Arm Performance Libraries Reference Manual

Version 18.1.0

Document Number 101004_1801_00

Short Contents

Table of Contents

1 Introduction

The Arm Performance Libraries are a set of numerical routines tuned specifically for Arm processors. The routines, which are available via both FORTRAN and C interfaces, include:

- BLAS - Basic Linear Algebra Subprograms (including XBLAS, the extended precision BLAS);
- LAPACK - A comprehensive package of higher level linear algebra routines;
- FFT - a set of Fast Fourier Transform routines for real and complex data;

The BLAS and LAPACK routines provide a portable and standard set of interfaces for common numerical linear algebra operations that allow code containing calls to these routines to be readily ported across platforms. Full documentation for the BLAS and LAPACK are available online. This manual will, therefore, be restricted to providing brief descriptions of the BLAS and LAPACK and providing links to their documentation and other materials (see Chapter 3 [The BLAS], page 6 and see Chapter 4 [LAPACK], page 26).

The FFT is an implementation of the Discrete Fourier Transform (DFT) that makes use of symmetries in the definition to reduce the number of operations required from O(n*n) to O(n*log n) when the sequence length, n, is the product of small prime factors; in particular, when n is a power of 2. Despite the popularity and widespread use of FFT algorithms, the definition of the DFT is not sufficiently precise to prescribe either the forward and backward directions (these are sometimes interchanged), or the scaling factor associated with the forward and backward transforms (the combined forward and backward transforms may only reproduce the original sequence by following a prescribed scaling). See Chapter 5 [Fast Fourier Transforms], page 28.

Chapter 2 [General Information], page 2 provides details on:

- how to link a user program to Arm Performance Libraries;
- FORTRAN and C interfaces to Arm Performance Libraries routines;
- how to obtain the Arm Performance Libraries version and build information;
- how to access the Arm Performance Libraries documentation.

2 General Information

2.1 Accessing the Library

Arm Performance Libraries is compatible with two compilers: GNU GCC (gcc/gfortran) and the Arm C/C++/Fortran Compiler (armclang/armflang) that is packaged with the Arm Allinea Studio. Separate binary distributions of the library are supplied for these two compilers. We recommend using the environment modules provided to manage accessing the correct library for your choice of compiler. This will set the the `ARMPL_DIR` environment variable appropriately. (If you are not using the environment modules then assume that `${ARMPL_DIR}` is set to `/opt/arm/armpl-18.1.0_CPU_DISTRO`.)

The commands:

```
gfortran driver.f90 -L${ARMPL_DIR}/lib -larmpl_lp64
armflang driver.f90 -L${ARMPL_DIR}/lib -larmpl_lp64
```

can be used to compile the program driver.f90 using `gfortran` and `armflang`, respectively, and link it to Arm Performance Libraries.

The Arm Performance Libraries are supplied in both static and shareable versions, `libarmpl_lp64.a` and `libarmpl_lp64.so`, respectively. By default, the commands given above will link to the shareable version of the library, `libarmpl_lp64.so`, if that exists in the directory specified. Linking with the static library can be forced either by using the compiler flag `-static`, e.g.

```
gfortran driver.f90 -L${ARMPL_DIR}/lib -static -larmpl_lp64 -lrt
armflang driver.f90 -L${ARMPL_DIR}/lib -static -larmpl_lp64 -lrt
```

or by inserting the name of the static library explicitly in the command line, e.g.

```
gfortran driver.f90 ${ARMPL_DIR}/lib/libarmpl_lp64.a -lrt
armflang driver.f90 ${ARMPL_DIR}/lib/libarmpl_lp64.a -lrt
```

Note that the `LD_LIBRARY_PATH` is automatically updated if you are using the supplied environment modules; if not then you should set it yourself when linking to the shareable library.

If you wish to link against a multi-threaded version of Arm Performance Libraries, do it like this:

```
gfortran -fopenmp driver.f90 -L${ARMPL_DIR}/lib -larmpl_lp64_mp
armflang -fopenmp driver.f90 -L${ARMPL_DIR}/lib -larmpl_lp64_mp
```

Note that the library names involved now include the suffix `_mp`.

To compile and link a C program with Arm Performance Libraries, invoke as appropriate:

```
gcc -I${ARMPL_DIR}/include driver.c \
    -L${ARMPL_DIR}/lib -larmpl_lp64 -lgfortran
armclang -I${ARMPL_DIR}/include driver.c \
    -L${ARMPL_DIR}/lib -larmpl_lp64 -lflang -lflangrti
```

The switch `"-I${ARMPL_DIR}/include"` tells the compiler to search the specified directory for the Arm Performance Libraries C header file `armpl.h`, which should be included by `driver.c`. Note that it is necessary to add the Fortran compiler run-time library/libraries (`-lgfortran` or `-lflang -lflangrti`) when linking the program.

2.2 Arm Performance Libraries FORTRAN and C interfaces

Most routines in Arm Performance Libraries come with both FORTRAN and C interfaces. The FORTRAN interfaces typically follow the relevant standard (e.g. LAPACK, BLAS).

In C code that uses Arm Performance Libraries routines, be sure to include the header file `armpl.h`, which contains function prototypes for all Arm Performance Libraries C interfaces. The header file also contains C prototypes for FORTRAN interfaces, thus the C programmer could call the FORTRAN interfaces from C, though there is little reason to do so.

For more details on the C interfaces see Chapter 3 [BLAS], page 6, Chapter 4 [LAPACK], page 26 and Chapter 5 [FFTs], page 28.

Arm Performance Libraries also includes Fortran interface block modules for user-callable routines. These modules define the type and arguments of each Arm Performance Libraries routine. Their purpose is to allow the Fortran compiler to check that Arm Performance Libraries routines are called correctly. The interface blocks enable the compiler to check that:

- subroutines are called as such
- functions are declared with the right type
- the correct number of arguments are passed
- all arguments match in type and structure

The interface block moduless are not essential to calling Arm Performance Libraries from Fortran programs, but the supplied example programs use them, and you are advised to use them yourself in order to avoid common mistakes.

The interfaces to Arm Performance Libraries routines are declared in files `armpl_blas.f90`, `armpl_lapack.f90` and `armpl_fft.f90`, and they are also are aggregated into one module named `armpl_library`. These modules can be found in the Arm Performance Libraries include directory. See the Fortran programs in the Arm Performance Libraries examples directory for guidance on use.

2.3 Arm Performance Libraries variant using 64-bit integer (INTEGER*8) arguments

If you have a Fortran program using `INTEGER*8` variables, or a C program using 8-byte long variables, there is an Arm Performance Libraries version that you can use.

The `INTEGER*8` versions of the libraries are distinguished from the 32-bit integer versions by having the string "`_ilp64`" as part of the name of the library rather than "`_lp64`". Additional versions of the module files compiled to use `INTEGER*8` are found in the `include_ilp64` directory.

For these Arm Performance Libraries variants, all documentation that mentions arguments of Fortran type `INTEGER` or C type `int` should be read as `INTEGER*8` or `long` respectively.

```
gfortran driver.f90 -L${ARMPL_DIR}/lib \
         -larmpl_ilp64 -I${ARMPL_DIR}/include_ilp64
armflang driver.f90 -L${ARMPL_DIR}/lib \
         -larmpl_ilp64 -I${ARMPL_DIR}/include_ilp64
```

It is important to ensure that if you have INTEGER*8 variables in your code, you link to the ilp64 variant, and not otherwise. Unexpected program crashes are likely to occur if you link to the wrong version.

2.4 Library Version and Build Information

This document is applicable to version 18.1.0 of Arm Performance Libraries. The utility routine `armplversion()` can be called to obtain the major, minor and patch version numbers of the installed Arm Performance Libraries.

The utility routine `armplinfo()` can be called to obtain information on the compiler used to build Arm Performance Libraries, the version of the compiler, and the options used for building the Library. This subroutine takes no arguments and prints the information to the current standard output.

FORTRAN specifications:

`ARMPLVERSION` (*`MAJOR`*, *`MINOR`*, *`PATCH`*, *`BUILD`*) [SUBROUTINE]

`MAJOR`, *`MINOR`*, *`PATCH`*, *`BUILD`* [INTEGER]

`ARMPLINFO` () [SUBROUTINE]

C specifications:

`void armplversion` (*int* `*major`, *int* `*minor`, *int* `*patch`, *int* `*build`); [function]

`void armplinfo` (*void*); [function]

2.5 Library Documentation

The /Doc subdirectory of the top Arm Performance Libraries installation directory, e.g. `${ARMPL_DIR}/Doc`, should contain this document in the following formats:

- Printed Manual / PDF format – armpl.pdf
- Info Pages – armpl.info
- Plain text – armpl.txt

The info file can be read using *info* after updating the environment variable `INFOPATH` to include the doc subdirectory of the Arm Performance Libraries installation directory, e.g.

```
% setenv INFOPATH ${INFOPATH}:${ARMPL_DIR}/Doc

% info armpl
```

or simply by using the full name of the file:

```
% info ${ARMPL_DIR}/Doc/armpl.info
```

2.6 Example programs calling Arm Performance Libraries

The `examples` subdirectory of the top Arm Performance Libraries installation directory contains example programs showing how to call the Arm Performance Libraries, along with a `GNUmakefile` to build and run them. Examples of calling both FORTRAN and C interfaces are included. They may be used as a Arm Performance Libraries installation test.

Depending on where your copy of the Arm Performance Libraries is installed, and which compiler and flags you wish to use, it may be necessary to modify some variables in the GNUmakefile before using it.

