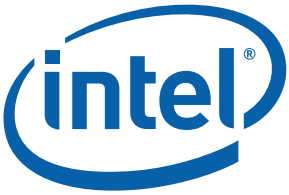


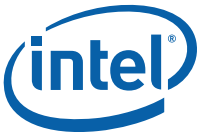
Intel® Math Kernel Library FFT to DFTI Wrappers

Technical User Notes

September 2006

Document Number: A314775-001US

World Wide Web: http://developer.intel.com



Version	Version Information	Date
-001	Documents Intel® Math Kernel Library (Intel® MKL) 9.0 FFT to DFTI Wrappers.	09/2006

Contents

Installation

Overview

1

Intel® Math Kernel Library (Intel® MKL) FFT to DFTI wrappers allow the Intel® Math Kernel Library Fast Fourier Transform (FFT) interface to call another Intel® MKL Fourier transform interface (DFTI). The FFT interface is removed from Intel® MKL 9.0 and the wrappers provide compatibility with previous versions of the library.

About the Wrappers

The purpose of this collection of wrappers is to enable developers whose programs currently use the FFT interface, which is no longer supported, to continue using Intel MKL Fourier transforms without changing the program source code.

Because of the differences between FFT and DFTI functionalities, 1D FFTs are parallelized only on Intel® Itanium® processors in the current version of Intel MKL. The functionality differences also cause some performance drop when using wrappers instead of FFT C-interface functions. Fortran-interface functions, on the contrary, gain DFTI performance advantage.

Technical Support

Visit the Intel MKL support website at
http://www.intel.com/support/performancetools/libraries/mkl/
for self help information, including getting started tips, known product issues, product errata, license information, user forums, and more.

About This Document

These technical user notes provide the wrappers reference for Fortran and C interfaces and describe the wrappers installation procedure, including creation of the wrapper library, applications assembling and running examples (see Installation).

The Wrappers Reference section contains reference information on each wrapper. The routines are described in groups by two differing only in data precision. Each group is introduced by the routine names, a short description of their purpose, and the calling sequence, or syntax, for each type of data with which the routines are used. The following sections are also included for each routine:

Description	Describes the operation performed by routines.
Input Parameters	Defines the data type for each parameter on entry, for example: *r* COMPLEX for `cfft1d` DOUBLE COMPLEX for `zfft1d`
Output Parameters	Lists resultant parameters on exit.

Notational Conventions

The following font conventions are used in this document:

UPPERCASE MONOSPACE	Data types used in the discussion of input and output parameters for Fortran interface. For example, `DOUBLE COMPLEX`.
`lowercase monospace`	Data types used in the discussion of input and output parameters for C interface, for example, `double*;` denotation of mathematical functions, for example, *t* `= cmplx(`*r*`,0);` routine names, for example, `csfft1dc;` file and path names, for example, `mkl_dfti.h`; command names in installation discussion, for example, `make lib64`.
`lowercase monospace italic`	Arguments and parameters in different contexts. For example, *`wsave`* in routine parameters discussion or *`platform`* in library creation instructions.

Wrappers Reference 2

This chapter contains the following major parts:

- One-dimensional FFTs
- Two-dimensional FFTs

Each part contains the description of three groups of the FFTs.

One-dimensional FFTs

The one-dimensional FFTs include the following groups:

- Complex-to-Complex Transforms
- Real-to-Complex Transforms
- Complex-to-Real Transforms.

All one-dimensional FFTs are in-place. The transform length must be a power of 2. The complex-to-complex transform routines perform both forward and inverse transforms of a complex vector. The real-to-complex transform routines perform forward transforms of a real vector. The complex-to-real transform routines perform inverse transforms of a complex conjugate-symmetric vector, which is packed in a real array.

Data Storage Types

Each FFT group contains two sets of FFTs having the similar functionality: one set is used for the Fortran interface and the other for the C interface. The former set stores the complex data as a Fortran complex data type, while the latter stores the complex data as float arrays of real and imaginary parts separately. These sets are distinguished by naming the FFTs within each set. The names of the FFTs used for the C interface have the letter “c” added to the end of the FFTs’ Fortran names. For example, the names of the `cfft1d/zfft1d` FFTs for the corresponding C-interface routines are `cfft1dc/zfft1dc`. All names of the C-type data items are lower case.

Table 2-1 lists the one-dimensional FFT routine groups and the data types associated with them.

Table 2-1 One-dimensional FFTs: Names and Data Types

Group	Stored as Fortran Complex Data	Stored as C Real Data	Data Types	Description
Complex-to-Complex	cfft1d/ zfft1d	cfft1dc/ zfft1dc	c, z	Transform complex data to complex data.
Real-to-Complex	scfft1d/ dzfft1d	scfft1dc/ dzfft1dc	sc, dz	Transform forward real-to-complex data. Complement csfft1d/zdfft1d and csfft1dc/zdfft1dc FFTs.
Complex-to-Real	csfft1d/ zdfft1d	csfft1dc/ zdfft1dc	cs, zd	Transform inverse complex-to-real data. Complement scfft1d/dzfft1d and scfft1dc/dzfft1dc FFTs.

Data Structure Requirements

For C interface, storage of the complex-to-complex transform routines data requires separate float arrays for the real and imaginary parts. The real-to-complex and complex-to-real pairs require a single float input/output array.

The C interface requires scalar values to be passed by value.

All transforms require additional memory to store the transform coefficients. When performing multiple FFTs of the same dimension, the table of coefficients should be created only once and then used on all the FFTs afterwards. Using the same table rather than creating it repeatedly for each FFT produces an obvious performance gain.

Complex-to-Complex One-dimensional FFTs

Each of the complex-to-complex routines computes a forward or inverse FFT of a complex vector.
The forward FFT is computed according to the mathematical equation

$$z_j = \sum_{k=0}^{n-1} r_k * w^{-j*k}, \quad 0 \le j \le n-1$$

The inverse FFT is computed according to the mathematical equation

$$r_j = \frac{1}{n}\sum_{k=0}^{n-1} z_k * w^{j*k}, \quad 0 \le j \le n-1$$

where $w = \exp\left[\frac{2\pi i}{n}\right]$, *i* being the imaginary unit.

The operation performed by the complex-to-complex routines is determined by the value of the *isign* parameter used by each of these routines.

If *isign* = -1, perform the forward FFT where input and output are in normal order.

