Intel® MKL
Fast Fourier Transform (FFT)
**Agenda**

Overview

Implementation of DFT
- MKL implementation and API
- FFTW implementation and API

Performance overview
- Basic cases
- Parallelization
- Performance improvement tips

Summary
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

2D FFT Performance Boost by using Intel® Math Kernel Library versus FFTW*

- Intel® MKL provides higher performance than FFTW*
- Performance scales as number of CPU cores increases

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 L2, 2.9GHz), 32GB of RAM; Operating System: RHEL 5 GA x86_64; Benchmark: Single precision complex 2-dimension FFT, data may have been padded to avoid each 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 the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.html.

* 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: Cluster FFT

• Single and double precision complex and real transforms.
  – 1, 2, 3 and multidimensional transforms
• Transform sizes: 2-powers and mixed radix.
• Similar interface to Intel MKL DFTI.
• Work with MPI through BLACS
  – Support Intel MPI, Open MPI, and MPICH.
• Integrated FFTW interfaces
  – FFTW3 and FFTW2 wrappers in C/C++ and Fortran.
  – FFTW3 wrappers are also built in the library.
Cluster FFT Performance

- 2D FFT on a cluster of 512 cores (32 nodes, 16 cores per node)
Cluster FFT Scalability

Cluster 3D FFT Maximum Performance

Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 11.0, FFTW 3.3.2;
Hardware of cluster nodes: Intel® Xeon® Processor E5-2670 2 Eight-Core CPUs (20MB LLC, 2.6GHz), 64GB of RAM; Operating System: RHEL 6.1 GA x86_64;
Benchmark: Double precision complex 3-dimension FFT, source: Intel Corporation; Software: Intel® MPI 4.0.3.008, Intel® C++ Compiler 12.1.137;
Cluster configuration: Number of MPI processes per node - 16; OMF_NUM_THREADS = 1; Interconnects - FDR Infiniband.

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 effect 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/performancemanagement/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_ASSERT_DESCRIPTOR_HANDLE — pointer to an opaque structure
• The 5-stage usage model: Create, Configureopt, Commit, Compute, Free
• Numerous parameters for Configureopt

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
  - Create default computation plan
- DftiSetValue
  - Adjust configuration of the plan
- DftiCommitDescriptor
  - Commit the plan
- DftiComputeForward
- DftiComputeBackward
  - Forward/Backward Transforms
- DftiFreeDescriptor
  - Release plan’s memory
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 );
Cluster FFT Example

• Complex-to-complex 2D transform, single precision, not in-place.

/* Create a descriptor */
Status=DftiCreateDescriptorDM(MPI_COMM_WORLD,&desc,PREC,DFTI_COMPLEX,2,lengths);

/* Obtain some values of configuration parameters */
Status=DftiGetValueDM(desc,CDFT_LOCAL_SIZE,&size);

/* Specify a value(s) of configuration parameters */
Status=DftiSetValueDM(desc,CDFT_WORKSPACE,work);

/* Commit the descriptor */
Status = DftiCommitDescriptorDM( &desc );
Cluster FFT Example (continue)

/* Create arrays for local parts of input and output data */
Status=MKL_CDFT_ScatterData( MPI_COMM_WORLD,RootRank,
    ElementSize,2,lengths,x_in,
    nx,start_x,local );

/* Compute the transform */
Status=DftiComputeForwardDM( desc,local,work );

/* Gather data among processors */
Status=MKL_CDFT_GatherData( MPI_COMM_WORLD,RootRank,
    ElementSize,2,lengths,x_in,
    nx,start_x,work );

Repeat the above three steps for backward transform

/* Release memory allocated for a descriptor */
Status = DftiFreeDescriptorDM( &desc );