If you need more example programs showing how to call LAPACK routines from Fortran, we refer you to this web page:

`http://www.nag.com/lapack/`

Here you will find examples for all double precision LAPACK driver routines, and all of these should work when linked with Arm Performance Libraries. Note that as well as the example programs themselves, it is necessary to download and compile a small amount of utility code used by the programs. See the web page for detailed instructions.

3 BLAS: Basic Linear Algebra Subprograms

The BLAS are a set of well defined basic linear algebra operations ([1], [2], [3]). These operations are subdivided into three groups:

- Level 1: operations acting on vectors only (e.g. dot product)
- Level 2: matrix-vector operations (e.g. matrix-vector multiplication)
- Level 3: matrix-matrix operations (e.g. matrix-matrix multiplication)

Efficient machine-specific implementations of the BLAS are available for many modern high-performance computers. The implementation of higher level linear algebra algorithms on these systems depends critically on the use of the BLAS as building blocks.

For any information relating to the standard BLAS Fortran interfaces please refer to the BLAS FAQ:

`http://www.netlib.org/blas/faq.html`

For C users, Arm Performance Libraries includes CBLAS, which are C interfaces to the Fortran BLAS. Names of the CBLAS routines are identical to the Fortran routines except they are prefixed by the string `cblas_`. For example, the Fortran BLAS routine `dgemm` becomes `cblas_dgemm` in C. Other differences between Fortran BLAS and CBLAS interfaces:

- Scalar input arguments are passed by value in CBLAS interfaces. FORTRAN interfaces pass all arguments (except for character string *length* arguments that are normally hidden from FORTRAN programmers) by reference.
- Unlike FORTRAN, C has no native *complex* data type. Arm Performance Libraries C routines which operate on complex data use the types `armpl_singlecomplex_t` and `armpl_doublecomplex_t` defined in `armpl.h` for single and double precision computations respectively.
- As with Fortran, all CBLAS array arguments are required to be stored in contiguous memory. However, all Fortran BLAS routines which take one or more matrices (2D arrays) as arguments have an extra parameter in the CBLAS interface, which appears before all other parameters. This parameter is of the enum type `CBLAS_ORDER`, and it can take one of the values `CblasRowMajor` or `CblasColMajor`.

  The value `CblasColMajor` indicates that elements within any column of each array are contiguous in memory while elements within array rows are offset by a constant stride. This is the usual Fortran storage order. Such strides are given by "leading dimension" arguments (such as `LDA`) in the Fortran and C interfaces.

  The value `CblasRowMajor` indicates that elements within any row of each 2D array are contiguous in memory while elements within array columns are offset by a constant stride such as `LDA`.
- Fortran's `character` type arguments are converted into C enumerated types, as shown in the table below:

Fortran Interface		*CBLAS Interface*	
Character Argument	*Value*	*enum type argument*	*Value*
SIDE	'L'	CBLAS_SIDE	CblasLeft
	'R'		CblasRight
UPLO	'L'	CBLAS_UPLO	CblasLower
	'U'		CblasUpper
DIAG	'N'	CBLAS_DIAG	CblasNonUnit
	'U'		CblasUnit
TRANSPOSE	'N'	CBLAS_TRANSPOSE	CblasNoTrans
	'T'		CblasTrans
	'C'		CblasConjTrans

See the `armpl.h` header file to find prototypes for all CBLAS functions. Note that the functions may also be called from a C++ program if you include `armpl.h`.

3.1 BLAS Extensions

In addition to the standard BLAS interfaces, Arm Performance Libraries also supports some commonly used extensions. We document here the Fortran interfaces, but the equivalent CBLAS-like interfaces are supported and prototypes are provided in `armpl.h`.

`SAXPBY` Routine Documentation

SAXPBY performs the following vector update: $y_i \leftarrow ax_i + by_i$ for $i = 0, n-1$.

`SAXPBY (`*`N`*`,`*`A`*`,`*`X`*`,`*`INCX`*`,`*`B`*`,`*`Y`*`,`*`INCY`*`)` [SUBROUTINE]

`INTEGER N` [Input]
On input: The length of the vectors $\underline{x}$ and $\underline{y}$.

`REAL A` [Input]
On input: The scalar a.

`REAL X(1+(N-1)*ABS(INCX))` [Input]
On input: Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

`INTEGER INCX` [Input]
On input: The increment used to store successive elements of the vector in X.

`REAL B` [Input]
On input: The scalar b.

`REAL Y(1+(N-1)*ABS(INCY))` [Input/Output]
On input: Contains the elements of the vector $\underline{y}$ with the I^{th} element stored in Y(1+(I-1)*ABS(INCY)).
On output: Contains the updated values of $\underline{y}$.

`INTEGER INCY` [Input]
On input: The increment used to store successive elements of the vector in Y.

DAXPBY Routine Documentation

DAXPBY performs the following vector update: $y_i \leftarrow ax_i + by_i$ for $i = 0, n-1$.

DAXPBY (*N*,*A*,*X*,*INCX*,*B*,*Y*,*INCY*) [SUBROUTINE]

INTEGER N [Input]
On input: The length of the vectors $\underline{x}$ and $\underline{y}$.

DOUBLE PRECISION A [Input]
On input: The scalar a.

DOUBLE PRECISION X(1+(N-1)*ABS(INCX)) [Input]
On input: Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INTEGER INCX [Input]
On input: The increment used to store successive elements of the vector in X.

DOUBLE PRECISION B [Input]
On input: The scalar b.

DOUBLE PRECISION Y(1+(N-1)*ABS(INCY)) [Input/Output]
On input: Contains the elements of the vector $\underline{y}$ with the I^{th} element stored in Y(1+(I-1)*ABS(INCY)).
On output: Contains the updated values of $\underline{y}$.

INTEGER INCY [Input]
On input: The increment used to store successive elements of the vector in Y.

CAXPBY Routine Documentation

CAXPBY performs the following vector update: $y_i \leftarrow ax_i + by_i$ for $i = 0, n-1$.

CAXPBY (*N*,*A*,*X*,*INCX*,*B*,*Y*,*INCY*) [SUBROUTINE]

INTEGER N [Input]
On input: The length of the vectors $\underline{x}$ and $\underline{y}$.

COMPLEX A [Input]
On input: The scalar a.

COMPLEX X(1+(N-1)*ABS(INCX)) [Input]
On input: Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INTEGER INCX [Input]
On input: The increment used to store successive elements of the vector in X.

COMPLEX B [Input]
On input: The scalar b.

COMPLEX Y(1+(N-1)*ABS(INCY)) [Input/Output]
On input: Contains the elements of the vector $\underline{y}$ with the I^{th} element stored in Y(1+(I-1)*ABS(INCY)).
On output: Contains the updated values of $\underline{y}$.

INTEGER INCY [Input]
On input: The increment used to store successive elements of the vector in Y.

ZAXPBY Routine Documentation

ZAXPBY performs the following vector update: $y_i \leftarrow ax_i + by_i$ for $i = 0, n-1$.

ZAXPBY (*N*,*A*,*X*,*INCX*,*B*,*Y*,*INCY*) [SUBROUTINE]

INTEGER N [Input]
On input: The length of the vectors $\underline{x}$ and $\underline{y}$.

COMPLEX*16 A [Input]
On input: The scalar a.

COMPLEX*16 X(1+(N-1)*ABS(INCX)) [Input]
On input: Contains the elements of the vector $\underline{x}$ with the I^{th} element stored in X(1+(I-1)*ABS(INCX)).

INTEGER INCX [Input]
On input: The increment used to store successive elements of the vector in X.

COMPLEX*16 B [Input]
On input: The scalar b.

COMPLEX*16 Y(1+(N-1)*ABS(INCY)) [Input/Output]
On input: Contains the elements of the vector $\underline{y}$ with the I^{th} element stored in Y(1+(I-1)*ABS(INCY)).
On output: Contains the updated values of $\underline{y}$.

INTEGER INCY [Input]
On input: The increment used to store successive elements of the vector in Y.

CGEMM3M Routine Documentation

CGEMM3M performs one of the following matrix-matrix operations:

$C \leftarrow \alpha op(A) op(B) + \beta C$

where $op(X)$ is defined as one of:

$op(X) = X$

$op(X) = X^T$ (matrix transpose)

$op(X) = X^H$ (matrix conjugate transpose)

and α and β are scalars, $op(A)$ is an m by k matrix, $op(B)$ a k by n matrix and C is an m by n matrix.

CGEMM3M (*TRANSA*,*TRANSB*,*M*,*N*,*K*,*ALPHA*,*A*,*LDA*,*B*,*LDB*,*BETA*,*C*,*LDC*) [SUBROUTINE]

CHARACTER*1 TRANSA [Input]

On input:

- If *TRANSA* = 'n' or 'N' then $op(A) = A$;
- If *TRANSA* = 't' or 'T' then $op(A) = A^T$;
- If *TRANSA* = 'c' or 'C' then $op(A) = A^H$.

CHARACTER*1 TRANSB [Input]

On input:

- If *TRANSB* = 'n' or 'N' then $op(B) = B$;
- If *TRANSB* = 't' or 'T' then $op(B) = B^T$;
- If *TRANSB* = 'c' or 'C' then $op(B) = B^H$.

INTEGER M [Input]

On input: The number of rows of the matrices $op(A)$ and C.
Constraint: $M \geq 0$.

INTEGER N [Input]

On input: The number of columns of the matrices $op(B)$ and C.
Constraint: $N \geq 0$.

INTEGER K [Input]

On input: The number of columns of the matrix $op(A)$ and rows of the matrix $op(B)$.
Constraint: $K \geq 0$.

COMPLEX ALPHA [Input]

On input: The value of the scalar α.

