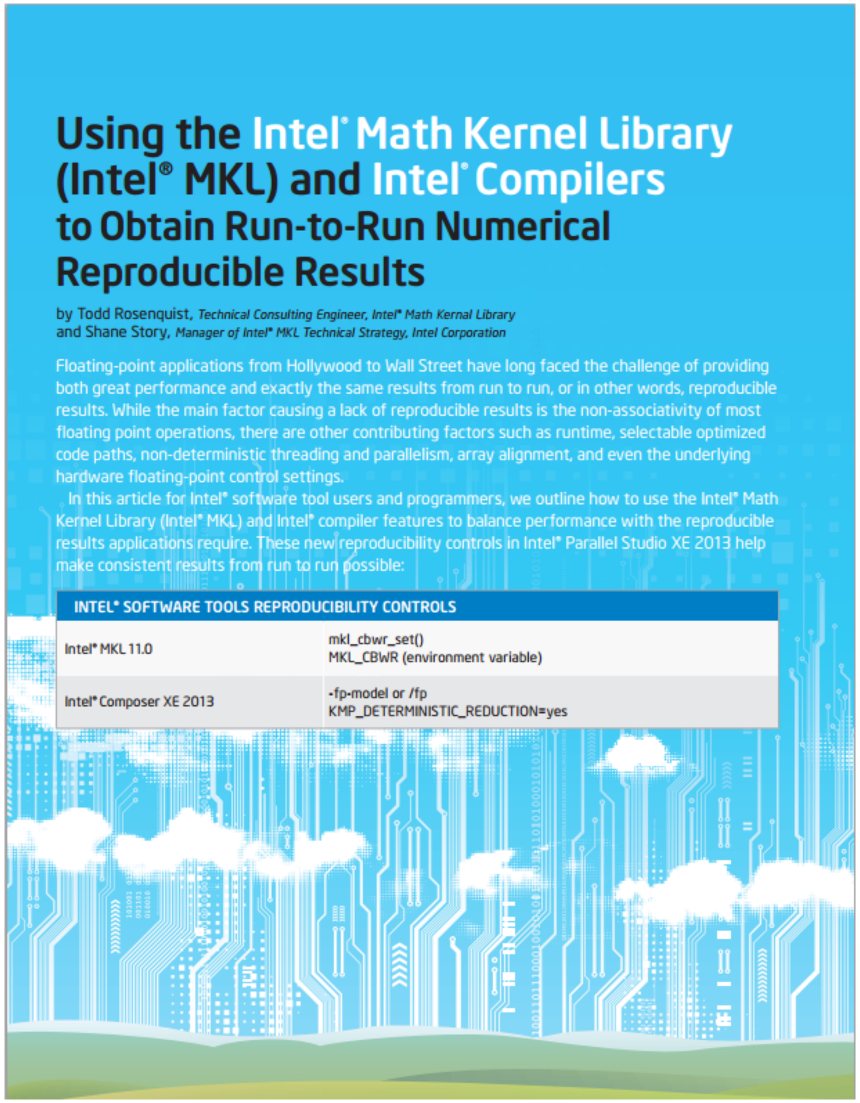
Run-to-run Numerical Reproducibility with the Intel® Math Kernel Library and Intel® Composer XE 2013

Todd Rosenquist

Agenda

- Why do floating point results vary?
- Reproducibility in the Intel compilers
- New reproducibility features in Intel MKL



Using the Intel® Math Kernel Library (Intel® MKL) and Intel® Compilers to Obtain Run-to-Run Numerical Reproducible Results

by Todd Rosenquist, *Technical Consulting Engineer, Intel® Math Kernal Library*
and Shane Story, *Manager of Intel® MKL Technical Strategy, Intel Corporation*