If *isign* = +1, perform the inverse FFT where input and output are in normal order.

If *isign* = -2, perform the forward FFT where input is in normal order and output is in bit-reversed order.

If *isign* = +2, perform the inverse FFT where input is in bit-reversed order and output is in normal order.

If *isign* = 0, initialize FFT coefficients for both the forward and inverse FFTs.

The above equations apply to all FFTs with all data types indicated in Table 2-1.

To compute a forward or inverse FFT of a given length, first initialize the coefficients by calling the function with *isign* = 0. Thereafter, any number of transforms of the same length can be computed by calling the function with *isign* = +1, -1, +2, -2.

cfft1d/zfft1d

Fortran-interface routines. Compute the forward or inverse FFT of a complex vector (in-place).

Syntax

```
call cfft1d( r, n, isign, wsave )
call zfft1d( r, n, isign, wsave )
```

Description

The operation performed by the cfft1d/zfft1d routines is determined by the value of *isign*. See the equations of the operations for the Complex-to-Complex One-dimensional FFTs above.

Input Parameters

r — COMPLEX for cfft1d
DOUBLE COMPLEX for zfft1d
Array, DIMENSION at least (*n*). Contains the complex vector on which the transform is to be performed. Not referenced if *isign* = 0.

n — INTEGER. Transform length; *n* must be a power of 2.

isign — INTEGER. Flag indicating the type of operation to be performed:
if *isign* = 0, initialize the coefficients *wsave*;
if *isign* = -1, perform the forward FFT where input and output are in normal order;
if *isign* = +1, perform the inverse FFT where input and output are in normal order;
if *isign* = -2, perform the forward FFT where input is in normal order and output is in bit-reversed order;
if *isign* = +2, perform the inverse FFT where input is in bit-reversed order and output is in normal order.

wsave — COMPLEX for cfft1d
DOUBLE COMPLEX for zfft1d
Array, DIMENSION at least ((3*n)/2). If *isign* = 0, then *wsave* is an output parameter. Otherwise, *wsave* contains the FFT coefficients initialized on a previous call with *isign* = 0.

Output Parameters

r — Contains the complex result of the transform depending on *isign*. Does not change if *isign* = 0.

wsave — If *isign* = 0, wsave contains the initialized FFT coefficients. Otherwise, *wsave* does not change.

cfft1dc/zfft1dc

C-interface routines. Compute the forward or inverse FFT of a complex vector (in-place).

Syntax

```
void cfft1dc(float* r, float* i, int n, int isign, float* wsave)
void zfft1dc(double* r, double* i, int n, int isign, double* wsave)
```

Description

The operation performed by the cfft1dc/zfft1dc routines is determined by the value of *isign*. See the equations of the operations for the Complex-to-Complex One-dimensional FFTs.

Input Parameters

r — float* for cfft1dc
double* for zfft1dc
Pointer to an array of size at least (*n*). Contains the real parts of complex vector to be transformed. Not referenced if *isign* = 0.

i — float* for cfft1dc
double* for zfft1dc

Pointer to an array of size at least (*n*). Contains the imaginary parts of complex vector to be transformed.

Not referenced if *isign* = 0.

n — int. Transform length; *n* must be a power of 2.

isign — int. Flag indicating the type of operation to be performed:
if *isign* = 0, initialize the coefficients *wsave*;
if *isign* = -1, perform the forward FFT where input and output are in normal order;
if *isign* = +1, perform the inverse FFT where input and output are in normal order;
if *isign* = -2, perform the forward FFT where input is in normal order and output is in bit-reversed order;
if *isign* = +2, perform the inverse FFT where input is in bit-reversed order and output is in normal order.

wsave — float* for cfft1dc
double* for zfft1dc
Pointer to an array of size at least (3*n). If *isign* = 0, then wsave is an output parameter. Otherwise, wsave contains the FFT coefficients initialized on a previous call with *isign* = 0.

Output Parameters

r — Contains the real part of the transform depending on *isign*. Does not change if *isign* = 0.

i — Contains the imaginary part of the transform depending on *isign*. Does not change if *isign* = 0.

wsave — If *isign* = 0, *wsave* contains the initialized FFT coefficients. Otherwise, *wsave* does not change.

Real-to-Complex One-dimensional FFTs

Each of the real-to-complex routines computes forward FFT of a real input vector according to the mathematical equation

$$z_j = \sum_{k=0}^{n-1} t_k * w^{-j*k}, \quad 0 \le j \le n-1$$

for t_k = cmplx(r_k,0), where r_k is the real input vector, $0 \le k \le n-1$.
The mathematical result z_j, $0 \le j \le n-1$, is the complex conjugate-symmetric vector, where z(n/2+i) = conjg(z(n/2-i)), $1 \le i \le n/2-1$, and moreover z(0) and z(n/2) are real values.

This complex conjugate-symmetric (CCS) vector can be stored in the complex array of size (n/2+1) or in the real array of size (n+2). The data storage of the CCS format is defined later for Fortran-interface and C-interface routines separately.

Table 2-2 shows a comparison of the effects of performing the cfft1d/zfft1d complex-to-complex FFT on a vector of length n=8 in which all the imaginary elements are zeros, with the real-to-complex scfft1d/zdfft1d FFT applied to the same vector. The advantage of the latter approach is that only half of the input data storage is required and there is no need to zero the imaginary part. The last two columns are stored in the real array of size (n+2) containing the complex conjugate-symmetric vector in CCS format.

To compute a forward FFT of a given length, first initialize the coefficients by calling the routine you are going to use with *isign* = 0. Thereafter, any number of real-to-complex and complex-to-real transforms of the same length can be computed by calling that routine with the *isign* value other than 0.

