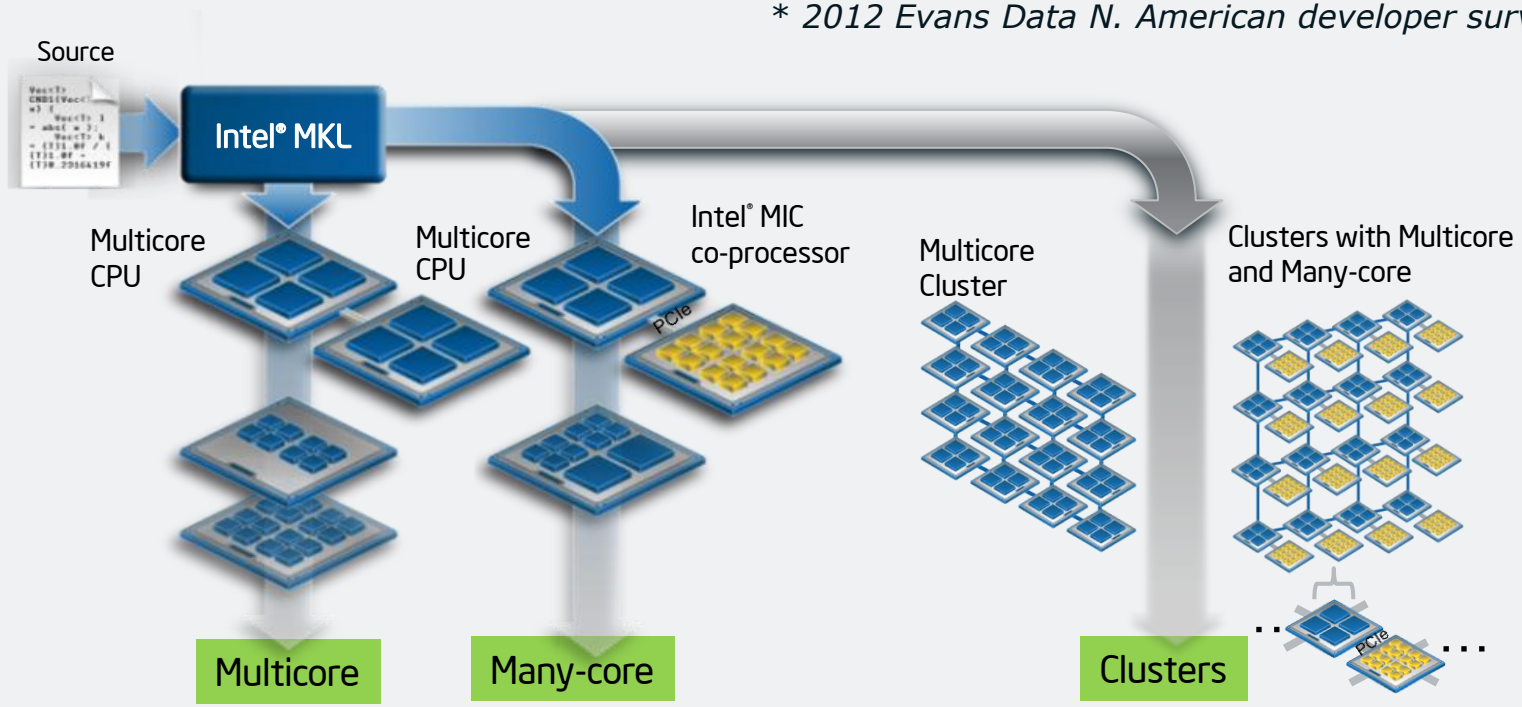
Intel® Math Kernel Library

Intel® MKL is industry's leading math library *

Linear Algebra	Fast Fourier Transforms	Vector Math	Vector Random Number Generators	Summary Statistics	Data Fitting
•BLAS •LAPACK •Sparse solvers •ScaLAPACK	•Multidimensional (up to 7D) •FFTW interfaces •Cluster FFT	•Trigonometric •Hyperbolic •Exponential, Logarithmic •Power / Root •Rounding	•Congruential •Recursive •Wichmann-Hill •Mersenne Twister •Sobol •Neiderreiter •Non-deterministic	•Kurtosis •Variation coefficient •Quantiles, order statistics •Min/max •Variance-covariance •…	•Splines •Interpolation •Cell search

** 2012 Evans Data N. American developer survey*



What Is Intel Math Kernel Library?