COMPLEX A [Input]

On input: An array of dimension (*LDA*, KA) containing the matrix A, where if *TRANSA* = 'n' or 'N' then KA = K and the matrix is held in the leading M by K part of the array, otherwise KA = M and the matrix is held in the leading K by M part of the array.

`INTEGER LDA` [Input]

On input: The leading dimension of the array A as declared in the calling subprogram.

Constraint: If $TRANSA$ = 'n' or 'N' then $LDA \geq max(1, M)$, otherwise $LDA \geq max(1, K)$.

`COMPLEX B` [Input]

On input: An array of dimension (LDB, KB) containing the matrix B, where if $TRANSB$ = 'n' or 'N' then KB = N and the matrix is held in the leading K by N part of the array, otherwise KB = K and the matrix is held in the leading N by K part of the array.

`INTEGER LDB` [Input]

On input: The leading dimension of the array B as declared in the calling subprogram.

Constraint: If $TRANSB$ = 'n' or 'N' then $LDB \geq max(1, K)$, otherwise $LDB \geq max(1, N)$.

`COMPLEX BETA` [Input]

On input: The value of the scalar β.

`COMPLEX C` [Input/Output]

On input: An array of dimension (LDC, N) containing the matrix C, in the leading M by N part of the array. Note that if $BETA = 0$ then the matrix C need not be initialised on input.

On output: The array is overwritten with the result of the operation: $\alpha op(A)op(B) + \beta C$.

`INTEGER LDC` [Input]

On input: The leading dimension of the array C as declared in the calling subprogram.

Constraint: $LDC \geq max(1, M)$.

ZGEMM3M Routine Documentation

ZGEMM3M performs one of the following matrix-matrix operations:

$C \leftarrow \alpha op(A) op(B) + \beta C$

where $op(X)$ is defined as one of:

$op(X) = X$

$op(X) = X^T$ (matrix transpose)

$op(X) = X^H$ (matrix conjugate transpose)

and α and β are scalars, $op(A)$ is an m by k matrix, $op(B)$ a k by n matrix and C is an m by n matrix.

ZGEMM3M (*TRANSA*,*TRANSB*,*M*,*N*,*K*,*ALPHA*,*A*,*LDA*,*B*,*LDB*,*BETA*,*C*,*LDC*) [SUBROUTINE]

CHARACTER*1 TRANSA [Input]

On input:

- If *TRANSA* = 'n' or 'N' then $op(A) = A$;
- If *TRANSA* = 't' or 'T' then $op(A) = A^T$;
- If *TRANSA* = 'c' or 'C' then $op(A) = A^H$.

CHARACTER*1 TRANSB [Input]

On input:

- If *TRANSB* = 'n' or 'N' then $op(B) = B$;
- If *TRANSB* = 't' or 'T' then $op(B) = B^T$;
- If *TRANSB* = 'c' or 'C' then $op(B) = B^H$.

INTEGER M [Input]

On input: The number of rows of the matrices $op(A)$ and C.
Constraint: $M \geq 0$.

INTEGER N [Input]

On input: The number of columns of the matrices $op(B)$ and C.
Constraint: $N \geq 0$.

INTEGER K [Input]

On input: The number of columns of the matrix $op(A)$ and rows of the matrix $op(B)$.
Constraint: $K \geq 0$.

COMPLEX*16 ALPHA [Input]

On input: The value of the scalar α.

COMPLEX*16 A [Input]

On input: An array of dimension (*LDA*, KA) containing the matrix A, where if *TRANSA* = 'n' or 'N' then KA = K and the matrix is held in the leading M by K part of the array, otherwise KA = M and the matrix is held in the leading K by M part of the array.

`INTEGER LDA` [Input]

On input: The leading dimension of the array A as declared in the calling subprogram.

Constraint: If $TRANSA$ = 'n' or 'N' then $LDA \geq max(1, M)$, otherwise $LDA \geq max(1, K)$.

`COMPLEX*16 B` [Input]

On input: An array of dimension (LDB, KB) containing the matrix B, where if $TRANSB$ = 'n' or 'N' then KB = N and the matrix is held in the leading K by N part of the array, otherwise KB = K and the matrix is held in the leading N by K part of the array.

`INTEGER LDB` [Input]

On input: The leading dimension of the array B as declared in the calling subprogram.

Constraint: If $TRANSB$ = 'n' or 'N' then $LDB \geq max(1, K)$, otherwise $LDB \geq max(1, N)$.

`COMPLEX*16 BETA` [Input]

On input: The value of the scalar β.

`COMPLEX*16 C` [Input/Output]

On input: An array of dimension (LDC, N) containing the matrix C, in the leading M by N part of the array. Note that if $BETA = 0$ then the matrix C need not be initialised on input.

On output: The array is overwritten with the result of the operation: $\alpha op(A)op(B) + \beta C$.

`INTEGER LDC` [Input]

On input: The leading dimension of the array C as declared in the calling subprogram.

Constraint: $LDC \geq max(1, M)$.

SGEMM_BATCH Routine Documentation

SGEMM_BATCH computes a batch of *TOTAL_SIZE* SGEMM problems which are split into *GROUP_COUNT* groups, where problems in a group share the same transpose options, problem dimensions, array leading dimensions and scalars α and β. This routine implements the following operation:

```
      TOTAL_SIZE = 0
      PID = 1
      DO 10 J = 1, GROUP_COUNT
          DO 20 I = 1, GROUP_SIZE(J)
              CALL SGEMM(TRANSA(J),TRANSB(J),M(J),N(J),K(J),ALPHA(J),
     &          A(PID),LDA(J),B(PID),LDB(J),BETA(J),C(PID),LDC(J))
              PID = PID + 1
   20     CONTINUE
          TOTAL_SIZE = TOTAL_SIZE + GROUP_SIZE(J)
   10 CONTINUE
```

Note that the above also defines the value of *TOTAL_SIZE*, which is not required as a routine argument, but is used in the documentation below; it is the sum of the *GROUP_SIZE* array.

SGEMM is the standard BLAS single precision real matrix-matrix multiplication routine. For information relating to standard BLAS routines users are referred here:

`http://www.netlib.org/blas/faq.html`

`SGEMM_BATCH` (*TRANSA*,*TRANSB*,*M*,*N*,*K*, *ALPHA*,*A*,*LDA*,*B*,*LDB*,*BETA*,*C*,*LDC*,*GROUP_COUNT*,*GROUP_SIZE*) [SUBROUTINE]

`CHARACTER*1 TRANSA(GROUP_COUNT)` [Input]

On input:

- If *TRANSA(J)* = 'n' or 'N' then for the J^{th} group, for all matrices A, $op(A) = A$;
- If *TRANSA(J)* = 't' or 'T' or 'c' or 'C' then for the J^{th} group, for all matrices A, $op(A) = A^T$.

`CHARACTER*1 TRANSB(GROUP_COUNT)` [Input]

On input:

- If *TRANSB(J)* = 'n' or 'N' then for the J^{th} group, for all matrices B, $op(B) = B$;
- If *TRANSB(J)* = 't' or 'T' or 'c' or 'C' then for the J^{th} group, for all matrices B, $op(B) = B^T$.

`INTEGER M(GROUP_COUNT)` [Input]

On input: For all matrices A and C in the J^{th} group *M(J)* specifies the number of rows of $op(A)$ and C.
Constraint: $M(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`INTEGER N(GROUP_COUNT)` [Input]

On input: For all matrices B and C in the J^{th} group *N(J)* specifies the number of columns of $op(B)$ and C.
Constraint: $N(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`INTEGER K(GROUP_COUNT)` [Input]

On input: For all matrices A and B in the J^{th} group *K(J)* specifies the number of columns of $op(A)$ and rows of $op(B)$.
Constraint: $K(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`REAL ALPHA(GROUP_COUNT)` [Input]

On input: *ALPHA(J)* gives the value of the scalar α for the J^{th} group.

`INTEGER*8 A(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input matrices A.

`INTEGER LDA(GROUP_COUNT)` [Input]

On input: *LDA(J)* gives the leading dimension of all A matrices in the J^{th} group as declared in the calling subprogram.
Constraint: If *TRANSA(J)* = 'n' or 'N' then $LDA(J) \geq max(1, M(J))$, otherwise $LDA(J) \geq max(1, K(J))$ for $J = 1, GROUP\_COUNT$.

`INTEGER*8 B(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input matrices B.

`INTEGER LDB(GROUP_COUNT)` [Input]

On input: *LDB(J)* gives the leading dimension of all B matrices in the J^{th} group as declared in the calling subprogram.
Constraint: If *TRANSB(J)* = 'n' or 'N' then $LDB(J) \geq max(1, K(J))$, otherwise $LDB(J) \geq max(1, N(J))$ for $J = 1, GROUP\_COUNT$.

`REAL BETA(GROUP_COUNT)` [Input]

On input: *BETA(J)* gives the value of the scalar β for the J^{th} group.

`INTEGER*8 C(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input/output matrices C.

`INTEGER LDC(GROUP_COUNT)` [Input]

On input: *LDC(J)* gives the leading dimension of all C matrices in the J^{th} group as declared in the calling subprogram.
Constraint: $LDC(J) \geq max(1, M(J))$ for $J = 1, GROUP\_COUNT$.

`INTEGER GROUP_COUNT` [Input]

On input: Specifies the number of groups.
Constraint: $GROUP\_COUNT \geq 0$.