Table 2-2 Comparison of the Storage Effects of Complex-to-Complex and Real-to-Complex FFTs

Input Vectors			Output Vectors			
cfft1d		scfft1d	cfft1d		scfft1d	
Complex Data		Real Data	Complex Data		Real Data	
Real	Imaginary		Real	Imaginary	Real	Imaginary
0.841471	0.000000	0.841471	1.543091	0.000000	1.543091	0.000000
0.909297	0.000000	0.909297	3.875664	0.910042	3.875664	0.910042
0.141120	0.000000	0.141120	-0.915560	-0.397326	-0.915560	-0.397326
-0.756802	0.000000	-0.756802	-0.274874	-0.121691	-0.274874	-0.121691
-0.958924	0.000000	-0.958924	-0.181784	0.000000	-0.181784	0.000000
-0.279415	0.000000	-0.279415	-0.274874	0.121691		
0.656987	0.000000	0.656987	-0.915560	0.397326		

Table 2-2 Comparison of the Storage Effects of Complex-to-Complex and Real-to-Complex FFTs

Input Vectors			Output Vectors			
cfft1d		scfft1d	cfft1d		scfft1d	
Complex Data		Real Data	Complex Data		Real Data	
Real	Imaginary		Real	Imaginary	Real	Imaginary
0.989358	0.000000	0.989358	3.875664	-0.910042		

scfft1d/dzfft1d

Fortran-interface routines. Compute forward FFT of a real vector and represent the complex conjugate-symmetric result in CCS format (in-place).

Syntax

```
call scfft1d( r, n, isign, wsave )
call dzfft1d( r, n, isign, wsave )
```

Description

The operation performed by the scfft1d/dzfft1d routines is determined by the value of *isign*. See the equations of the operations for Real-to-Complex One-dimensional FFTs above. These routines are complementary to the complex-to-real transform routines csfft1d/zdfft1d.

Input Parameters

r — REAL for scfft1d
DOUBLE PRECISION for dzfft1d

Array, DIMENSION at least (*n*+2). First *n* elements contain the input vector to be transformed. The elements *r*(*n*+1) and *r*(*n*+2) are used on output. The array *r* is not referenced if *isign* = 0.

n — INTEGER. Transform length; *n* must be a power of 2.

isign — INTEGER. Flag indicating the type of operation to be performed:
if *isign* is 0, initialize the coefficients *wsave*;
if *isign* is not 0, perform the forward FFT.

wsave REAL for scfft1d
DOUBLE PRECISION for dzfft1d

Array, DIMENSION at least (2*n+4). If *isign* = 0, then *wsave* contains output data. Otherwise, *wsave* contains coefficients required to perform the FFT that has been initialized on a previous call to this routine or the complementary complex-to-real FFT routine.

Output Parameters

r If *isign* = 0, *r* does not change. If *isign* is not 0, the output real-valued array *r*(1:*n*+2) contains the complex conjugate-symmetric vector *z*(1:*n*) packed in CCS format for Fortran interface.
The table below shows the relationship between them.

r(1)	*r*(2)	*r*(3)	*r*(4)	.. .	*r*(*n*-1)	*r*(*n*)	*r*(*n*+1)	*r*(*n*+2)
z(1)	0	RE*z*(2)	IM*z*(2)	.. .	RE*z*(*n*/2)	IM*z*(*n*/2)	*z*(*n*/2+1)	0

The full complex vector *z*(1:*n*) is defined by

z(i) = cmplx(*r*(2*i-1), *r*(2*i)), 1 ≤ i ≤ *n*/2+1,

z(*n*/2+i) = conjg(*z*(*n*/2+2-i)), 2 ≤ i ≤ n/2.

Then, *z*(1:*n*) is the forward FFT of a real input vector *r*(1:*n*).

wsave If *isign* = 0, *wsave* contains the coefficients required by the called routine. Otherwise *wsave* does not change.

scfft1dc/dzfft1dc

C-interface routines. Compute forward FFT of a real vector and represent the complex conjugate-symmetric result in CCS format (in-place).

Syntax

```
void scfft1dc( float* r, int n, int isign, float* wsave );
void dzfft1dc( double* r, int n, int isign, double* wsave );
```

Description

The operation performed by the scfft1dc/dzfft1dc routines is determined by the value of *isign*. See the equations of the operations for the Real-to-Complex One-dimensional FFTs above.
These routines are complementary to the complex-to-real transform routines csfft1dc/zdfft1dc.

Input Parameters

r float* for scfft1dc
double* for dzfft1dc

Pointer to an array of size at least (*n*+2). First *n* elements contain the input vector to be transformed. The array *r* is not referenced if *isign* = 0.

n int. Transform length; *n* must be a power of 2.

isign int. Flag indicating the type of operation to be performed:

if *isign* is 0, initialize the coefficients *wsave*;

if *isign* is not 0, perform the forward FFT.

wsave float* for scfft1dc
double* for dzfft1dc

Pointer to an array of size at least (2**n*+4).
If *isign* = 0, then *wsave* contains output data. Otherwise, *wsave* contains coefficients required to perform the FFT that has been initialized on a previous call to this routine or the complementary complex-to-real FFT routine.

Output Parameters

r If *isign* = 0, *r* does not change. If *isign* is not 0, the output real-valued array *r*(0:*n*+1) contains the complex conjugate-symmetric vector *z*(0:*n*-1) packed in CCS format for C interface.
The table below shows the relationship between them.

r(0)	*r*(1)	*r*(2)	.. .	*r*(*n*/2)	*r*(*n*/2+1)	*r*(*n*/2+2)	.. .	*r*(*n*)	*r*(*n*+1)
z(0)	RE*z*(1)	RE*z*(2)	.. .	*z*(*n*/2)	0	IM*z*(1)	.. .	IM*z*(*n*/2-1)	0

The full complex vector *z*(0:*n*-1) is defined by

z(i) = cmplx(*r*(i),*r*(n/2+1+i)), 0 ≤ i ≤ *n*/2,

z(*n*/2+i) = conjg(*z*(*n*/2-i)), 1 ≤ i ≤ *n*/2-1.
Then, *z*(0:*n*-1) is the forward FFT of the real input vector of length *n*.

wsave If *isign* = 0, *wsave* contains the coefficients required by the called routine. Otherwise wsave does not change.

Complex-to-Real One-dimensional FFTs

Each of the complex-to-real routines computes a one-dimensional inverse FFT according to the mathematical equation

$$t_j = \frac{1}{n}\sum_{k=0}^{n-1} z_k * w^{j*k}, \quad 0 \le j \le n-1$$

The mathematical input is the complex conjugate-symmetric vector z_j, $0 \le j \le n-1$, where $z(n/2+i) = \text{conjg}(z(n/2-i))$, $1 \le i \le n/2-1$, and moreover $z(0)$ and $z(n/2)$ are real values.

The mathematical result is $t_j = \text{cmplx}(r_j, 0)$, where r_j is a real vector, $0 \le j \le n-1$.