Performance, Performance, Performance!

Industry's leading math library (* *2012 Evans Data N. American developer survey*)

Addresses:

- Linear equation Solvers
- Eigenvector/Eigenvalue solvers
- PDEs, signal processing, seismic, solid-state physics (FFTs)
- General scientific, financial [vector transcendental functions (VML) and vector random number generators (VSL)]
- Sparse Solvers (PARDISO, DSS and ISS)
- Data fitting functions, Spline construction, interpolation, extrapolation, cell search

Tuned for Intel processors – current and the next generation

Vectorized, threaded, and distributed multiprocessor aware

Third-party Tools Powered by Intel MKL

IMSL* Fortran Numerical Libraries (Rogue Wave)

NAG* Libraries

MATLAB* (MathWorks)

GNU Octave*

NumPy* / SciPy*

PETSc* (Portable Extensible Toolkit for Scientific Computation)

WRF* (Weather Research & Forecsting run-time environment)

The HPCC* benchmark

And more …

Where Does the Parallelism Come From?

Domain	SIMD	Open MP
BLAS 1, 2, 3	X	X
FFTs	X	X
LAPACK (dense LA solvers)	X (relies on BLAS 3)	X
ScaLAPACK (cluster dense LA solvers)		X (hybrid)
PARDISO (sparse solver)	X (relies on BLAS 3)	X
VML/VSL	X	X
Cluster FFT		X

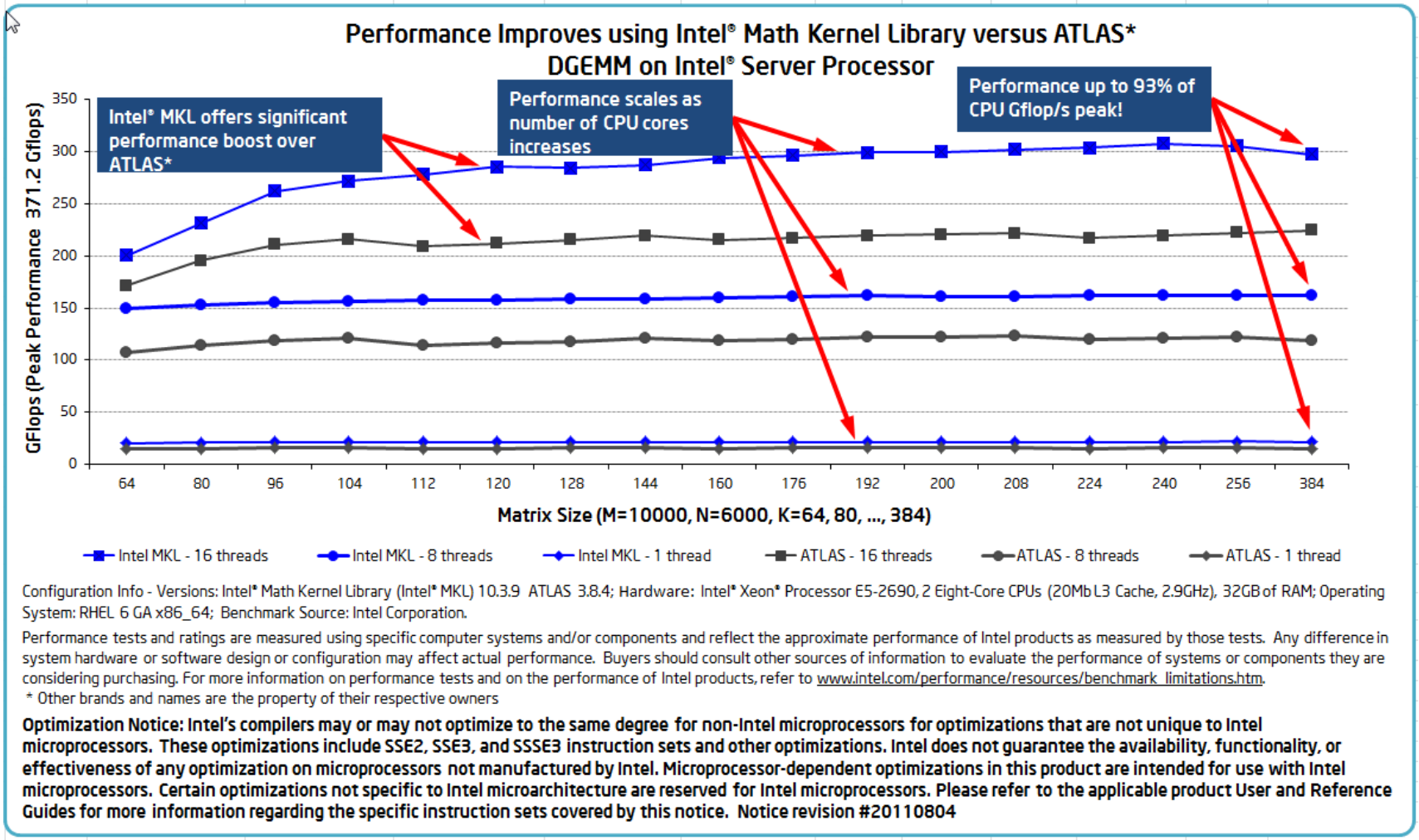
Why is Intel MKL faster?