`INTEGER GROUP_SIZE(GROUP_COUNT)` [Input]

On input: *GROUP_SIZE(J)* specifies the number of SGEMM problems in the J^{th} group.
Constraint: $GROUP\_SIZE(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

DGEMM_BATCH Routine Documentation

DGEMM_BATCH computes a batch of *TOTAL_SIZE* DGEMM problems which are split into *GROUP_COUNT* groups, where problems in a group share the same transpose options, problem dimensions, array leading dimensions and scalars α and β. This routine implements the following operation:

```
      TOTAL_SIZE = 0
      PID = 1
      DO 10 J = 1, GROUP_COUNT
          DO 20 I = 1, GROUP_SIZE(J)
              CALL DGEMM(TRANSA(J),TRANSB(J),M(J),N(J),K(J),ALPHA(J),
     &          A(PID),LDA(J),B(PID),LDB(J),BETA(J),C(PID),LDC(J))
              PID = PID + 1
   20     CONTINUE
          TOTAL_SIZE = TOTAL_SIZE + GROUP_SIZE(J)
   10 CONTINUE
```

Note that the above also defines the value of *TOTAL_SIZE*, which is not required as a routine argument, but is used in the documentation below; it is the sum of the *GROUP_SIZE* array.

DGEMM is the standard BLAS double precision real matrix-matrix multiplication routine. For information relating to standard BLAS routines users are referred here:

`http://www.netlib.org/blas/faq.html`

`DGEMM_BATCH` (*TRANSA*,*TRANSB*,*M*,*N*,*K*, *ALPHA*,*A*,*LDA*,*B*,*LDB*,*BETA*,*C*,*LDC*,*GROUP_COUNT*,*GROUP_SIZE*) [SUBROUTINE]

`CHARACTER*1 TRANSA(GROUP_COUNT)` [Input]

On input:

- If *TRANSA(J)* = 'n' or 'N' then for the J^{th} group, for all matrices A, $op(A) = A$;
- If *TRANSA(J)* = 't' or 'T' or 'c' or 'C' then for the J^{th} group, for all matrices A, $op(A) = A^T$ (matrix transpose).

`CHARACTER*1 TRANSB(GROUP_COUNT)` [Input]

On input:

- If *TRANSB(J)* = 'n' or 'N' then for the J^{th} group, for all matrices B, $op(B) = B$;
- If *TRANSB(J)* = 't' or 'T' or 'c' or 'C' then for the J^{th} group, for all matrices B, $op(B) = B^T$ (matrix transpose).

`INTEGER M(GROUP_COUNT)` [Input]

On input: For all matrices A and C in the J^{th} group *M(J)* specifies the number of rows of $op(A)$ and C.
Constraint: $M(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`INTEGER N(GROUP_COUNT)` [Input]

On input: For all matrices B and C in the J^{th} group *N(J)* specifies the number of columns of $op(B)$ and C.
Constraint: *N(J)* ≥ 0 for $J = 1, GROUP\_COUNT$.

`INTEGER K(GROUP_COUNT)` [Input]

On input: For all matrices A and B in the J^{th} group *K(J)* specifies the number of columns of $op(A)$ and rows of $op(B)$.
Constraint: *K(J)* ≥ 0 for $J = 1, GROUP\_COUNT$.

`DOUBLE PRECISION ALPHA(GROUP_COUNT)` [Input]

On input: *ALPHA(J)* gives the value of the scalar α for the J^{th} group.

`INTEGER*8 A(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input matrices A.

`INTEGER LDA(GROUP_COUNT)` [Input]

On input: *LDA(J)* gives the leading dimension of all A matrices in the J^{th} group as declared in the calling subprogram.
Constraint: If *TRANSA(J)* = 'n' or 'N' then *LDA(J)* $\geq max(1, M(J))$, otherwise *LDA(J)* $\geq max(1, K(J))$ for $J = 1, GROUP\_COUNT$.

`INTEGER*8 B(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input matrices B.

`INTEGER LDB(GROUP_COUNT)` [Input]

On input: *LDB(J)* gives the leading dimension of all B matrices in the J^{th} group as declared in the calling subprogram.
Constraint: If *TRANSB(J)* = 'n' or 'N' then *LDB(J)* $\geq max(1, K(J))$, otherwise *LDB(J)* $\geq max(1, N(J))$ for $J = 1, GROUP\_COUNT$.

`DOUBLE PRECISION BETA(GROUP_COUNT)` [Input]

On input: *BETA(J)* gives the value of the scalar β for the J^{th} group.

`INTEGER*8 C(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input/output matrices C.

`INTEGER LDC(GROUP_COUNT)` [Input]

On input: *LDC(J)* gives the leading dimension of all C matrices in the J^{th} group as declared in the calling subprogram.
Constraint: *LDC(J)* $\geq max(1, M(J))$ for $J = 1, GROUP\_COUNT$.

`INTEGER GROUP_COUNT` [Input]

On input: Specifies the number of groups.
Constraint: *GROUP_COUNT* ≥ 0.

`INTEGER GROUP_SIZE(GROUP_COUNT)` [Input]

On input: *GROUP_SIZE(J)* specifies the number of DGEMM problems in the J^{th} group.
Constraint: *GROUP_SIZE(J)* ≥ 0 for $J = 1, GROUP\_COUNT$.

CGEMM_BATCH Routine Documentation

CGEMM_BATCH computes a batch of *TOTAL_SIZE* CGEMM problems which are split into *GROUP_COUNT* groups, where problems in a group share the same transpose options, problem dimensions, array leading dimensions and scalars α and β. This routine implements the following operation:

```
      TOTAL_SIZE = 0
      PID = 1
      DO 10 J = 1, GROUP_COUNT
          DO 20 I = 1, GROUP_SIZE(J)
              CALL CGEMM(TRANSA(J),TRANSB(J),M(J),N(J),K(J),ALPHA(J),
     &          A(PID),LDA(J),B(PID),LDB(J),BETA(J),C(PID),LDC(J))
              PID = PID + 1
   20     CONTINUE
          TOTAL_SIZE = TOTAL_SIZE + GROUP_SIZE(J)
   10 CONTINUE
```

Note that the above also defines the value of *TOTAL_SIZE*, which is not required as a routine argument, but is used in the documentation below; it is the sum of the *GROUP_SIZE* array.

CGEMM is the standard BLAS single precision complex matrix-matrix multiplication routine. For information relating to standard BLAS routines users are referred here:

`http://www.netlib.org/blas/faq.html`

`CGEMM_BATCH` (*TRANSA*,*TRANSB*,*M*,*N*,*K*, *ALPHA*,*A*,*LDA*,*B*,*LDB*,*BETA*,*C*,*LDC*,*GROUP_COUNT*,*GROUP_SIZE*) [SUBROUTINE]

`CHARACTER*1 TRANSA(GROUP_COUNT)` [Input]

On input:

- If *TRANSA(J)* = 'n' or 'N' then for the J^{th} group, for all matrices A, $op(A) = A$;
- If *TRANSA(J)* = 't' or 'T' then for the J^{th} group, for all matrices A, $op(A) = A^T$ (matrix transpose);
- If *TRANSA(J)* = 'c' or 'C' then for the J^{th} group, for all matrices A, $op(A) = A^H$ (matrix conjugate transpose).

`CHARACTER*1 TRANSB(GROUP_COUNT)` [Input]

On input:

- If *TRANSB(J)* = 'n' or 'N' then for the J^{th} group, for all matrices B, $op(B) = B$;
- If *TRANSB(J)* = 't' or 'T' then for the J^{th} group, for all matrices B, $op(B) = B^T$ (matrix transpose);
- If *TRANSB(J)* = 'c' or 'C' then for the J^{th} group, for all matrices B, $op(B) = B^H$ (matrix conjugate transpose).

`INTEGER M(GROUP_COUNT)` [Input]

On input: For all matrices A and C in the J^{th} group *M(J)* specifies the number of rows of $op(A)$ and C.
Constraint: $M(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`INTEGER N(GROUP_COUNT)` [Input]

On input: For all matrices B and C in the J^{th} group *N(J)* specifies the number of columns of $op(B)$ and C.
Constraint: $N(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`INTEGER K(GROUP_COUNT)` [Input]

On input: For all matrices A and B in the J^{th} group *K(J)* specifies the number of columns of $op(A)$ and rows of $op(B)$.
Constraint: $K(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`COMPLEX ALPHA(GROUP_COUNT)` [Input]

On input: *ALPHA(J)* gives the value of the scalar α for the J^{th} group.

`INTEGER*8 A(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input matrices A.

`INTEGER LDA(GROUP_COUNT)` [Input]

On input: *LDA(J)* gives the leading dimension of all A matrices in the J^{th} group as declared in the calling subprogram.
Constraint: If *TRANSA(J)* = 'n' or 'N' then $LDA(J) \geq max(1, M(J))$, otherwise $LDA(J) \geq max(1, K(J))$ for $J = 1, GROUP\_COUNT$.

`INTEGER*8 B(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input matrices B.

`INTEGER LDB(GROUP_COUNT)` [Input]

On input: *LDB(J)* gives the leading dimension of all B matrices in the J^{th} group as declared in the calling subprogram.
Constraint: If *TRANSB(J)* = 'n' or 'N' then $LDB(J) \geq max(1, K(J))$, otherwise $LDB(J) \geq max(1, N(J))$ for $J = 1, GROUP\_COUNT$.

`COMPLEX BETA(GROUP_COUNT)` [Input]

On input: *BETA(J)* gives the value of the scalar β for the J^{th} group.

`INTEGER*8 C(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input/output matrices C.

`INTEGER LDC(GROUP_COUNT)` [Input]

On input: *LDC(J)* gives the leading dimension of all C matrices in the J^{th} group as declared in the calling subprogram.
Constraint: $LDC(J) \geq max(1, M(J))$ for $J = 1, GROUP\_COUNT$.

`INTEGER GROUP_COUNT` [Input]

On input: Specifies the number of groups.
Constraint: $GROUP\_COUNT \geq 0$.