Input to the complex-to-real transform routines is a real array of size $(n+2)$, which contains the complex conjugate-symmetric vector $z(0:n-1)$ in CCS format (see Real-to-Complex One-dimensional FFTs above).

Output of the complex-to-real routines is a real vector of size n.

Table 2-3 is identical to Table 2-2, except for reversing the input and output vectors. In the complex-to-real routines the last two columns are stored in the input real array of size $(n+2)$ containing the complex conjugate-symmetric vector in CCS format.

To compute an inverse FFT of a given length, first initialize the coefficients by calling the routine you are going to use with *isign* = 0. Thereafter, any number of real-to-complex and complex-to-real transforms of the same length can be computed by calling the appropriate routine with the *isign* value other than 0.

Table 2-3 Comparison of the Storage Effects of Complex-to-Real and Complex-to-Complex FFTs

Output Vectors			Input Vectors			
cfft1d		csfft1d	cfft1d		csfft1d	
Complex Data		Real Data	Complex Data		Real Data	
Real	Imaginary		Real	Imaginary	Real	Imaginary
0.841471	0.000000	0.841471	1.543091	0.000000	1.543091	0.000000
0.909297	0.000000	0.909297	3.875664	0.910042	3.875664	0.910042
0.141120	0.000000	0.141120	-0.915560	-0.397326	-0.915560	-0.397326
-0.756802	0.000000	-0.756802	-0.274874	-0.121691	-0.274874	-0.121691

Table 2-3 Comparison of the Storage Effects of Complex-to-Real and Complex-to-Complex FFTs

Output Vectors			Input Vectors			
cfft1d		csfft1d	cfft1d		csfft1d	
Complex Data		Real Data	Complex Data		Real Data	
Real	Imaginary		Real	Imaginary	Real	Imaginary
-0.958924	0.000000	-0.958924	-0.181784	0.000000	-0.181784	0.000000
-0.279415	0.000000	-0.279415	-0.274874	0.121691		
0.656987	0.000000	0.656987	-0.915560	0.397326		
0.989358	0.000000	0.989358	3.875664	-0.910042		

csfft1d/zdfft1d

Fortran-interface routines. Compute inverse FFT of a complex conjugate-symmetric vector packed in CCS format (in-place).

Syntax

```
call csfft1d( r, n, isign, wsave )
call zdfft1d( r, n, isign, wsave )
```

Description

The operation performed by the `csfft1d/zdfft1d` routines is determined by the value of *isign*. See the equations of the operations for the Complex-to-Real One-dimensional FFTs above.

These routines are complementary to the real-to-complex transform routines `scfft1d/dzfft1d`.

Input Parameters

r `REAL` for `csfft1d`
`DOUBLE PRECISION` for `zdfft1d`

Array, `DIMENSION` at least `(`*n*`+2)`.
Not referenced if *isign* = 0.

If *isign* is not 0, then *r*(1:*n*+2) contains the complex conjugate-symmetric vector packed in CCS format for Fortran interface.
The table below shows the relationship between them.

r(1)	*r*(2)	*r*(3)	*r*(4)	.. .	*r*(*n*-1)	*r*(*n*)	*r*(*n*+1)	*r*(*n*+2)
z(1)	0	RE*z*(2)	IM*z*(2)	.. .	RE*z*(*n*/2)	IM*z*(*n*/2)	z(*n*/2+1)	0

The full complex vector *z*(1:*n*) is defined by

z(i) = cmplx(*r*(2*i-1), *r*(2*i)), 1 ≤ i ≤ *n*/2+1,

z(*n*/2+i) = conjg(*z*(*n*/2+2-i)), 2 ≤ i ≤ *n*/2.

After the transform, *r*(1:*n*) contains the inverse FFT of the complex conjugate-symmetric vector *z*(1:*n*).

n INTEGER. Transform length; *n* must be a power of 2.

isign INTEGER. Flag indicating the type of operation to be performed:
if *isign* is 0, initialize the coefficients wsave;
if *isign* is not 0, perform the inverse FFT.

wsave REAL for csfft1d
DOUBLE PRECISION for zdfft1d
Array, DIMENSION at least (2**n*+4). If *isign* = 0, then *wsave* contains output data. Otherwise, *wsave* contains coefficients required to perform the FFT that has been initialized on a previous call to this routine or the complementary real-to-complex FFT routine.

Output Parameters

r If *isign* is not 0, then *r*(1:*n*) is the real result of the inverse FFT of the complex conjugate-symmetric vector *z*(1:*n*). Does not change if *isign* = 0.

wsave If *isign* = 0, wsave contains the coefficients required by the called routine. Otherwise wsave does not change.

csfft1dc/zdfft1dc

C-interface routines.Compute inverse FFT of a complex conjugate-symmetric vector packed in CCS format (in-place).

Syntax

```
void csfft1dc( float* r, int n, int isign, float* wsave )
void zdfft1dc( double* r, int n, int isign, double* wsave )
```

Description

The operation performed by the `csfft1dc/zdfft1dc` routines is determined by the value of *isign*. See the equations of the operations for the Complex-to-Real One-dimensional FFTs above.

These routines are complementary to the real-to-complex transform routines scfft1dc/dzfft1dc.

Input Parameters

r `float*` for `csfft1dc`
`double*` for `zdfft1dc`

Pointer to an array of size at least `(n+2)`. Not referenced if *isign* = `0`.

If *isign* is not `0`, then `r(0:n+1)` contains the complex conjugate-symmetric vector packed in CCS format for C interface.
The table below shows the relationship between them.

r(0)	r(1)	r(2)	...	r(n/2)	r(n/2+1)	r(n/2+2)	...	r(n)	r(n+1)
z(0)	RE z(1)	RE z(2)	...	z(n/2)	0	IM z(1)	...	IM z(n/2-1)	0

The full complex vector `z(0:n-1)` is defined by
`z(i) = cmplx(r(i),r(n/2+1+i)), 0 ≤ i ≤ n/2,`

`z(n/2+i) = conjg(z(n/2-i)), 1 ≤ i ≤ n/2-1.`
After the transform, `r(0:n-1)` is the inverse FFT of the complex conjugate-symmetric vector `z(0:n-1)`.

n `int`. Transform length; *n* must be a power of 2.

isign `int`. Flag indicating the type of operation to be performed:
if *isign* = 0, initialize the coefficients *wsave*;
if *isign* is not 0, perform the inverse FFT.

wsave `float*` for `csfft1dc`
`double*` for `zdfft1dc`
Pointer to an array of size at least `(2*`*n*`+4)`.
If *isign* = 0, then *wsave* contains output data. Otherwise, *wsave* contains coefficients required to perform the FFT that has been initialized on a previous call to this routine or the complementary real-to-complex FFT routine.