Optimization done for maximum speed.

Resource limited optimization – exhaust one or more resource of system:

- **CPU**: Vectorization, Register use, FP units.
- **Cache**: Keep data in cache as long as possible; deal with cache interleaving.
- **TLBs**: Maximally use data on each page.
- **Memory bandwidth**: Minimally access memory.
- **Computer**: Use all the processor cores available using threading.
- **System**: Use all the nodes available. Optimized for Intel® MPI.

DGEMM Performance (Intel MKL vs. ATLAS*)



Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 10.3.9 ATLAS 3.8.4; Hardware: Intel® Xeon® Processor E5-2690, 2 Eight-Core CPUs (20Mb L3 Cache, 2.9GHz), 32GB of RAM; Operating System: RHEL 6 GA x86_64; Benchmark Source: Intel Corporation.

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

Intel MKL Components

BLAS

- Basic vector-vector/matrix-vector/matrix-matrix computation routines.

Sparse BLAS

- BLAS for sparse vectors/matrices

LAPACK (Linear algebra package)

- Solvers and Eigenvalue solvers. Many hundreds of routines total!
- C interface to LAPACK

ScaLAPACK

- Computational, driver and auxiliary routines for distributed-memory architectures

DFTs (General FFTs)

- Mixed radix, multi-dimensional transforms, FFTW interfaces

Sparse Solvers (PARDISO, DSS and ISS)

- Direct and Iterative sparse solvers for symmetric, structurally symmetric or non-symmetric, positive definite, indefinite or Hermitian sparse linear system of equations
- Out-Of-Core (OOC) version for huge problem sizes

Intel MKL Components (cont'd)

VML (Vector Math Library)

- Set of vectorized transcendental functions, most of libm functions, but faster

VSL (Vector Statistical Library)

- Set of vectorized random number generators
- SSL (Summary Statistical Library) : Computationally intensive core/building blocks for statistical analysis

DFL (Data Fitting Library)

- Linea, quadratic, cubic, step-wise const, and user-defined Splines
- Cell search with configuration parameters for optimal performance
- User defined interpolation & extrapolation

PDEs (Partial Differential Equations)

- Trigonometric transform and Poisson solvers.

Optimization Solvers

- Solvers for nonlinear least square problems with/without constraints.