`INTEGER GROUP_SIZE(GROUP_COUNT)` [Input]

On input: *GROUP_SIZE(J)* specifies the number of CGEMM problems in the J^{th} group.

Constraint: *GROUP_SIZE(J)* ≥ 0 for $J = 1, GROUP\_COUNT$.

ZGEMM_BATCH Routine Documentation

GEMM_BATCH computes a batch of *TOTAL_SIZE* ZGEMM problems which are split into *GROUP_COUNT* groups, where problems in a group share the same transpose options, problem dimensions, array leading dimensions and scalars α and β. This routine implements the following operation:

```
      TOTAL_SIZE = 0
      PID = 1
      DO 10 J = 1, GROUP_COUNT
          DO 20 I = 1, GROUP_SIZE(J)
              CALL ZGEMM(TRANSA(J),TRANSB(J),M(J),N(J),K(J),ALPHA(J),
     &          A(PID),LDA(J),B(PID),LDB(J),BETA(J),C(PID),LDC(J))
              PID = PID + 1
   20     CONTINUE
          TOTAL_SIZE = TOTAL_SIZE + GROUP_SIZE(J)
   10 CONTINUE
```

Note that the above also defines the value of *TOTAL_SIZE*, which is not required as a routine argument, but is used in the documentation below; it is the sum of the *GROUP_SIZE* array.

ZGEMM is the standard BLAS double precision complex matrix-matrix multiplication routine. For information relating to standard BLAS routines users are referred here:

`http://www.netlib.org/blas/faq.html`

`ZGEMM_BATCH` (*TRANSA*,*TRANSB*,*M*,*N*,*K*, [SUBROUTINE]
ALPHA,*A*,*LDA*,*B*,*LDB*,*BETA*,*C*,*LDC*,*GROUP_COUNT*,*GROUP_SIZE*)

`CHARACTER*1 TRANSA(GROUP_COUNT)` [Input]

On input:

- If *TRANSA(J)* = 'n' or 'N' then for the J^{th} group, for all matrices A, $op(A) = A$;
- If *TRANSA(J)* = 't' or 'T' then for the J^{th} group, for all matrices A, $op(A) = A^T$ (matrix transpose);
- If *TRANSA(J)* = 'c' or 'C' then for the J^{th} group, for all matrices A, $op(A) = A^H$ (matrix conjugate transpose).

`CHARACTER*1 TRANSB(GROUP_COUNT)` [Input]

On input:

- If *TRANSB(J)* = 'n' or 'N' then for the J^{th} group, for all matrices B, $op(B) = B$;
- If *TRANSB(J)* = 't' or 'T' then for the J^{th} group, for all matrices B, $op(B) = B^T$ (matrix transpose);
- If *TRANSB(J)* = 'c' or 'C' then for the J^{th} group, for all matrices B, $op(B) = B^H$ (matrix conjugate transpose).

`INTEGER M(GROUP_COUNT)` [Input]

On input: For all matrices A and C in the J^{th} group *M(J)* specifies the number of rows of $op(A)$ and C.
Constraint: $M(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`INTEGER N(GROUP_COUNT)` [Input]

On input: For all matrices B and C in the J^{th} group *N(J)* specifies the number of columns of $op(B)$ and C.
Constraint: $N(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`INTEGER K(GROUP_COUNT)` [Input]

On input: For all matrices A and B in the J^{th} group *K(J)* specifies the number of columns of $op(A)$ and rows of $op(B)$.
Constraint: $K(J) \geq 0$ for $J = 1, GROUP\_COUNT$.

`COMPLEX*16 ALPHA(GROUP_COUNT)` [Input]

On input: *ALPHA(J)* gives the value of the scalar α for the J^{th} group.

`INTEGER*8 A(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input matrices A.

`INTEGER LDA(GROUP_COUNT)` [Input]

On input: *LDA(J)* gives the leading dimension of all A matrices in the J^{th} group as declared in the calling subprogram.
Constraint: If *TRANSA(J)* = 'n' or 'N' then $LDA(J) \geq max(1, M(J))$, otherwise $LDA(J) \geq max(1, K(J))$ for $J = 1, GROUP\_COUNT$.

`INTEGER*8 B(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input matrices B.

`INTEGER LDB(GROUP_COUNT)` [Input]

On input: *LDB(J)* gives the leading dimension of all B matrices in the J^{th} group as declared in the calling subprogram.
Constraint: If *TRANSB(J)* = 'n' or 'N' then $LDB(J) \geq max(1, K(J))$, otherwise $LDB(J) \geq max(1, N(J))$ for $J = 1, GROUP\_COUNT$.

`COMPLEX*16 BETA(GROUP_COUNT)` [Input]

On input: *BETA(J)* gives the value of the scalar β for the J^{th} group.

`INTEGER*8 C(TOTAL_SIZE)` [Input]

On input: An array of pointers to *TOTAL_SIZE* input/output matrices C.

`INTEGER LDC(GROUP_COUNT)` [Input]

On input: *LDC(J)* gives the leading dimension of all C matrices in the J^{th} group as declared in the calling subprogram.
Constraint: $LDC(J) \geq max(1, M(J))$ for $J = 1, GROUP\_COUNT$.

`INTEGER GROUP_COUNT` [Input]

On input: Specifies the number of groups.
Constraint: $GROUP\_COUNT \geq 0$.

`INTEGER GROUP_SIZE(GROUP_COUNT)` [Input]

On input: *GROUP_SIZE(J)* specifies the number of ZGEMM problems in the J^{th} group.

Constraint: *GROUP_SIZE(J)* ≥ 0 for $J = 1, GROUP\_COUNT$.

4 LAPACK: Linear Algebra Package

4.1 Introduction to LAPACK

LAPACK ([4]) is a library of FORTRAN 77 subroutines for solving commonly occurring problems in numerical linear algebra. LAPACK components can solve systems of linear equations, linear least squares problems, eigenvalue problems and singular value problems. Dense and banded matrices are provided for, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices.

LAPACK routines are written so that as much as possible of the computations is performed by calls to the BLAS. The efficiency of LAPACK routines depends, in large part, on the efficiency of the BLAS being called. Block algorithms are employed wherever possible to maximize the use of calls to level 3 BLAS, which generally run faster than lower level BLAS due to the high number of operations per memory access.

The performance of some of the LAPACK routines has been further improved by reworking the computational algorithms. Some of the LAPACK routines contained in Arm Performance Libraries are therefore based on code that is different from the LAPACK sources available in the public domain. In all these cases the algorithmic and numerical properties of the original LAPACK routines have been strictly preserved. Furthermore, key LAPACK routines have been treated using OpenMP to take advantage of multiple processors when running on SMP machines. Your application will automatically benefit when you link with the OpenMP versions of Arm Performance Libraries.

4.2 Reference sources for LAPACK

The LAPACK homepage can be accessed on the World Wide Web via the URL address:

`http://www.netlib.org/lapack/`

The on-line version of the Lapack User's Guide, Third Edition ([4]) is available from this homepage, or directly using the URL:

`http://www.netlib.org/lapack/lug/index.html`

The standard source code is available for download from netlib:

`http://www.netlib.org/lapack/lapack.tgz`

A list of known problems, bugs, and compiler errors for LAPACK, as well as an errata list for the LAPACK User's Guide ([4]), is maintained on netlib

`http://www.netlib.org/lapack/release_notes`

A LAPACK FAQ (Frequently Asked Questions) file can also be accessed via the LAPACK homepage

`http://www.netlib.org/lapack/faq.html`

For C users, Arm Performance Libraries includes the LAPACKE C interfaces to Fortran LAPACK. Names of these routines are similar to the Fortran routines except they are prefixed by the string `LAPACKE_`. For example, the Fortran LAPACK routine `dgetrf` becomes `LAPACKE_dgetrf` in C. Other differences between Fortran BLAS and CBLAS interfaces:

- Scalar input arguments are passed by value in LAPACKE interfaces. FORTRAN interfaces pass all arguments (except for character string *length* arguments that are normally hidden from FORTRAN programmers) by reference.
- Unlike FORTRAN, C has no native *complex* data type. Arm Performance Libraries C routines which operate on complex data use the types `armpl_singlecomplex_t` and `armpl_doublecomplex_t` defined in `armpl.h` for single and double precision computations respectively.
- As with Fortran, all LAPACKE array arguments are required to be stored in contiguous memory. However, all Fortran LAPACK routines which take one or more matrices (2D arrays) as arguments have an extra parameter in the LAPACKE interface, which appears before all other parameters. This parameter is of integer type and is named `matrix_order`, and it can take one of the values `LAPACK_ROW_MAJOR` or `LAPACK_COL_MAJOR`, where `LAPACK_ROW_MAJOR` and `LAPACK_COL_MAJOR` are integers defined in `armpl.h`

  The value `LAPACK_COL_MAJOR` indicates that elements within any column of each array are contiguous in memory while elements within array rows are offset by a constant stride. This is the usual Fortran storage order. Such strides are given by "leading dimension" arguments (such as `LDA`) in the Fortran and C interfaces.

  The value `LAPACK_ROW_MAJOR` indicates that elements within any row of each 2D array are contiguous in memory while elements within array columns are offset by a constant stride such as `LDA`.
- Fortran's `character` type arguments are passed as values of the C `char` type.
- In general, Fortran "workspace" type arguments (with names such as `WORK`, `RWORK` and `CWORK`) do not appear in the C interfaces - workspace memory is allocated internally. However, LAPACKE also includes variants which *do* have the workspace arguments - these variants have the string `_work` appended to the function name, for example `LAPACKE_dgetrf_work`.

See the `armpl.h` header file to find prototypes for all LAPACKE functions. Note that the functions may also be called from a C++ program if you include `armpl.h`.