MKL_CDFT_ScatterData() and MKL_CDFT_GatherData() are not MKL functions. But users can
find example implementations in $MKLROOT/examples/cdftc/source/cdft_example_support.c
**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(\ldots,Y,X,FFTW_BACKWARD,FFTW_PATIENT)`
- loop
  - `fftw_execute(fwd);`
  - `fftw_execute(bwd);`
- `fftw_destroy_plan(fwd);`
- `fftw_destroy_plan(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 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.

```c
/* 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`
Reference and FAQs

Intel® MKL product page:


Intel® MKL forum:


MKL FFT related Knowledge Base articles


• http://software.intel.com/en-us/articles/mkl-threaded-1d-ffts/

Summary

• 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.
Backup Slides
Definition of Fourier Transform
(A not rigorous introduction)

\[ \hat{f}(\kappa) = \int_0^T e^{-2\pi i \kappa t} f(t) \, dt \]

**Spectrum estimate**

**Signal**

**Quantization**

N samples of \( f(t) \) at \( t=0, \Delta, 2\Delta, \ldots, (N-1)\Delta \), covering \( T=N\Delta \)

\[ \hat{f} \left( \frac{k}{N\Delta} \right) = \sum_{n=0}^{N-1} e^{-\frac{2\pi i}{N} kn} f(n\Delta) \]

Discrete Fourier transform comes as the result of straightforward discretization of the Fourier integral.
Definition of DFT: 1D DFT

\[ y = F_N x \]
\[ y_k = \sum_{n=0}^{N-1} \omega_N^{kn} x_n \]
\[ \omega_N = e^{-\frac{2\pi i}{N}} \]

\[ F_N = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{N-2} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{N-3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{N-2} & \omega^{N-3} & \cdots & \omega \end{pmatrix} \]

\[ F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad F_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega_3 & \overline{\omega_3} \\ 1 & \overline{\omega_3} & \omega_3 \end{pmatrix} \]

\[ F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \]

Naïve matrix-vector multiply is \(O(N^2)\) flops

Use of symmetries in \(F_N\) \(\rightarrow\) \(O(N \log N)\) algorithms
Some Tensor Algebra — Symbols

\[
F_N = \begin{pmatrix}
1 & 1 & 1 & \cdots \\
1 & \omega_N & \omega_N^2 & \cdots \\
1 & \omega_N^2 & \omega_N^4 & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{pmatrix}
\]

\[
I_N = \begin{pmatrix}
1 & & & \\
& \ddots & & \\
& & 1 & \\
& & & 1
\end{pmatrix}
\]

Properties

\[
F_N = F_N^T
\]

\[
L_K^{MK} L_M^{MK} = I_{MK}
\]

\[
F_N F_N = NI_N
\]

\[
L_M^{MKS} = (L_M^{MK} \otimes I_S)(I_K \otimes L_M^{MS})
\]

\[
F_N^2 = N \text{perm}_N
\]

\[
L_M^{MK} = (I_M \otimes L_K^{KS})(L_M^{MS} \otimes I_K)
\]

\[
F_N^4 = N^2 I_N
\]

\[
L_K^{MK} T_K^{MK} = T_M^{MK} L_K^{MK}
\]

\[
T_K^{MK} = \text{diag}_{MK}(\omega_\epsilon^{f(i)})
\]

\[
T_2^6 = \begin{pmatrix}
1 & & & & & \\
& 1 & & & & \\
& & \omega_\epsilon & & & \\
& & & 1 & & \\
& & & & \omega_\epsilon^2 & \\
& & & & & 1
\end{pmatrix}
\]

\[
L_3^6 = \begin{pmatrix}
1 & & & & & \\
& \ddots & 1 & & & \\
& & \ddots & \ddots & & \\
& & & \ddots & \ddots & \\
& & & & \ddots & \ddots \\
& & & & & 1
\end{pmatrix}
\]