Support Functions

Intel MKL Environment

	Linux*
Compiler	Intel, Gnu
Libraries	.a, .so

64 bit static and dynamic libraries

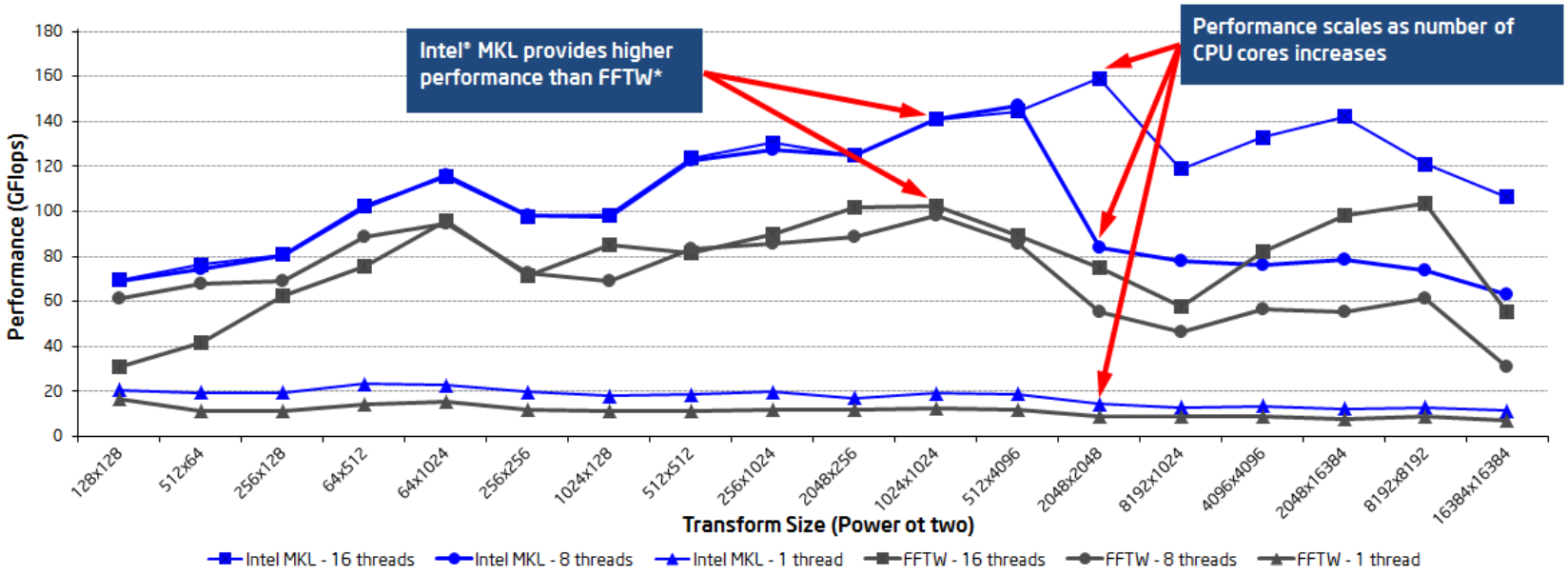
Language Support	
Domain	**C/C++**
BLAS	Via CBLAS
Sparse BLAS Level 1	Via CBLAS
Sparse BLAS level 1&2	X
LAPACK	X
PARDISO	X
DSS & ISS	X
VML/VSL	X
FFT	X
PDEs	X
Optimization (TR) Solvers	X
SSL	X

Intel® MKL: Fast Fourier Transform (FFT)

- Single and double precision complex and real transforms.
  - 1, 2, 3 and multidimensional transforms
- Multithreaded and thread-safe.
- Transform sizes: 2-powers, mixed radix, prime sizes
  - Transforms provide for efficient use of memory and meet the needs of many physical problems. *Any* size transform can be specified, but not all transform sizes run equally fast.
- User-specified scaling supported.
- Multiple transforms on single call.
- Strides
  - Allow FFT of a part of image, padding for better performance, transform combined with transposition, facilitates development of mixed-language applications.
- Integrated FFTW interfaces
  - Source code of FFTW3 and FFTW2 wrappers in C/C++ and Fortran are provided.
  - FFTW3 wrappers are also built into the library.
  - Not all FFTW features are supported.