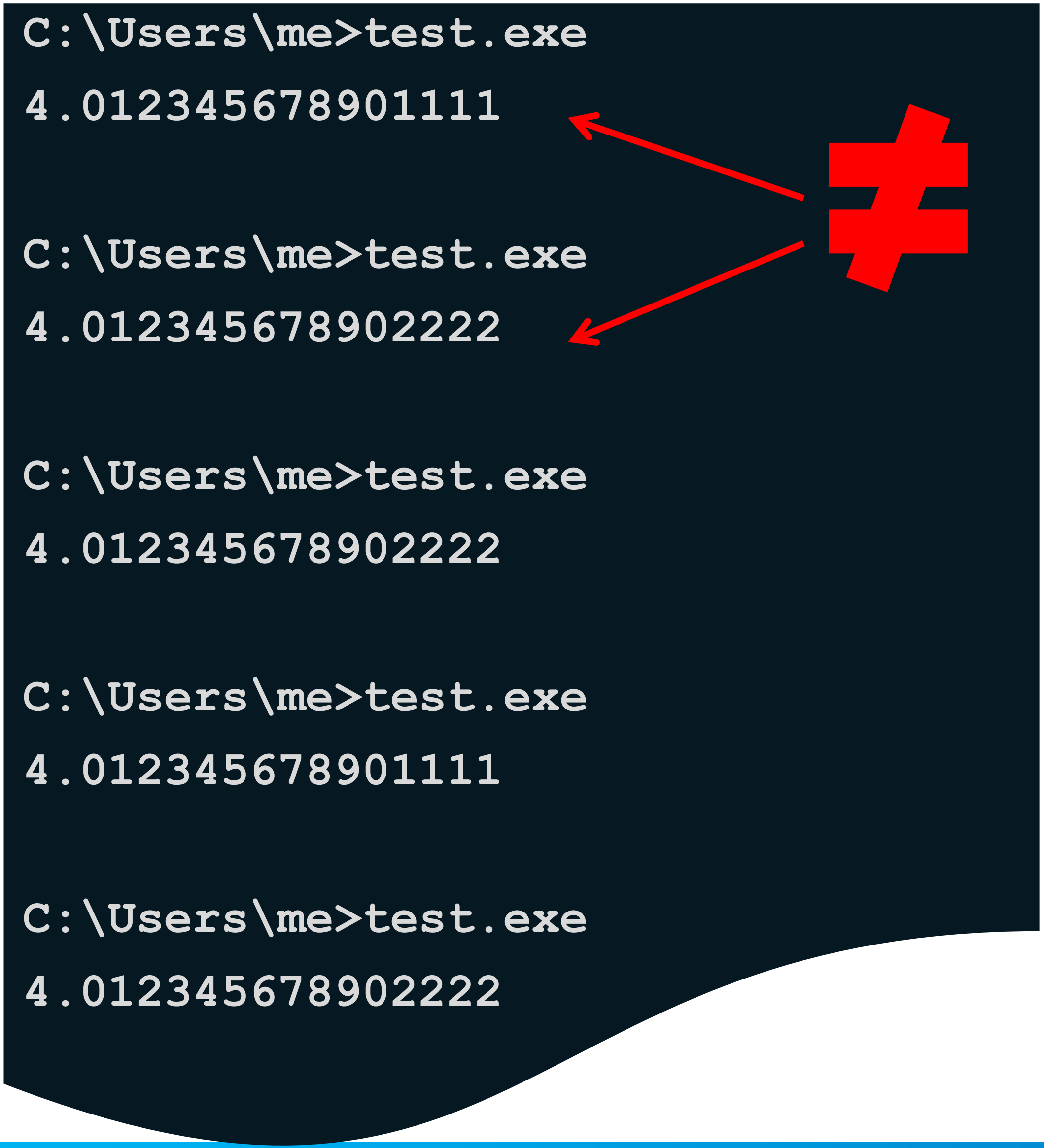
Floating-point applications from Hollywood to Wall Street have long faced the challenge of providing both great performance and exactly the same results from run to run, or in other words, reproducible results. While the main factor causing a lack of reproducible results is the non-associativity of most floating point operations, there are other contributing factors such as runtime, selectable optimized code paths, non-deterministic threading and parallelism, array alignment, and even the underlying hardware floating-point control settings.