5 Fast Fourier Transforms (FFTs)

5.1 Introduction to FFTs

There are two main types of Discrete Fourier Transform (DFT):

- routines for the transformation of complex data: in the Arm Performance Libraries, these routines have names beginning with `ZFFT` or `CFFT`, for double and single precision, respectively;
- routines for the transformation of real to complex data and vice versa: in the Arm Performance Libraries the names for the former begin with `DZFFT` or `SCFFT`, for double and single precision, respectively; the names for the latter begin with `ZDFFT` or `CSFFT`.

The following subsections provide definitions of the DFT for complex and real data types, and some guidelines on the efficient use of the Arm Performance Libraries FFT routines.

C interfaces to Arm Performance Libraries FFT routines are not specifically documented here, but their prototypes are in the `armpl.h` header file, which may be consulted for specific functions. The C FFT interfaces differ from FORTRAN interfaces in the following major respects:

- Scalar input arguments are passed by value in C interfaces. FORTRAN interfaces pass all arguments (except for character string *length* arguments that are normally hidden from FORTRAN programmers) by reference.
- Unlike FORTRAN, C has no native *complex* data type. Arm Performance Libraries C interfaces to FFT routines which operate on complex data use the types `armpl_singlecomplex_t` and `armpl_doublecomplex_t` defined in `armpl.h` for single and double precision computations respectively. Some of the programs in the Arm Performance Libraries examples directory (see Section 2.6 [Examples], page 5) make use of these types.

5.1.1 Transform definitions and Storage for Complex Data

The simplest transforms to describe are those performed on sequences of complex data. Such data are stored as arrays of type complex. The result of a complex FFT is also a complex sequence of the same length and, for the simple interfaces, is written back to the original array. Where multiple (m, say), same-length sequences (of length n) of complex data are to be transformed, the sequences are held in a single complex array; in the simple interfaces the array will be of length $m * n$ containing m end-to-end sequences and the results of the m FFTs are returned in the original array. Expert interfaces are provided which give: greater flexibility in the storage of the original data and results, user provided scaling, and whether results should be written to a separate array or not.

The definition of a complex DFT used here is given by:

$$\tilde{x}_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} x_k \exp\left(\pm i \frac{2\pi jk}{n}\right) \text{ for } j = 0, 1, \ldots, n-1$$

where x_k are the complex data to be transformed, $\tilde{x}_j$ are the transformed data, and the sign of $\pm$ determines the direction of the transform: $(-)$ for forward and $(+)$ for backward.

Note that, in this definition, both directional transforms have the same scaling and performing both consecutively recovers the original data; this is the prescribed scaling provided in the simple FFT interfaces, whereas, in the expert interfaces, the scaling factor must be supplied by the user.

For the simple interfaces, a two dimensional array of complex data, with m rows and n columns is stored in the same order as a set of n sequences of length m (as described above). That is, column elements are stored contiguously and the first element of the next column follows the last element of the current column. In the expert interfaces, column elements may be separated by a fixed step length (increment) while row elements may be separated by a second increment; if the first increment is 1 and the second increment is m then we have the same storage as in the simple interface.

The definition of a complex 2D DFT used here is given by:

$$\tilde{x}_{jp} = \frac{1}{\sqrt{m * n}} \sum_{l=0}^{m-1} \sum_{k=0}^{n-1} x_{kl} \exp\left(\pm i \frac{2\pi jk}{n}\right) \exp\left(\pm i \frac{2\pi pl}{m}\right)$$

for $j = 0, 1, \ldots, n-1$ and $p = 0, 1, \ldots, m-1$, where x_{kl} are the complex data to be transformed, $\tilde{x}_{jp}$ are the transformed data, and the sign of $\pm$ determines the direction of the transform.

5.1.2 Transform definitions and Storage for Real Data

The DFT of a sequence of real data results in a special form of complex sequence known as a Hermitian sequence.

If the original sequence is purely real valued, i.e. $z_j = x_j$, then the definition of the real DFT used here is given by:

$$\tilde{z}_j = a_j + ib_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} x_k \exp\left(-i \frac{2\pi jk}{n}\right) \text{ for } j = 0, 1, \ldots, n-1$$

where x_k are the real data to be transformed, $\tilde{z}_j$ are the transformed complex data.

In full complex representation, the Hermitian sequence would be a sequence of n complex values $\tilde{z}_j$ for $j = 0, 1, ..., n-1$, where $\tilde{z}_{n-j}$ is the complex conjugate of $\tilde{z}_j$ for $j = 1, 2, ..., (n-1)/2$; $\tilde{z}_0$ is real valued; and, if n is even, $\tilde{z}_{n/2}$ is also real valued. The symmetries defining Hermitian sequence mean that it can be stored using reduced amount of memory.

In Arm Performance Libraries, there are two storage formats for the representation of Hermitian sequences. The first format, called Hermitian-packed format, is used on output from `DZFFT` routines and on input to `ZDFFT` routines and is defined as follows: let X be an array of length n and with first index 0,

- $X(j)$ contains the real part of $\tilde{z}_j$ (i.e. a_j) for $j = 0, ..., n/2$
- $X(n-j)$ contains the imaginary part of $\tilde{z}_j$ (i.e. b_j) for $j = 1, ..., (n-1)/2$

with integer division used where appropriate. As seen, a Hermitian sequence can be fully represented by a set of n real values, where n is the length of the original real sequence. It is therefore conventional for the array containing the real sequence to be overwritten by such a representation of the transformed Hermitian sequence.

An alternative way to store the Hermitian sequence, called complex-Hermitian format, is to keep the complex representation, store only the first $n/2+1$ complex numbers and drop the remaining elements that contain redundant information. This approach can be conveniently extended to multiple dimensions. For example, a 3D real data set of size $l*m*n$ can be transformed into Fourier space as a complex data set of size $(l/2+1)*m*n$. It is still possible to use $l*m*n$ real numbers to store all Fourier coefficients but interpreting such information would be extremely difficult. Some subroutines in Arm Performance Libraries (such as `DZFFT1D/1M/2D/3D`) use the complex-Hermitian format. In two dimensions the non-stored coefficients are available from the fact that $\tilde{z}_{m-i,n-j}$ is the complex conjugate of $\tilde{z}_{i,j}$ for $i = 0, 1, ..., m/2$. Similarly, in three dimensions, $\tilde{z}_{l-i,m-j,n-k}$ is the complex conjugate of $\tilde{z}_{i,j,k}$.

The following table shows the storage of a one-dimensional Hermitian sequence with 8 elements.

Real Input x_j	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
Complex Output $\tilde{z}_j = a_j + ib_j$	$\tilde{z}_0$	$\tilde{z}_1$	$\tilde{z}_2$	$\tilde{z}_3$	$\tilde{z}_4$	$\tilde{z}_5 = \tilde{z}_3^*$	$\tilde{z}_6 = \tilde{z}_2^*$	$\tilde{z}_7 = \tilde{z}_1^*$
Hermitian-Packed Storage	a_0	a_1	a_2	a_3	a_4	b_3	b_2	b_1
Complex-Hermitian Storage	$\tilde{z}_0$	$\tilde{z}_1$	$\tilde{z}_2$	$\tilde{z}_3$	$\tilde{z}_4$	–	–	–

Given a Hermitian sequence, the inverse discrete transform can be defined as:

$$x_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \tilde{z}_k \exp\left(i\frac{2\pi jk}{n}\right) \text{ for } j = 0, 1, \ldots, n-1$$

or if written using the real and complex components:

$$x_j = \frac{1}{\sqrt{n}} \left(a_0 + 2 \sum_{k=1}^{n/2-1} \left(a_k \cos\left(\frac{2\pi jk}{n}\right) - b_k \sin\left(\frac{2\pi jk}{n}\right)\right) + a_{n/2} \right)$$

where $a_{n/2} = 0$ if n is odd, and $\tilde{z}_k = a_k + ib_k$ is the Hermitian sequence to be transformed.

Note that, in the second definition above, the inverse transform has a negative sign in the exponent. So if a Hermitian sequence is stored in a Hermitian-packed format (for DZFFT, DZFFTM, SCFFT and SCFFTM), performing a forward and a backward transform consecutively does not recover the original data. To recover original real data, or otherwise to perform an inverse transform on a set of Hermitian data, the Hermitian data must be conjugated prior to performing the transform (i.e. changing the sign of the stored imaginary parts). This however does not apply to the remaining subroutines using the complex-Hermitian data storage.

5.1.3 Efficiency

The efficiency of the FFT is maximized by choosing the sequence length to be a power of 2. Good efficiency can also be achieved when the sequence length has small prime factors, up to a factor 13; however, the time taken for an FFT increases as the size of the prime factor increases.

5.1.4 Default and Generated Plans

For those FFT routines that can be initialized prior to computing the FFTs, the initialization can be performed in one of two ways. In either case, initialization involves the storing of the factorization of N, and the twiddle factors associated with this factorization, in the communication array *COMM*.

The simpler way to initialize is by setting the argument *MODE* to zero. This means that a default plan, for the given input dimensions, is used to calculate the FFT. This has the advantage that the initialization phase is very quick and is generally a small fraction of the time required to perform the FFT computation. However, for some problem dimensions the default plan may not be optimal, especially where there is a mixture of prime factors.

Under some circumstances, optimality of performance of an FFT computation may be crucial. For example, where a very large number of FFTs are to be performed on problems of a fixed size (e.g. N remains the same), then it is best to initialize by setting the argument *MODE* to 100. This will time a number of plans (this number can be quite large when N has a significant number of prime factors) and initialize using the plan with the best time. Using this form of initialization can, potentially, lead to significant improvements in the performance of the FFT computation for the given dimensions.