Fast Fourier Transform *Performance*

Threading Optimizations



Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 11.0 , FFTW* 3.3.2; Hardware: Intel® Xeon® Processor E5-2690, 2 Eight-Core CPUs (20MB LLC, 2.9GHz), 32GB of RAM; Operating System: RHEL 6 GA x86_64; Benchmark: Single precision complex 2-dimension FFT, data may have been padded to avoid cach thrashing, source: Intel Corporation.

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

Intel® MKL FFT Interface (DFTI)

(see also http://portal.acm.org/citation.cfm?id=1114271)

Overview

- DFTI_DESCRIPTOR_HANDLE — pointer to an opaque structure
- The 5-stage usage model: Create, Configure$_{opt}$, Commit, Compute, Free
- Numerous parameters for Configure$_{opt}$

Example (configuring this $F_M \otimes I_N \otimes F_K$):

- `DftiCreateDescriptor(&hand, DFTI_SINGLE, DFTI_COMPLEX, 2, &{M,K} );`
- `DftiSetValue(hand, DFTI_INPUT_STRIDES, &{0,NK,1} ); /* row-major */`
- `DftiSetValue(hand, DFTI_NUMBER_OF_TRANSFORMS, N );`
- `DftiSetValue(hand, DFTI_INPUT_DISTANCE, K );`
- `DftiCommitDescriptor(hand);`
- `loop (call this repeatedly to compute arbitrary number of FFTs)`
  - `DftiComputeForward(hand, X, Y);`
  - `DftiComputeBackward(hand, Y, X); /* caution: Y uses input strides */`
- `DftiFreeDescriptor(&hand)`