In this article for Intel® software tool users and programmers, we outline how to use the Intel® Math Kernel Library (Intel® MKL) and Intel® compiler features to balance performance with the reproducible results applications require. These new reproducibility controls in Intel® Parallel Studio XE 2013 help make consistent results from run to run possible:

INTEL® SOFTWARE TOOLS REPRODUCIBILITY CONTROLS	
Intel® MKL 11.0	mkl_cbwr_set() MKL_CBWR (environment variable)
Intel® Composer XE 2013	-fp-model or /fp KMP_DETERMINISTIC_REDUCTION=yes

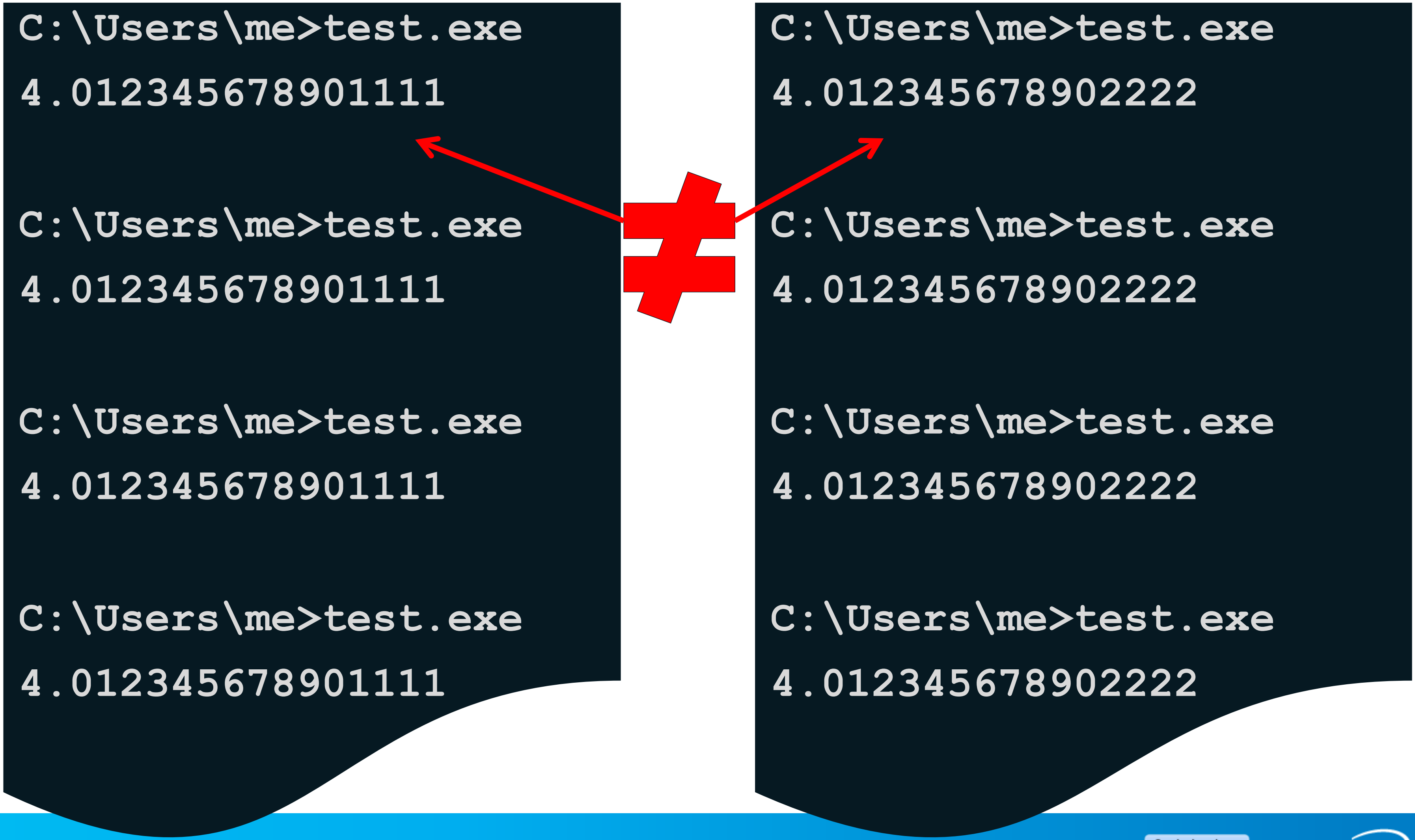
Optimization Notice

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

```
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
```

$\neq$

Intel® Xeon® Processor E3-1275

```
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
```

Optimization Notice

Why do results vary?