Output Parameters

r If *isign* is not 0, then *r*`(0:`*n*`-1)` is the real result of the inverse FFT of the complex conjugate-symmetric vector *z*`(0:`*n*`-1)`. Does not change if *isign* `= 0`.

wsave If *isign* `= 0`, wsave contains the coefficients required by the called routine. Otherwise wsave does not change.

Two-dimensional FFTs

The two-dimensional FFTs are functionally the same as one-dimensional FFTs. They contain the following groups:

- Complex-to-Complex Transforms
- Real-to-Complex Transforms
- Complex-to-Real Transforms.

All two-dimensional FFTs are in-place. Transform lengths must be a power of 2. The complex-to-complex transform routines perform both forward and inverse transforms of a complex matrix. The real-to-complex transform routines perform forward transforms of a real matrix. The complex-to-real transform routines perform inverse transforms of a complex conjugate-symmetric matrix, which is packed in a real array.

The naming conventions are also the same as those for one-dimensional FFTs, with “2d” replacing “1d” in all cases. Table 2-4 lists the two-dimensional FFT routine groups and the data types associated with them.

Table 2-4 Two-dimensional FFTs: Names and Data Types

Group	Stored as FORTRAN Complex Data	Stored as C Real Data	Data Types	Description
Complex-to-Complex	`cfft2d/ zfft2d`	`cfft2dc/ zfft2dc`	c, z	Transform complex data to complex data.

Table 2-4 Two-dimensional FFTs: Names and Data Types

Group	Stored as FORTRAN Complex Data	Stored as C Real Data	Data Types	Description
Real-to-Complex	scfft2d/ dzfft2d	scfft2dc/ dzfft2dc	sc, dz	Transform forward real-to-complex data. Complement csfft2d/zdfft2d and csfft2dc/zdfft2dc FFTs.
Complex-to-Real	csfft2d/ zdfft2d	csfft2dc/ zdfft2dc	cs, zd	Transform inverse complex-to-real data. Complement scfft2d/dzfft2d and scfft2dc/dzfft2dc FFTs.

The C interface requires scalar values to be passed by value. The major difference between the one-dimensional and two-dimensional FFTs is that your application does not need to provide storage for transform coefficients.

The data storage types and data structure requirements are the same as for one-dimensional FFTs. For more information, see the Data Storage Types and Data Structure Requirements sections at the beginning of this chapter.

Complex-to-Complex Two-dimensional FFTs

Each of the complex-to-complex routines computes a forward or inverse FFT of a complex matrix in-place.

The forward FFT is computed according to the mathematical equation

$$z_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} r_{k,l} * w_m^{-i*k} * w_n^{-j*l}, \quad 0 \le i \le m-1, \quad 0 \le j \le n-1$$

The inverse FFT is computed according to the mathematical equation

$$r_{i,j} = \frac{1}{m*n} \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} z_{k,l} * w_m^{i*k} * w_n^{j*l}, \quad 0 \le i \le m-1, \; 0 \le j \le n-1$$

where $w_m = \exp\left[\frac{2\pi i}{m}\right]$, $w_n = \exp\left[\frac{2\pi i}{n}\right]$, *i* being the imaginary unit.

The operation performed by the complex-to-complex routines is determined by the value of the *isign* parameter.

If *isign* = -1, perform the forward FFT where input and output are in normal order.
If *isign* = +1, perform the inverse FFT where input and output are in normal order.
If *isign* = -2, perform the forward FFT where input is in normal order and output is in bit-reversed order.
If *isign* = +2, perform the inverse FFT where input is in bit-reversed order and output is in normal order.

The above equations apply to all FFTs with all data types indicated in Table 2-4.

cfft2d/zfft2d

Fortran-interface routines. Compute the forward or inverse FFT of a complex matrix (in-place).

Syntax

```
call cfft2d( r, m, n, isign )
call zfft2d( r, m, n, isign )
```

Description

The operation performed by the cfft2d/zfft2d routines is determined by the value of *isign*. See the equations of the operations for Complex-to-Complex Two-dimensional FFTs.

Input Parameters

r — COMPLEX for cfft2d
DOUBLE COMPLEX for zfft2d
Array, DIMENSION at least (*m*,*n*), with its leading dimension equal to *m*. This array contains the complex matrix to be transformed.

m — INTEGER. Column transform length (number of rows);
m must be a power of 2.

n — INTEGER. Row transform length (number of columns); *n* must be a power of 2.

isign — INTEGER. Flag indicating the type of operation to be performed:
if *isign* = -1, perform the forward FFT where input and output are in normal order;
if *isign* = +1, perform the inverse FFT where input and output are in normal order;
if *isign* = -2, perform the forward FFT where input is in normal order and

output is in bit-reversed order;
if *isign* = +2, perform the inverse FFT where input is in bit-reversed order and output is in normal order.

Output Parameters

r Contains the complex result of the transform depending on *isign*.

cfft2dc/zfft2dc

C-interface routines. Compute the forward or inverse FFT of a complex matrix (in-place).

Syntax

```
void cfft2dc( float* r, float* i, int m, int n, int isign )
void zfft2dc( double* r, double* i, int m, int n, int isign )
```

Description

The operation performed by the `cfft2dc/zfft2dc` routines is determined by the value of *isign*. See the equations of the operations for the Complex-to-Complex Two-dimensional FFTs above.

Input Parameters

r `float*` for `cfft2dc`
`double*` for `zfft2dc`

Pointer to a two-dimensional array of size at least (*m*,*n*), with its leading dimension equal to *n*. The array contains the real parts of a complex matrix to be transformed.

i `float*` for `cfft2dc`
`double*` for `zfft2dc`

Pointer to a two-dimensional array of size at least (*m*,*n*), with its leading dimension equal to *n*. The array contains the imaginary parts of a complex matrix to be transformed.

m `int`. Column transform length (number of rows); *m* must be a power of 2.

n `int`. Row transform length (number of columns); *n* must be a power of 2.

isign `int`. Flag indicating the type of operation to be performed:

if *isign* = -1, perform the forward FFT where input and output are in normal order;
if *isign* = +1, perform the inverse FFT where input and output are in normal order;
if *isign* = -2, perform the forward FFT where input is in normal order and output is in bit-reversed order;
if *isign* = +2, perform the inverse FFT where input is in bit-reversed order and output is in normal order.