DFTI Functions

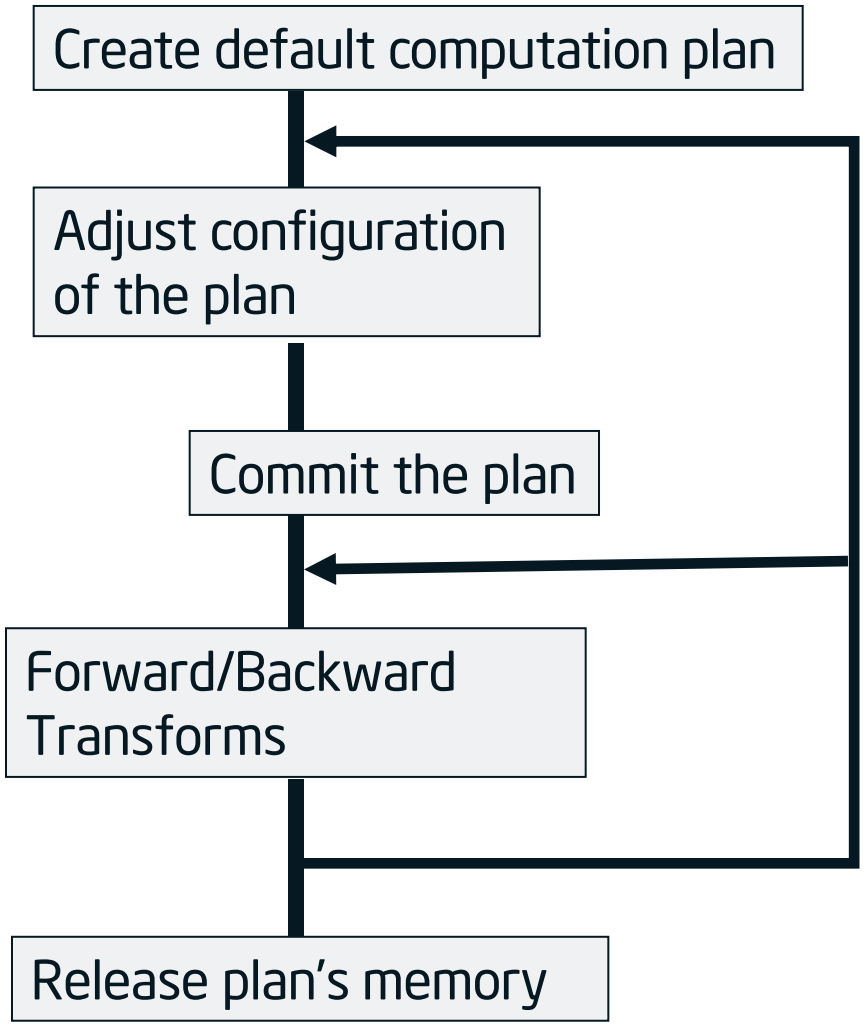
DftiCreateDescriptor

DftiSetValue

DftiCommitDescriptor

DftiComputeForward
DftiComputeBackward

DftiFreeDescriptor



DFTI Example

- Complex-to-complex 1D transform, double precision, not in place.

```
/* Create a descriptor */
Status = DftiCreateDescriptor( &Desc_Handle, DFTI_DOUBLE,
                               DFTI_COMPLEX, 1, n );

/* Set placement of result: DFTI_NOT_INPLACE */
Status = DftiSetValue(Desc_Handle, DFTI_PLACEMENT,
                               DFTI_NOT_INPLACE);

/* Commit the descriptor */
Status = DftiCommitDescriptor( Desc_Handle );

/* Compute a forward transform */
Status = DftiComputeForward(Desc_Handle, x_in, x_out);
```

DFTI Example (continue)

```
/* Set Scale number for backward transform */
Scale = 1.0/(double)n;
Status = DftiSetValue( Desc_Handle, DFTI_BACKWARD_SCALE,
                       Scale );

/* Commit the change made to the descriptor */
Status = DftiCommitDescriptor( Desc_Handle );

/* Compute a backward transform */
Status = DftiComputeBackward( Desc_Handle, x_out, x_in );

/*  Free the descriptor */
Status = DftiFreeDescriptor( &Desc_Handle );
```

FFTW API (see http://www.fftw.org)

Overview

- fftw_plan — pointer to an opaque structure, created by planners.
- Many planners
  - problem types: dft, r2c, c2r, and r2r (limited support in MKL).
  - data layout: complex vs split-complex, embedded data.
  - simple and guru interfaces.
- Wisdom management.

Example (computing $F_M \otimes I_N \otimes F_K$):

- plan *fwd = **fftw_plan_guru_dft**(**2**,&{{K,1,1},{M,NK,NK}},
  1,&{{N,K,K}},X,Y,FFTW_FORWARD,FFTW_PATIENT)
- `plan *bwd = fftw_plan_guru_dft(...,Y,X,FFTW_BACKWARD,FFTW_PATIENT)`
- loop
  - `fftw_execute(fwd);`
  - `fftw_execute(bwd);`

Compute FFT as many times as you like, with data contained in arrays X and Y.
Alternatively, use new-array execute functions, like
`fftw_execute_dft( fwd, another_X, another_Y )`

- `fftw_destroy_plan(fwd);`
- `fftw_destroy_plan(bwd);`

FFTW Usage Model

Setup

- `plan p = plan_dft(rank,dims,X,Y,sign,flags)`
- `plan_dft_1d(n,…), …_2d(nx,ny,…),…_3d(nx,ny,nz,…)`
- `FFTW_ESTIMATE | _MEASURE | _PATIENT | _EXHAUSTIVE`
- In-place or out-of-place
- Alignment
- Measurement (unless FFTW_ESTIMATE)

Execution

- `execute_dft(p,X,Y),`
  `execute_split_dft(p,Xr,Xi,Yr,Yi)`

Cleanup

- `destroy_plan(p)`

MKL FFTW Interface via Wrappers

Note: The FFTW3 wrappers are built as part of library. Users don't need to build by themselves.

```
/* Create & Commit a descriptor for 1D forward transform */
plan = fftw_plan_dft_1d( n, x_in, x_out,
                         FFTW_FORWARD,FFTW_ESTIMATE );

/* Compute forward DFT*/
fftw_execute( plan );

/* Set Scale number for Backward transform */
Scale = 1.0/(double)n;
```