Root cause for variations in results in Intel MKL

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

$2^{-63} + 1 + -1 = 2^{-63}$	(mathematical result)
$(2^{-63} + 1) + -1 \approx 0$	(correct IEEE result)
$2^{-63} + (1 + -1) \approx 2^{-63}$	(correct IEEE result)

Why might the order of operations change in a computer program

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

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:
http://www.computer.org/cms/Computer.org/ComputingNow/homepage/2012/0312/W_CS_TheLimitsofReproducibilityinNumericalSimulation.pdf

Optimization Notice

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

Value	Description
fast[=1] *(default)*	allows value-unsafe optimizations
fast=2	allows additional optimizations
precise	value-safe optimizations only
except	enables floating point exception semantics
strict	precise + except + disable fma + don't assume default floating-point environment

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

Reassociation

```
#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";
}
```

-fp-model precise

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

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

```
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;    }
```

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)

Optimization Notice

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

Optimization Notice

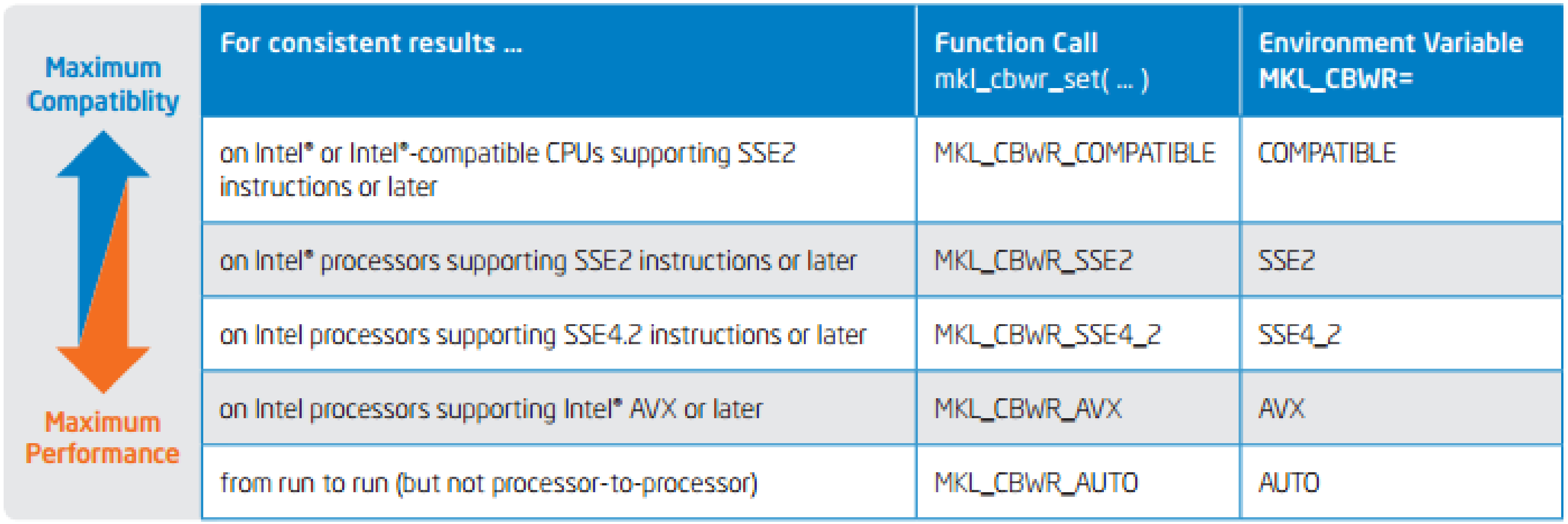
Balancing Reproducibility and Performance: Conditional Numerical Reproducibility (CNR)

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

New!