Output Parameters

r	Contains the real parts of the complex result depending on *isign*.
i	Contains the imaginary parts of the complex depending on *isign*.

Real-to-Complex Two-dimensional FFTs

Each of the real-to-complex routines computes the forward FFT of a real matrix according to the mathematical equation

$$z_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} t_{k,l} * w_m^{-i*k} * w_n^{-j*l}, \quad 0 \le i \le m-1, \quad 0 \le j \le n-1$$

$t_{k,l}$ = cmplx($r_{k,l}$,0), where $r_{k,l}$ is a real input matrix, $0 \le k \le m-1$, $0 \le l \le n-1$.
The mathematical result $z_{i,j}$, $0 \le i \le m-1$, $0 \le j \le n-1$, is the complex matrix of size (m,n). Each column is the complex conjugate-symmetric vector as follows:

for $0 \le j \le n-1$,

$z(m/2+i,j)$ = conjg($z(m/2-i,j)$), $1 \le i \le m/2-1$.
Moreover, $z(0,j)$ and $z(m/2,j)$ are real values for $j=0$ and $j=n/2$.

This mathematical result can be stored in the real two-dimensional array of size $(m+2,n+2)$ or in the complex two-dimensional array of size $(m/2+1,n+1)$ for Fortran interface and in the complex two-dimensional array of size $(m+1,n/2+1)$ for C interface. The data storage of CCS format is defined later for Fortran-interface and C-interface routines separately.

scfft2d/dzfft2d

Fortran-interface routines. Compute forward FFT of a real matrix and represent the complex conjugate-symmetric result in CCS format (in-place).

Syntax

```
call scfft2d( r, m, n )
call dzfft2d( r, m, n )
```

Description

See the equations of the operations for the Real-to-Complex Two-dimensional FFTs above.

These routines are complementary to the complex-to-real transform routines csfft2d/zdfft2d.

Input Parameters

r	REAL for scfft2d DOUBLE PRECISION for dzfft2d Array, DIMENSION at least (*m*+2,*n*+2), with its leading dimension equal to (*m*+2). The first *m* rows and n columns of this array contain the real matrix to be transformed. Table 2-5 presents the input data layout.
m	INTEGER. Column transform length (number of rows); *m* must be a power of 2.
n	INTEGER. Row transform length (number of columns); *n* must be a power of 2.

Table 2-5 Fortran-interface Real Data Storage for the Real-to-Complex and Complex-to-Real Two-dimensional FFTs

r(1,1)	r(1,2)	. . .	r(1,n-1)	r(1,n)	n/u	n/u
r(2,1)	r(2,2)	. . .	r(2,n-1)	r(2,n)	n/u	n/u
r(3,1)	r(3,2)	. . .	r(3,n-1)	r(3,n)	n/u	n/u
r(4,1)	r(4,2)	. . .	r(4,n-1)	r(4,n)	n/u	n/u
. . .	. . .	. . .	. . .	. . .	...	...
r(m-1,1)	r(m-1,2)	. . .	r(m-1,n-1)	r(m-1,n)	n/u	n/u
r(m,1)	r(m,2)	. . .	r(m,n-1)	r(m,n)	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u

* n/u - not used

Output Parameters

r The output real array r(1:m+2,1:n+2) contains the complex conjugate-symmetric matrix z(1:m,1:n) packed in CCS format for Fortran interface as follows:

- Rows 1 and m+1 contain in n+2 locations the complex conjugate-symmetric vectors z(1,j) and z(m/2+1,j) packed in CCS format (see Real-to-Complex One-dimensional FFTs above).

  The full complex vector z(1,j) is defined by:

  $z(1,j) = \mathrm{cmplx}(r(1,2*j-1), r(1,2*j)),\ 1 \le j \le n/2+1,$

  $z(1,n/2+1+j) = \mathrm{conjg}(z(1,n/2+1-j)),\ 1 \le j \le n/2-1.$

  The full complex vector z(m/2+1,j) is defined by:

  $z(m/2+1,j) = \mathrm{cmplx}(r(m+1,2*j-1), r(m+1,2*j)),\ 1 \le j \le n/2+1,$

  $z(m/2+1,n/2+1+j) = \mathrm{conjg}(z(m/2+1,n/2+1-j)),\ 1 \le j \le n/2-1;$

- Rows from 3 to m contain in n locations complex vectors represented as

  $z(i+1,j) = \mathrm{cmplx}(r(2*i+1,j), r(2*i+2,j)),\ 1 \le i \le m/2-1,\ 1 \le j \le n.$

- The rest matrix elements can be obtained from

  $z(m/2+1+i,j) = \mathrm{conjg}(z(m/2+1-i,j)),\ 1 \le i \le m/2-1,\ 1 \le j \le n.$

The storage of the complex conjugate-symmetric matrix *z* for Fortran interface is shown in Table 2-6.

Table 2-6 Fortran-interface Data Storage of CCS Format for the Real-to-Complex and Complex-to-Real Two-Dimensional FFTs

z(1,1)	0	RE*z*(1,2)	IM*z*(1,2)	...	RE*z*(1,*n*/2)	IM*z*(1,*n*/2)	*z*(1, *n*/2+1)	0
0	0	0	0	...	0	0	0	0
RE*z*(2,1)	RE*z*(2,2)	RE*z*(2,3)	RE*z*(2,4)	...	RE*z*(2,*n*-1)	RE*z*(2,*n*)	n/u	n/u
IM*z*(2,1)	IM*z*(2,2)	IM*z*(2,3)	IM*z*(2,4)	...	IM*z*(2,*n*-1)	IM*z*(2,*n*)	n/u	n/u
...	...	...	...	...	...	...	n/u	n/u
RE*z*(*m*/2,1)	RE*z*(*m*/2,2)	RE*z*(*m*/2,3)	RE*z*(*m*/2,4)	...	RE*z*(*m*/2, *n*-1)	RE*z*(*m*/2, *n*)	n/u	n/u
IM*z*(*m*/2,1)	IM*z*(*m*/2,2)	IM*z*(*m*/2,3)	IM*z*(*m*/2,4)	...	IM*z*(*m*/2, *n*-1)	IM*z*(*m*/2, *n*)	n/u	n/u
z(*m*/2+1,1)	0	RE*z*(*m*/2+1,2)	IM*z*(*m*/2+1,2)	...	RE*z*(*m*/2+1, *n*/2)	IM*z*(*m*/2+1, *n*/2)	*z*(*m*/2+1, *n*/2+1)	0
0	0	0	0	...	0	0	n/u	n/u

* n/u - not used

scfft2dc/dzfft2dc

C-interface routine. Compute forward FFT of a real matrix and represent the complex conjugate-symmetric result in CCS format (in-place).