MKL FFTW Interface via Wrappers (continue)

```
/* Create & Commit a descriptor for 1D backward transform */
Desc_Handle = fftw_plan_dft_1d( n, x_out , x_in,
                                FFTW_BACKWARD,FTW_ESTIMATE );

/* Compute backward DFT */
fftw_execute(Desc_Handle);

/* Free Dfti descriptor */
fftw_destroy_plan(Desc_Handle);

/*  Result scaling  */
scaling_d(x_in, Scale, n);
```

Performance Tips

- Split plan creation and computation.
  - "plan + compute in one function" is a bad usage model
- Rely on MKL's threaded FFT functions.
  - instead of calling sequential FFT functions on multiple threads.
- Use bundled transforms where possible.
- Know optimized radices: 2, 3, 5, 7, 11, 13.
- Align data to help vector load/store.
- Avoid cache-thrashing alignments (e.g. 2048x2048) by padding.

Performance Tips (continue)

- Avoid thread migration by setting thread affinity.
  - KMP_AFFINITY=compact,granularity=fine
  - Know processor topology (topology enumeration software from Intel)
- Skip hyper-threads, if Hyper-threading is enabled.
  - e.g. KMP_AFFINITY=compact,1,0,…
- Interleave memory placement on NUMA systems.
  - e.g. numactl –interleave=all ./a.out

Summary of FFT Support

- Intel MKL FFTs support 1, 2, 3 and multidimensional transforms.
- Mixed Radix Support.
- Multithreaded for 1, 2, 3 and multidimensional transforms.
- Scales very well on multi-core systems (single node) and across many nodes in clusters.

Why Do Results Vary From Run To Run?

Root cause for variations in results

- floating-point numbers ➔ order of computation matters!
- double precision arithmetic example (a+b)+c ≠ a+(b+c)

$2^{-63} + 1 + -1 = 2^{-63}$ (infinitely precise result)

$(2^{-63} + 1) + -1 = 0$ (correct IEEE single precision result)

$2^{-63} + (1 + -1) = 2^{-63}$ (correct IEEE single precision result)

- **Why does the order of operations change in Intel MKL?**
- Memory alignment — affects grouping of data in registers
- Number of threads — most functions are threaded to use all available cores
- Dynamic task scheduling — some algorithms use asynchronous task scheduling for optimal performance
- Different code branches — optimized to use all proc. resources

Order matters when doing floating point arithmetic.
Many optimizations require a change in order of operations.

New Feature: Balancing Performance and Reproducibility

	Memory alignment	• Align memory — try Intel MKL memory allocation functions • 64-bytes alignment for processors in the next few years
	Number of threads	• Set the number of threads to a constant number • Use sequential libraries
New!	Dynamic task scheduling	• New control to specify static scheduling • *Previous sequential operation was required*
New!	Code branches	• New controls to maintain consistent code branches across different processors • Means most optimized code path might be used

Intel MKL introduces functions to broaden the cases where reproducibility is possible.

Example – SSE2

For the same results on every Intel processor that supports SSE2 instructions or later

- function call

```
mkl_cbwr_set(MKL_CBWR_SSE2)
```

- or environment variable

```
set MKL_CBWR_BRANCH="SSE2"
```

The full list of options:

COMPATIBLE	3
SSE2	4
SSE3	5
SSSE3	6
SSE4_1	7
SSE4_2	8
AVX	9
AVX2	10

Note: on non-Intel processors the results may differ since only MKL_CBWR_COMPATIBLE branch is supported

Example – Find out the best performing option from a pool of processors

For the best option given a pool of computing resources in a grid setting, you may launch a simple program as follows

```
#include <mkl.h>
int main(void) {
  int my_cbwr_branch;
  /* Find the available MKL_CBWR_BRANCH */
  my_cbwr_branch = mkl_cbwr_get_auto_branch();
  if (!mkl_cbwr_set(my_cbwr_branch)) {
    printf("Error in setting branch. Aborting…\n");
    return;}
  return my_cbwr_branch;
}
```