Goal: Achieve best performance possible for cases that require reproducibility

Controls for CNR features



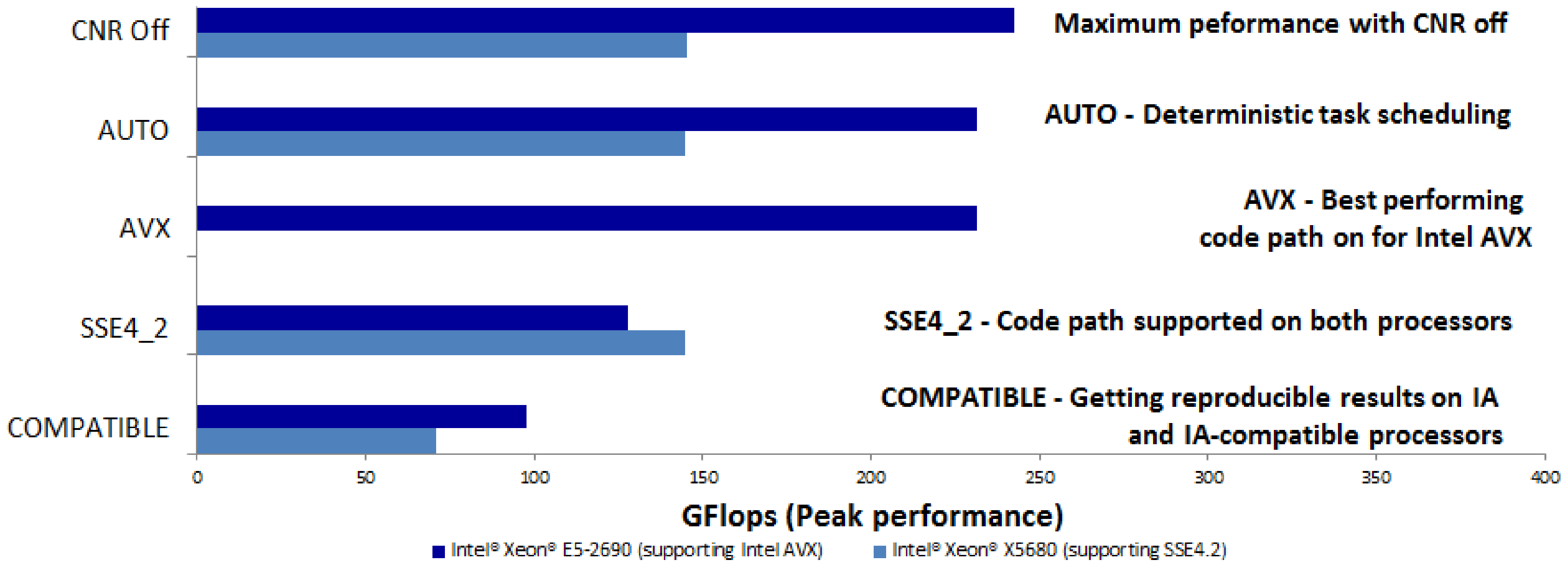
Maximum Compatiblity

Maximum Performance

For consistent results …	Function Call mkl_cbwr_set(…)	Environment Variable MKL_CBWR=
on Intel® or Intel®-compatible CPUs supporting SSE2 instructions or later	MKL_CBWR_COMPATIBLE	COMPATIBLE
on Intel® processors supporting SSE2 instructions or later	MKL_CBWR_SSE2	SSE2
on Intel processors supporting SSE4.2 instructions or later	MKL_CBWR_SSE4_2	SSE4_2
on Intel processors supporting Intel® AVX or later	MKL_CBWR_AVX	AVX
from run to run (but not processor-to-processor)	MKL_CBWR_AUTO	AUTO

