How did Intel MKL handle reproducibility historically?

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 themselves
Balancing Reproducibility and Performance: Conditional Numerical Reproducibility (CNR)

<table>
<thead>
<tr>
<th>New!</th>
<th>Memory alignment</th>
<th>Number of threads</th>
<th>Deterministic task scheduling</th>
<th>Code path control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Align memory — try Intel MKL memory allocation functions</td>
<td>• Set the number of threads to a constant number</td>
<td>• Ensures that FP operations occur in order to ensure reproducible results</td>
<td>• Maintains consistent code paths across processors</td>
</tr>
<tr>
<td></td>
<td>• 64-byte alignment for processors in the next few years</td>
<td>• Use sequential libraries</td>
<td></td>
<td>• Will often mean lower performance on the latest processors</td>
</tr>
</tbody>
</table>

**Goal:** Achieve best performance possible for cases that require reproducibility
## Controls for CNR features

<table>
<thead>
<tr>
<th>For consistent results ...</th>
<th>Function Call</th>
<th>Environment Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>on Intel® or Intel®-compatible CPUs supporting SSE2 instructions or later</td>
<td>mkl_cbwr_set( ... )</td>
<td>MKL_CBWR_COMPATIBLE</td>
</tr>
<tr>
<td>on Intel® processors supporting SSE2 instructions or later</td>
<td></td>
<td>COMPATIBLE</td>
</tr>
<tr>
<td>on Intel processors supporting SSE4.2 instructions or later</td>
<td>mkl_cbwr_SSE4_2</td>
<td>MKL_CBWR_SSE4_2</td>
</tr>
<tr>
<td>on Intel processors supporting Intel® AVX or later</td>
<td>mkl_cbwr_AVX</td>
<td>MKL_CBWR_AVX</td>
</tr>
<tr>
<td>from run to run (but not processor-to-processor)</td>
<td>mkl_cbwr_AUTO</td>
<td>MKL_CBWR_AUTO</td>
</tr>
</tbody>
</table>

*Other brands and names are the property of their respective owners.
CNR Impact on Performance of Intel® Optimized LINPACK Benchmark

- **CNR Off**: Maximum performance with CNR off
- **AUTO**: Deterministic task scheduling
- **AVX**: Best performing code path on for Intel AVX
- **SSE4_2**: Code path supported on both processors
- **COMPATIBLE**: Getting reproducible results on IA and IA-compatible processors

**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 and Intel® Xeon® Processor X5680, 2 Six-Core CPUs (12MB LLC, 3.33GHz), 48GB of RAM; Operating System: RHEL 6 GA x86_64; Benchmark Source: Intel Corporation.

Test environment: 64-bit executable, Matrix 40x x 40k, 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.

* 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
What’s next?

<table>
<thead>
<tr>
<th>Rate the importance of each of the following Conditional Numerical Reproducibility cases: +</th>
<th>Most Important</th>
<th>Very Important</th>
<th>Important</th>
<th>Somewhat Important</th>
<th>Least Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reproducibility from run to run* (*introduced in Intel MKL 11.0, Sept 2012)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reproducibility from processor to processor *</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reproducibility without memory alignment requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reproducibility on variable (versus fixed) numbers of threads</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reproducibility from OS to OS (Windows*, Linux*, Mac OS* X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reproducibility across architectures (32-bit to 64-bit OS’s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reproducibility Intel MKL version-to-version</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reproducibility from run to run on Intel® Xeon Phi™ coprocessors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reproducibility from processor to Intel® Xeon Phi™ coprocessor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

https://softwareproductsurvey.intel.com/survey/150072/1afd/
Further resources on reproducibility

Reference manuals, User Guides, Getting Started Guides...
- Intel® MKL Documentation
- Intel® Fortran Composer XE 2013 Documentation
- Intel® C++ Composer XE 2013 Documentation

Knowledgebase:
- CNR in Intel MKL 11.0, Consistency of Floating-Point Results

Support
- Intel MKL user forum
- Intel compiler forums [IVF, Fortran, and C++]
- Intel Premier support

Feedback
- Survey: https://softwareproductsurvey.intel.com/survey/150072/1afd/
Summary

When writing programs for reproducible results:
- Write your source code with reproducibility in mind
- Use “–fp-model precise” with Intel compilers
- Use the new Conditional Numerical Reproducibility (CNR) features in Intel MKL

Evaluate CNR in the following:
- Intel® Math Kernel Library 11.0
- Intel® Composer XE 2013
- Intel® Parallel Studio XE 2013
- Intel® Cluster Studio XE 2013

Provide feedback:
https://softwareproductsurvey.intel.com/survey/150072/1afd/
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 © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk
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