The full list of options:

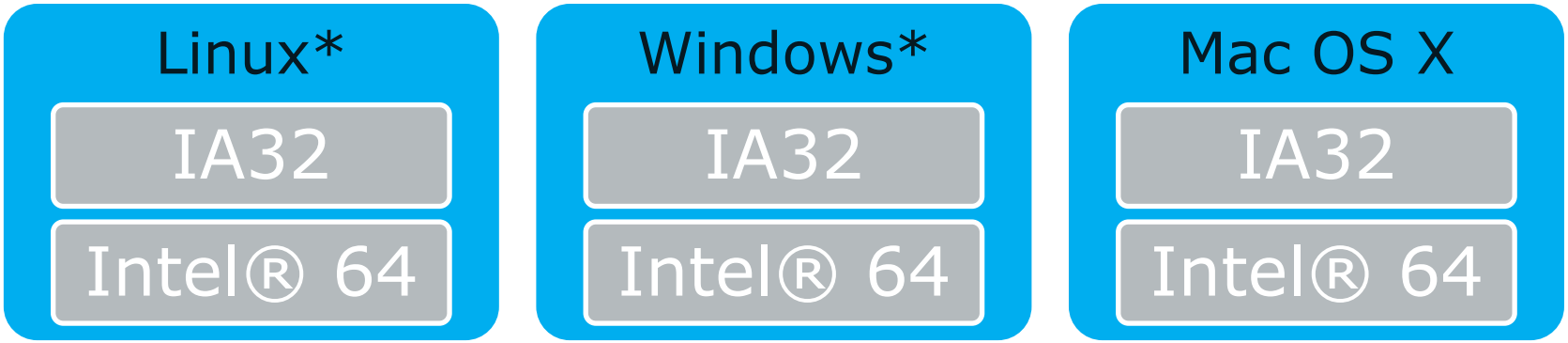
COMPATIBLE	3
SSE2	4
SSE3	5
SSSE3	6
SSE4_1	7
SSE4_2	8
AVX	9
AVX2	10

Examine all results and use `mkl_cbwr_set(`<*minimum_result*>`)`

Conditional Bitwise Reproducibility

- Reproducibility is not guaranteed across…
  - operating systems
  - architectures

Reproducibility only applies within these blue boxes…



- Reproducibility across non-Intel processors is limited
- Reproducibility will not cover all future Intel processors
- Reproducibility will not be maintained from one version of Intel MKL to the next

New functionality expands the scope for reproducibility, but not to every case.

Conditions for Reproducibility

Function calls occur in a single executable, running on one platform (architecture/ OS)

Aligned input and output arrays in function calls

- 16-byte alignment for the family of SSE instruction sets
- 32-byte alignment for AVX1
- 64-byte alignment for AVX2 *<- choose this to be safe*

A constant number of computational threads used by the library throughout the run

Use of new functions/controls to ensure deterministic task scheduling and to control code branches

- control variables must be set or control functions called before any other Intel MKL functions

New optimizations

Optimizations using the new Intel® Advanced Vector Extensions 2 (AVX2) including the new FMA3 instructions—the following parts have optimizations:

- BLAS
- FFTs
- Vector math functions
- Data fitting functions
- Random number generators
- Summary statistics functions

Reference and FAQs

Intel® MKL product page:

- http://software.intel.com/en-us/intel-mkl/

Intel® MKL forum :

- http://software.intel.com/en-us/forums/intel-math-kernel-library/

MKL FFT related Knowledge Base articles

- http://software.intel.com/en-us/articles/fft-interfaces-in-intelr-mkl/
- http://software.intel.com/en-us/articles/different-parallelization-techniques-and-intel-mkl-fft/
- http://software.intel.com/en-us/articles/mkl-threaded-1d-ffts/
- http://software.intel.com/en-us/articles/intel-mkl-main-libraries-contain-fftw3-interfaces/

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.

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