Where problem dimensions will not change over a number of runs of a program, the communication array could, for example, be written out to a file during an initialization run, and then read in from the same file on subsequent computation runs. This would be effective for problem dimensions that have a large number of possible plans (factor orderings and groupings) and therefore take a significant amount of time to find the optimal plan.

Please consult the individual FFT routine documents to determine whether plan generation is enabled.

5.1.5 FFTW Interface Support

FFTW3 wrappers in Arm Performance Libraries currently support single and double precision the basic and advanced FFTW3 functions with `_dft_` in the function name (i.e. real-to-real and guru transforms are not currently supported).

Existing source code calling in to the supported FFTW3 functions need not be modified. Include `fftw3.h` as normal in your source code. In compiling and linking follow the instructions given in `USAGE.txt` as normal. Both the single and double precision functions are to be found in a single library.

Where plans are not currently supported the interface will return a NULL pointer except for MPI functions, which are not provided at all (i.e. you will get a linker error if you try to link code which calls into the FFTW3 MPI functions).

All unsupported functions will emit an error message and functions which return a value will return an error indicator as documented by FFTW3.

Note that since the basic and advanced FFTW interfaces specify `int` and `unsigned` datatypes then even for ilp64 Arm Performance Libraries the supported FFTW interfaces support only 32-bit integers.

Behaviour is based on FFTW3 at version 3.3.5.

5.2 FFTs on Complex Sequences

5.2.1 FFT of a single sequence

The routines documented here compute the discrete Fourier transform (DFT) of a sequence of complex numbers in either single or double precision arithmetic. The DFT is computed using a highly-efficient FFT algorithm. There are two sets of interfaces available: simple drivers and expert drivers. The simple drivers perform in-place transforms on data held contiguously in memory using a fixed scaling factor; these are simpler to use and are sufficient for many problems. The expert drivers offer greater flexibility by including a number of additional arguments. These allow you to control: the scaling factor applied; whether the result should be output to a separate vector; and, the increments used in storing successive elements of both the input sequence and the result.

ZFFT1D Routine Documentation

ZFFT1D (*MODE*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZFFT1D. On input:

- *MODE*=0 : only default initializations (specific to *N*) are performed; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT1D.
- *MODE*=1 : a backward (reverse) transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT1D.
- *MODE*=−2 : initializations and a forward transform are performed.
- *MODE*=2 : initializations and a backward transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER N [Input]

On input: *N* is the length of the complex sequence *X*

COMPLEX*16 X(N) [Input/Output]

On input: *X* contains the complex sequence of length *N* to be transformed.
On output: *X* contains the transformed sequence.

COMPLEX*16 COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL ZFFT1D(0,N,X,COMM,INFO)
      CALL ZFFT1D(-1,N,X,COMM,INFO)
      CALL ZFFT1D(-1,N,Y,COMM,INFO)
      DO 10 I = 1, N
         X(I) = X(I)*DCONJG(Y(I))
10    CONTINUE
      CALL ZFFT1D(1,N,X,COMM,INFO)
```

CFFT1D Routine Documentation

CFFT1D (*MODE*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CFFT1D. On input:

- *MODE*=0 : only default initializations (specific to *N*) are performed; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward transform is performed. Initializations are assumed to have been performed by a prior call to CFFT1D.
- *MODE*=1 : a backward (reverse) transform is performed. Initializations are assumed to have been performed by a prior call to CFFT1D.
- *MODE*=−2 : (default) initializations and a forward transform are performed.
- *MODE*=2 : (default) initializations and a backward transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER N [Input]

On input: *N* is the length of the complex sequence *X*

COMPLEX X(N) [Input/Output]

On input: *X* contains the complex sequence of length *N* to be transformed.
On output: *X* contains the transformed sequence.

COMPLEX COMM(5*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL CFFT1D(0,N,X,COMM,INFO)
      CALL CFFT1D(-1,N,X,COMM,INFO)
      CALL CFFT1D(-1,N,Y,COMM,INFO)
      DO 10 I = 1, N
         X(I) = X(I)*CONJG(Y(I))
10    CONTINUE
      CALL CFFT1D(1,N,X,COMM,INFO)
```

ZFFT1DX Routine Documentation

ZFFT1DX (*MODE*,*SCALE*,*INPL*,*N*,*X*,*INCX*,*Y*,*INCY*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZFFT1DX.
On input:

- *MODE*=0 : only initializations (specific to the value of N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT1DX.
- *MODE*=1 : a backward (reverse) transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT1DX.
- *MODE*=−2 : (default) initializations and a forward transform are performed.
- *MODE*=2 : (default) initializations and a backward transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of N) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of N.

DOUBLE PRECISION SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequence

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then X is overwritten by the output sequence; otherwise the output sequence is returned in Y.

INTEGER N [Input]

On input: N is the number of elements to be transformed

COMPLEX*16 X(1+(N-1)*INCX) [Input/Output]

On input: X contains the complex sequence of length N to be transformed, with the ith element stored in X(1+(i-1)*INCX).
On output: if *INPL* is .TRUE. then X contains the transformed sequence in the same locations as on input; otherwise X remains unchanged.

INTEGER INCX [Input]

On input: *INCX* is the increment used to store successive elements of a sequence in X.
Constraint: *INCX* > 0.

COMPLEX*16 Y(1+(N-1)*INCY) [Output]

On output: if *INPL* is .FALSE. then Y contains the transformed sequence, with the ith element stored in Y(1+(i-1)*INCY); otherwise Y is not referenced.

`INTEGER INCY` [Input]

On input: *INCY* is the increment used to store successive elements of a sequence in *Y*. If *INPL* is .TRUE. then *INCY* is not referenced.
Constraint: *INCY* > 0.

`COMPLEX*16 COMM(3*N+100)` [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

`INTEGER INFO` [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward FFTs are performed unscaled and in-place on contiguous
C     vectors X and Y following initialization. Manipulations on
C     resultant Fourier coefficients are stored in X which is then
C     transformed back.
C
        SCALE = 1.0D0
        INPL = .TRUE.
        CALL ZFFT1DX(0,SCALE,INPL,N,X,1,DUM,1,COMM,INFO)
        CALL ZFFT1DX(-1,SCALE,INPL,N,X,1,DUM,1,COMM,INFO)
        CALL ZFFT1DX(-1,SCALE,INPL,N,Y,1,DUM,1,COMM,INFO)
        DO 10 I = 1, N
           X(I) = X(I)*DCONJG(Y(I))/DBLE(N)
   10   CONTINUE
        CALL ZFFT1DX(1,SCALE,INPL,N,X,1,DUM,1,COMM,INFO)
```

CFFT1DX Routine Documentation

CFFT1DX (*MODE*,*SCALE*,*INPL*,*N*,*X*,*INCX*,*Y*,*INCY*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CFFT1DX.
On input:

- *MODE*=0 : only initializations (specific to the value of N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward transform is performed. Initializations are assumed to have been performed by a prior call to CFFT1DX.
- *MODE*=1 : a backward (reverse) transform is performed. Initializations are assumed to have been performed by a prior call to CFFT1DX.
- *MODE*=−2 : (default) initializations and a forward transform are performed.
- *MODE*=2 : (default) initializations and a backward transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of N) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of N.

REAL SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequence

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then X is overwritten by the output sequence; otherwise the output sequence is returned in Y.

INTEGER N [Input]

On input: N is the number of elements to be transformed

COMPLEX X(1+(N-1)*INCX) [Input/Output]

On input: X contains the complex sequence of length N to be transformed, with the ith element stored in X(1+(i-1)*INCX).
On output: if *INPL* is .TRUE. then X contains the transformed sequence in the same locations as on input; otherwise X remains unchanged.

INTEGER INCX [Input]

On input: *INCX* is the increment used to store successive elements of a sequence in X.
Constraint: *INCX* > 0.

COMPLEX Y(1+(N-1)*INCY) [Output]

On output: if *INPL* is .FALSE. then Y contains the transformed sequence, with the ith element stored in Y(1+(i-1)*INCY); otherwise Y is not referenced.

`INTEGER INCY` [Input]

On input: *INCY* is the increment used to store successive elements of a sequence in *Y*. If *INPL* is .TRUE. then *INCY* is not referenced.
Constraint: *INCY* > 0.

`COMPLEX COMM(5*N+100)` [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

`INTEGER INFO` [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward FFTs are performed unscaled and in-place on contiguous
C     vectors X and Y following initialization. Manipulations on
C     resultant Fourier coefficients are stored in X which is then
C     transformed back.
C
        SCALE = 1.0
        INPL = .TRUE.
        CALL CFFT1DX(0,SCALE,INPL,N,X,1,DUM,1,COMM,INFO)
        CALL CFFT1DX(-1,SCALE,INPL,N,X,1,DUM,1,COMM,INFO)
        CALL CFFT1DX(-1,SCALE,INPL,N,Y,1,DUM,1,COMM,INFO)
        DO 10 I = 1, N
           X(I) = X(I)*CONJG(Y(I))/REAL(N)
   10   CONTINUE
        CALL CFFT1DX(1,SCALE,INPL,N,X,1,DUM,1,COMM,INFO)
```

5.2.2 FFT of multiple complex sequences

The routines documented here compute the discrete Fourier transforms (DFTs) of a number of sequences of complex numbers in either single or double precision arithmetic. The sequences must all have the same length. The DFTs are computed using a highly-efficient FFT algorithm. There are two sets of interfaces available: simple drivers and expert drivers. The simple drivers perform in-place transforms on data held contiguously in memory using a fixed scaling factor; these are simpler to use and are sufficient for many problems. The expert drivers offer greater flexibility by including a number of additional arguments. These allow you to control: the scaling factor applied; whether the result should be output to a separate vector; the increments used in storing successive elements of a given sequence (for both input and output sequences); and the increments used in storing corresponding elements in successive sequences (for both input and output).