Optimization Notice

CNR Impact on Performance of Intel® Optimized LINPACK Benchmark

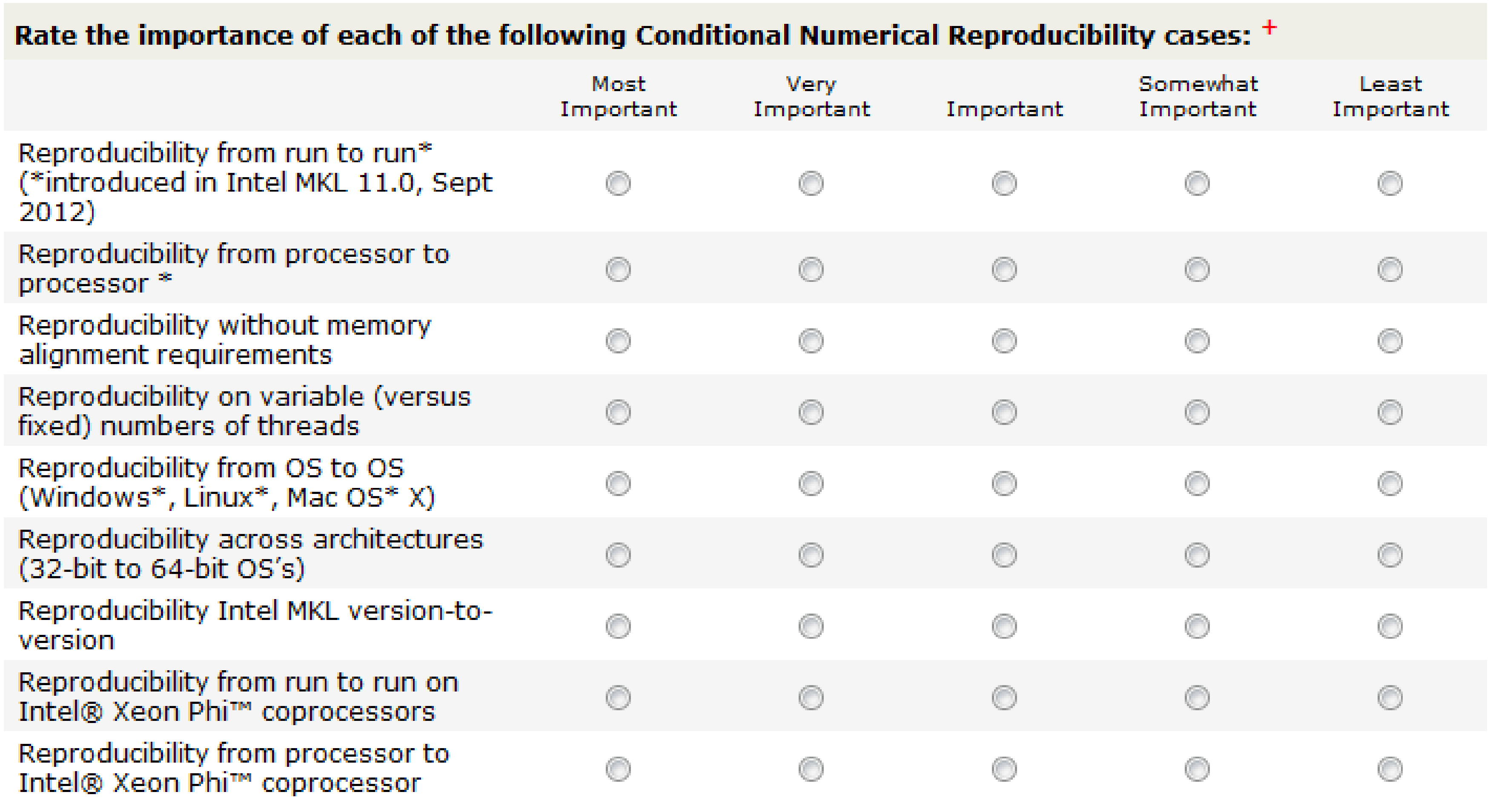


Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 11.0; Hardware: Intel® Xeon® Processor E5-2690, 2 Eight-Core CPUs (20MB LLC, 2.9GHz), 32GB of RA and Intel® Xeon® Processor X5680, 2 Six-Core CPUs (12MB LLC, 3.33GHz), 48GB of RAM;; Operating System: RHEL 6 GA x86_64; Benchmark Source: Intel Corporation.

Test environment: 64-bit executable, Matrix 40k x 40k, OMP_NUM_THREADS=12

Optimization Notice

What’s next?



Rate the importance of each of the following Conditional Numerical Reproducibility cases: +

	Most Important	Very Important	Important	Somewhat Important	Least Important
Reproducibility from run to run* (*introduced in Intel MKL 11.0, Sept 2012)	○	○	○	○	○
Reproducibility from processor to processor *	○	○	○	○	○
Reproducibility without memory alignment requirements	○	○	○	○	○
Reproducibility on variable (versus fixed) numbers of threads	○	○	○	○	○
Reproducibility from OS to OS (Windows*, Linux*, Mac OS* X)	○	○	○	○	○
Reproducibility across architectures (32-bit to 64-bit OS’s)	○	○	○	○	○
Reproducibility Intel MKL version-to-version	○	○	○	○	○
Reproducibility from run to run on Intel® Xeon Phi™ coprocessors	○	○	○	○	○
Reproducibility from processor to Intel® Xeon Phi™ coprocessor	○	○	○	○	○

https://softwareproductsurvey.intel.com/survey/150072/1afd/

Optimization Notice

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