Syntax

```
void scfft2dc( float* r, int m, int n )
void dzfft2dc( double* r, int m, int n )
```

Description

See the equations of the operations for the Real-to-Complex Two-dimensional FFTs above.

These routines are complementary to the complex-to-real transform routines csfft2dc/zdfft2dc.

Input Parameters

r — `float*` for `scfft2dc`
`double*` for `dzfft2dc`

Pointer to an array of size at least `(`*m*`+2,`*n*`+2)`, with its leading dimension equal to `(`*n*`+2)`. The first *m* rows and *n* columns of this array contain the real matrix to be transformed.

Table 2-7 presents the input data layout.

m — `int`. Column transform length;
m must be a power of 2.

n — `int`. Row transform length;
n must be a power of 2.

Table 2-7 C-interface Real Data Storage for a Real-to-Complex and Complex-to-Real Two-dimensional FFTs

r(0,0)	r(0,1)	...	r(0,n-2)	r(0,n-1)	n/u	n/u
r(1,0)	r(1,1)	...	r(1,n-2)	r(1,n-1)	n/u	n/u
r(2,0)	r(2,1)	...	r(2,n-2)	r(2,n-1)	n/u	n/u
r(3,0)	r(3,1)	...	r(3,n-2)	r(3,n-1)	n/u	n/u
...	...	...	...	...	...	...
r(m-2,0)	r(m-2,1)	...	r(m-2,n-2)	r(m-2,n-1)	n/u	n/u
r(m-1,0)	r(m-1,1)	...	r(m-1,n-2)	r(m-1,n-1)	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u

Output Parameters

r The output real array *r*(0:*m*+1,0:*n*+1) contains the complex conjugate-symmetric matrix *z*(0:*m*-1,0:*n*-1) packed in CCS format for C interface as follows:

- Columns 0 and *n*/2 contain in *m*+2 locations the complex conjugate-symmetric vectors *z*(i,0) and *z*(i,*n*/2) in CCS format (see Real-to-Complex One-dimensional FFTs above).

  The full complex vector *z*(i,0) is defined by:

  z(i,0) = cmplx(*r*(i,0),*r*(*m*/2+i+1,0)), 0 ≤ i ≤ *m*/2,

  z(*m*/2+i,0) = conjg(*z*(*m*/2-i,0)), 1 ≤ i ≤ *m*/2-1.

  The full complex vector z(i,n/2) is defined by:

  z(i,*n*/2) = cmplx(*r*(i,*n*/2),*r*(*m*/2+i+1,*n*/2)), 0 ≤ i ≤ *m*/2,

  z(*m*/2+i,*n*/2) = conjg(*z*(*m*/2-i,*n*/2)), 1 ≤ i ≤ *m*/2-1.
- Columns from 1 to *n*/2-1 contain real parts, and columns from *n*/2+2 to *n* contain imaginary parts of complex vectors. These values for each vector are stored in *m* locations represented as follows

  z(i,j) = cmplx(*r*(i,j),*r*(i,*n*/2+1+j)),

  0 ≤ i ≤ *m*-1, 1 ≤ j ≤ *n*/2-1.
- The rest matrix elements can be obtained from

  z(i,*n*/2+j) = conjg(*z*(i,*n*/2-j)),

  0 ≤ i ≤ *m*-1, 1 ≤ j ≤ *n*/2-1.

Table 2-8 shows the storage of the complex conjugate-symmetric matrix *z* for C interface.

Table 2-8 C-interface Data Storage of CCS Format for the Real-to-Complex and Complex-to-Real Two-dimensional FFT

z(0,0)	REz(0,1)	...	REz(0, n/2-1)	z(0,n/2)	0	IMz(0,1)	...	IMz(0, n/2-1)	0
REz(1,0)	REz(1,1)	...	REz(1, n/2-1)	REz(1,n/2)	0	IMz(1,1)	...	IMz(1, n/2-1)	0
...	...	...	...	...	0	...	...	...	0
REz(m/2-1, 0)	REz(m/2-1, 1)	...	REz(m/2-1, n/2-1)	REz(m/2-1, n/2)	0	IMz(m/2-1, 1)	...	IMz(m/2-1, n/2-1)	0
z(m/2,0)	REz(m/2,1)	...	REz(m/2, n/2-1)	z(m/2,n/2)	0	IMz(m/2,1)	...	IMz(m/2, n/2-1)	0
0	REz(m/2+1, 1)	...	REz(m/2+1, n/2-1)	0	0	IMz(m/2+1, 1)	...	IMz(m/2+1, n/2-1)	0
IMz(1,0)	REz(m/2+2, 1)	...	REz(m/2+2, n/2-1)	IMz(1,n/2)	0	IMz(m/2+2, 1)	...	IMz(m/2+2, n/2-1)	0
...	...	...	...	...	0	...	...	...	0
IMz(m/2-2, 0)	REz(m-1,1)	...	REz(m-1, n/2-1)	IMz(m/2-2, n/2)	0	IMz(m-1,1)	...	IMz(m-1, n/2-1)	0
IMz(m/2-1, 0)	n/u	...	n/u	IMz(m/2-1, n/2)	n/u	n/u	...	n/u	n/u
0	n/u	...	n/u	0	n/u	n/u	...	n/u	n/u

Complex-to-Real Two-dimensional FFTs

Each of the complex-to-real routines computes a two-dimensional inverse FFT according to the mathematical equation:

$$t_{i,j} = \frac{1}{m*n} \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} z_{k,l} * w_m^{i*k} * w_n^{j*l}, \quad 0 \le i \le m-1, \quad 0 \le j \le n-1$$

The mathematical input $z_{i,j}$, $0 \le i \le m-1$, $0 \le j \le n-1$, is a complex matrix of size (m,n). Each column is the complex conjugate-symmetric vector as follows:

for $0 \le j \le n-1$,
$z(m/2+i,j) = \text{conjg}(z(m/2-i,j))$, $1 \le i \le m/2-1$.
Moreover, $z(0,j)$ and $z(m/2,j)$ are real values for $j=0$ and $j=n/2$.