Some Tensor Algebra — Tensor Product

\[ A \otimes B = C = \{a_{ki}B\} \]
\[ A \quad \text{— } K \times L \text{ matrix} \]
\[ B \quad \text{— } M \times N \text{ matrix} \]
\[ C \quad \text{— } KM \times LN \text{ matrix} \]

\[ A \otimes B \neq B \otimes A \]

\[
A = \begin{pmatrix}
    a_{00} & a_{01} & a_{02} & \cdots \\
    a_{10} & a_{11} & a_{12} & \cdots \\
    a_{20} & a_{21} & a_{22} & \cdots \\
    \vdots & \vdots & \vdots & \ddots
\end{pmatrix}
\]

\[ A^\top = I_N^{MN} A^\downarrow \]

transposition
Some Tensor Algebra — p/v Operation

\[
F_N \otimes I_M = \begin{pmatrix}
I_M & I_M & I_M & \cdots \\
I_M & \omega_N I_M & \omega_N^2 I_M & \cdots \\
I_M & \omega_N^2 I_M & \omega_N^4 I_M & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{pmatrix}
\quad \text{vector operation}
\]

\[
I_M \otimes F_N = \begin{pmatrix}
F_N \\
F_N \\
\vdots \\
F_N
\end{pmatrix}
\quad \text{parallel operation}
\]

\[
I_M \otimes F_N \otimes I_K
\]

mixed operation

2D DFT

\[
F_M \otimes F_N = (F_M \otimes I_N)(I_M \otimes F_N)
\]
row-col algorithm

\[
= L_M^{NM} (I_N \otimes F_M) L_N^{MN} (I_M \otimes F_N)
\]
\|DFT with transposition

blocking: \[
F_N \otimes I_M \overset{M=M'}{=} L^T (I_{M'} \otimes F_N \otimes I_B)L = \sum_{m'=0}^{M'-1} S(F_N \otimes I_B)g
\]

D-dimensional DFT

\[
F_{N_1} \otimes F_{N_2} \otimes \cdots \otimes F_{N_D}
\]

\[
= \prod_{d=1}^{D} I_\ast \otimes F_{N_d} \otimes I_\ast
\]

fundamental factorization: tensor product \rightarrow matrix product

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
DFT Factorizations

Cooley-Tukey: The basis for DFT is Cooley-Tukey factorization.

\[
F_{MN} \rightarrow (F_M \otimes I_N) T_N (I_M \otimes F_N) L_M
\]

Vector form, twiddle factor, parallel M transforms of size N, permutation

\[
f_{\text{op}}(F_{MN}) = N f_{\text{op}}(F_M) + NM + M f_{\text{op}}(F_N) + 0
\]

\[
\frac{f_{\text{op}}(F_{MN})}{MN} = \frac{f_{\text{op}}(F_M)}{M} + \frac{f_{\text{op}}(F_N)}{N} + 1
\]

\[
\Rightarrow f_{\text{op}}(F_N) = \mathcal{O}(N \log N) \approx 5 N \log_2 N
\]

Other (Q,E,B,D — sparse matrices)

\[
F_{MN} \rightarrow Q^T (F_M \otimes F_N) Q \quad [\gcd(M, N) = 1]
\]

\[
F_N \rightarrow Q^T (I_1 \oplus F_{N-1}) E_N (I_1 \oplus F_{N-1}) Q \quad [N \text{ prime}]
\]

\[
F_N \rightarrow B_{M,N}^T D_M F_M D_M' F_M D_M'' B_{M,N} \quad [M \geq 2N - 1]
\]

- Not only 2-powers
- Multiple algorithms
- \( || \) vs vector
- “bad” sizes

Prime-Factor algorithm
Rader algorithm
Bluestein Algorithm
**FFTW API: N/V Tensors and Plans**

(\(N, is, os\))
I/O dimension

\[
\text{REAL } A(N,M) \quad (N,1,1)(M,N,N) \\
\text{float } B[N][M]; \quad (N,M,M)(M,1,1) \\
A = \text{transpose}(A) \quad (N,1,M)(M,N,1) \\
I_K \otimes F_M \otimes F_N \quad \left(\begin{array}{c}
N,1,1 \\
M,N,N \\
L_N^M
\end{array}\right) \cdot (K,MN,MN)
\]

Planning and wisdom

\[
F_{MN} \rightarrow (F_M \otimes I_N) T_N (I_M \otimes F_N) L_M
\]

\[
F_{30} \rightarrow [F_3, F_{10}] \text{ or } [F_{10}, F_3] \text{ or } [F_5, F_6] \text{ or } \ldots?
\]
Thank You
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