This mathematical result can be stored in the real two-dimensional array of size (*m*+2,*n*+2) or in the complex two-dimensional array of size (*m*/2+1,*n*+1) for Fortran interface and in the complex two-dimensional array of size (*m*+1,*n*/2+1) for C interface. The data storage of CCS format is defined later for Fortran-interface and C-interface routines separately.

The mathematical result of the transform is $t_{k,l} = cmplx(r_{k,l},0)$, where $r_{k,l}$ is the real matrix, $0 \le k \le m-1$, $0 \le l \le n-1$.

csfft2d/zdfft2d

Fortran-interface routine. Compute inverse FFT of a complex conjugate-symmetric matrix packed in CCS format (in-place).

Syntax

```
call csfft2d( r, m, n )
call zdfft2d( r, m, n )
```

Description

See the equations of the operations for the Complex-to-Real Two-dimensional FFTs above. These routines are complementary to the real-to-complex transform routines scfft2d/dzfft2d.

Input Parameters

r SINGLE PRECISION REAL*4 for csfft2d
DOUBLE PRECISION REAL*8 for zdfft2d

Array, DIMENSION at least (*m*+2,*n*+2), with its leading dimension equal to (*m*+2). This array contains the complex conjugate-symmetric matrix in CCS format to be transformed. The input data layout is given in Table 2-6.

m INTEGER. Column transform length (number of rows); *m* must be a power of 2.

n INTEGER. Row transform length (number of columns); *n* must be a power of 2.

Output Parameters

r Contains the real result returned by the transform. For the output data layout, see Table 2-5.

csfft2dc/zdfft2dc

C-interface routines. Compute inverse FFT of a complex conjugate-symmetric matrix packed in CCS format (in-place).

Syntax

```
void csfft2dc( float* r, int m, int n );
void zdfft2dc( double* r, int m, int n );
```

Description

See the equations of the operations for the Complex-to-Real Two-dimensional FFTs above. These routines are complementary to the real-to-complex transform routines scfft2dc/dzfft2dc.

Input Parameters

r — `float*` for `csfft2dc`; `double*` for `zdfft2dc`

Pointer to an array of size at least `(m+2,n+2)`, with its leading dimension equal to `(n+2)`. This array contains the complex conjugate-symmetric matrix in CCS format to be transformed. The input data layout is given in Table 2-8.

m — `int`. Column transform length; *m* must be a power of 2.

n — `int`. Row transform length; *n* must be a power of 2.

Output Parameters

r — Contains the real result returned by the transform. The output data layout is the same as that for the input data of `scfft2dc/dzfft2dc`. See Table 2-7 for the details.

Installation

3

The wrappers are delivered as the source code, which you must compile to build the wrapper library. Then you should use both wrapper and Intel MKL libraries instead of the Intel MKL library alone. The source code for the wrappers and makefiles with the wrapper list files are located in the `\interfaces\fftc` and `\interfaces\fftf` subdirectories in the Intel MKL directory for C and Fortran wrappers, respectively.

Creating a Wrapper Library

Two header files are used to compile the C-interface wrapper library: `mkl_fftcwrappers.h` and `mkl_dfti.h`.

The `mkl_fftcwrappers.h` file is located in the `\interfaces\fftc\wrappers` subdirectory in the Intel MKL directory. The `mkl_dfti.h` is placed in the `\include` subdirectory in the Intel MKL directory.

Two header files are used to compile the Fortran-interface wrapper library: `mkl_fftfwrappers.h` and `mkl_dfti.h`. The `mkl_fftfwrappers.h` file is located in the `\interfaces\fftf\wrappers` subdirectory in the Intel MKL directory. The `mkl_dfti.h` is placed in the `\include` subdirectory in the Intel MKL directory.

As the Fortran wrapper library is built by a C compiler, function names in the wrapper library and Fortran object module may be different. The file `mkl_fftfwrappers.h` in the `\interfaces\fftf\wrappers` subdirectory in the Intel MKL directory defines function names according to names in the Fortran module. If a required name is missing in the file, you can change the latter to add the name.

Makefiles contain the following parameters: *`platform`* (required) and *`compiler`*. Description of these parameters can be found in the makefile comment heading.

Examples

The command

```
make lib64
```

builds a wrapper library for the Intel® Itanium® processor family applications using the Intel® C++ Compiler and Intel® Fortran Compiler version 8.0 or higher (Compliers are chosen by default.).

The command

```
make lib64 F=gnu
```

builds a wrapper library for the Intel® Itanium® processor family applications using GNU C compiler.

As a result of a makefile operation, the wrapper library will be created in the directory with Intel MKL libraries corresponding to the used platform. For example, `\lib\64` or `\ia32\lib`.

In the wrapper library names, the suffix corresponds to the used compiler and the underscore is preceded with letter "f" for Fortran and "c" for C.

For example,

`fftf_intel.lib` (Windows*)	`libfftf_intel.a` (Linux*)
`fftc_intel.lib` (Windows)	`libfftc_intel.a` (Linux)
`fftc_ms.lib` (Windows)	`libfftc_gnu.a` (Linux).

Application Assembling

The native `mkl_fft.h` header file should be used when you build C applications.

Running Examples

There are some examples that demonstrate how to use the wrapper library.

The source code for the examples, makefiles used to run them, and the example list files are located in the `\examples\fftc` and `\examples\fftf` subdirectories in the Intel MKL directory.

Example makefile parameters are the same as wrapper library makefile parameters. Example makefiles normally invoke examples. However, if the appropriate wrapper library is not yet created, the makefile will first build it in the same way as the wrapper library makefile does and then proceed to examples.

If the parameter `function=<example_name>` is defined, then only the specified example will run. Otherwise, all examples from the `\examples\fftc\source` subdirectory for C or `\examples\fftf\source` subdirectory for Fortran will run. The subdirectory `\_results` will be created, and the results will be stored there in the `<example_name>.res` files.