ZFFT1M Routine Documentation

ZFFT1M (*MODE*,*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZFFT1M. On input:

- *MODE*=0 : only initializations (specific to the value of N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : forward transforms are performed. Initializations are assumed to have been performed by a prior call to ZFFT1M.
- *MODE*=1 : backward (reverse) transforms are performed. Initializations are assumed to have been performed by a prior call to ZFFT1M.
- *MODE*=−2 : (default) initializations and forward transforms are performed.
- *MODE*=2 : (default) initializations and backward transforms are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of N) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of N.

INTEGER M [Input]

On input: M is the number of sequences to be transformed.

INTEGER N [Input]

On input: N is the length of the complex sequences in X

COMPLEX*16 X(N*M) [Input/Output]

On input: X contains the M complex sequences of length N to be transformed. Element i of sequence j is stored in location $i + (j - 1) * N$ of X.
On output: X contains the transformed sequences.

COMPLEX*16 COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL ZFFT1M(0,1,N,X,COMM,INFO)
      CALL ZFFT1M(-1,2,N,X,COMM,INFO)
      DO 10 I = 1, N
         X(I,3) = X(I,1)*DCONJG(X(I,2))
         X(I,2) = DCMPLX(0.0D0,1.0D0)*X(I,2)
10    CONTINUE
      CALL ZFFT1M(1,2,N,X(1,2),COMM,INFO)
```

CFFT1M Routine Documentation

CFFT1M (*MODE*,*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CFFT1M. On input:

- *MODE*=0 : only initializations (specific to the value of N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : forward transforms are performed. Initializations are assumed to have been performed by a prior call to CFFT1M.
- *MODE*=1 : backward (reverse) transforms are performed. Initializations are assumed to have been performed by a prior call to CFFT1M.
- *MODE*=−2 : (default) initializations and forward transforms are performed.
- *MODE*=2 : (default) initializations and backward transforms are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of N) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of N.

INTEGER M [Input]

On input: M is the number of sequences to be transformed.

INTEGER N [Input]

On input: N is the length of the complex sequences in X

COMPLEX X(N*M) [Input/Output]

On input: X contains the M complex sequences of length N to be transformed. Element i of sequence j is stored in location $i + (j - 1) * N$ of X.
On output: X contains the transformed sequences.

COMPLEX COMM(5*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL CFFT1M(0,1,N,X,COMM,INFO)
      CALL CFFT1M(-1,2,N,X,COMM,INFO)
      DO 10 I = 1, N
         X(I,3) = X(I,1)*CONJG(X(I,2))
         X(I,2) = CMPLX(0.0D0,1.0D0)*X(I,2)
10    CONTINUE
      CALL CFFT1M(1,2,N,X(1,2),COMM,INFO)
```

ZFFT1MX Routine Documentation

ZFFT1MX (*MODE*,*SCALE*,*INPL*,*NSEQ*,*N*,*X*,*INCX1*,*INCX2*, [SUBROUTINE]
Y,*INCY1*,*INCY2*,*COMM*,*INFO*)

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZFFT1MX. On input:

- *MODE*=0 : only initializations (specific to the value of *N*) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT1MX.
- *MODE*=1 : a backward (reverse) transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT1MX.
- *MODE*=−2 : (default) initializations and a forward transform are performed.
- *MODE*=2 : (default) initializations and a backward transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of *N*) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of *N*.

DOUBLE PRECISION SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequences

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then *X* is overwritten by the output sequences; otherwise the output sequences are returned in *Y*.

INTEGER NSEQ [Input]

On input: *NSEQ* is the number of sequences to be transformed

INTEGER N [Input]

On input: *N* is the number of elements in each sequence to be transformed

COMPLEX*16 X(1+(N-1)*INCX1+(NSEQ-1)*INCX2) [Input/Output]

On input: *X* contains the *NSEQ* complex sequences of length *N* to be transformed; the ith element of sequence j is stored in X(1+(i-1)*INCX1+(j-1)*INCX2).

On output: if *INPL* is .TRUE. then *X* contains the transformed sequences in the same locations as on input; otherwise *X* remains unchanged.

`INTEGER INCX1` [Input]

On input: *INCX1* is the increment used to store successive elements of a given sequence in *X* (*INCX1*=1 for contiguous data).
Constraint: *INCX1* > 0.

`INTEGER INCX2` [Input]

On input: *INCX2* is the increment used to store corresponding elements of successive sequences in *X* (*INCX2*=*N* for contiguous data).
Constraint: *INCX2* > 0.

`COMPLEX*16 Y(1+(N-1)*INCY1+(NSEQ-1)*INCY2)` [Output]

On output: if *INPL* is .FALSE. then *Y* contains the transformed sequences with the ith element of sequence j stored in Y(1+(i-1)*INCY1+(j-1)*INCY2); otherwise *Y* is not referenced.

`INTEGER INCY1` [Input]

On input: *INCY1* is the increment used to store successive elements of a given sequence in *Y*. If *INPL* is .TRUE. then *INCY1* is not referenced.
Constraint: *INCY1* > 0.

`INTEGER INCY2` [Input]

On input: *INCY2* is the increment used to store corresponding elements of successive sequences in *Y* (*INCY2*=*N* for contiguous data). If *INPL* is .TRUE. then *INCY2* is not referenced.
Constraint: *INCY2* > 0.

`COMPLEX*16 COMM(3*N+100)` [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

`INTEGER INFO` [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward FFTs are performed unscaled and in-place on two
C     contiguous vectors stored in the first two columns of X.
C     Manipulations are stored in 2nd and 3rd columns of X which are
C     then transformed back.
C
        COMPLEX *16 X(N,3)
        SCALE = 1.0D0
        INPL = .TRUE.
        CALL ZFFT1MX(0,SCALE,INPL,2,N,X,1,N,DUM,1,N,COMM,INFO)
        CALL ZFFT1MX(-1,SCALE,INPL,2,N,X,1,N,DUM,1,N,COMM,INFO)
        DO 10 I = 1, N
           X(I,3) = X(I,1)*DCONJG(X(I,2))/DBLE(N)
           X(I,2) = DCMPLX(0.0D0,1.0D0)*X(I,2)/DBLE(N)
   10   CONTINUE
        CALL ZFFT1MX(1,SCALE,INPL,2,N,X(1,2),1,N,DUM,1,N,COMM,INFO)
```

CFFT1MX Routine Documentation

CFFT1MX (*MODE*,*SCALE*,*INPL*,*NSEQ*,*N*,*X*,*INCX1*,*INCX2*, [SUBROUTINE]
Y,*INCY1*,*INCY2*,*COMM*,*INFO*)

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CFFT1MX. On input:

- *MODE*=0 : only initializations (specific to the value of *N*) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward transform is performed. Initializations are assumed to have been performed by a prior call to CFFT1MX.
- *MODE*=1 : a backward (reverse) transform is performed. Initializations are assumed to have been performed by a prior call to CFFT1MX.
- *MODE*=−2 : (default) initializations and a forward transform are performed.
- *MODE*=2 : (default) initializations and a backward transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of *N*) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of *N*.

REAL SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequences

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then *X* is overwritten by the output sequences; otherwise the output sequences are returned in *Y*.

INTEGER NSEQ [Input]

On input: *NSEQ* is the number of sequences to be transformed

INTEGER N [Input]

On input: *N* is the number of elements in each sequence to be transformed

COMPLEX X(1+(N-1)*INCX1+(NSEQ-1)*INCX2) [Input/Output]

On input: *X* contains the *NSEQ* complex sequences of length *N* to be transformed; the ith element of sequence j is stored in X(1+(i-1)*INCX1+(j-1)*INCX2).

On output: if *INPL* is .TRUE. then *X* contains the transformed sequences in the same locations as on input; otherwise *X* remains unchanged.

`INTEGER INCX1` [Input]

On input: *INCX1* is the increment used to store successive elements of a given sequence in X (*INCX1*=1 for contiguous data).
Constraint: *INCX1* > 0.

`INTEGER INCX2` [Input]

On input: *INCX2* is the increment used to store corresponding elements of successive sequences in X (*INCX2*=N for contiguous data).
Constraint: *INCX2* > 0.

`COMPLEX Y(1+(N-1)*INCY1+(NSEQ-1)*INCY2)` [Output]

On output: if *INPL* is .FALSE. then Y contains the transformed sequences with the ith element of sequence j stored in Y(1+(i-1)*INCY1+(j-1)*INCY2); otherwise Y is not referenced.

`INTEGER INCY1` [Input]

On input: *INCY1* is the increment used to store successive elements of a given sequence in Y. If *INPL* is .TRUE. then *INCY1* is not referenced.
Constraint: *INCY1* > 0.

`INTEGER INCY2` [Input]

On input: *INCY2* is the increment used to store corresponding elements of successive sequences in Y (*INCY2*=N for contiguous data). If *INPL* is .TRUE. then *INCY2* is not referenced.
Constraint: *INCY2* > 0.

`COMPLEX COMM(5*N+100)` [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

`INTEGER INFO` [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward FFTs are performed unscaled and in-place on two
C     contiguous vectors stored in the first two columns of X.
C     Manipulations are stored in 2nd and 3rd columns of X which are
C     then transformed back.
C
        COMPLEX X(N,3)
        SCALE = 1.0
        INPL = .TRUE.
        CALL CFFT1MX(0,SCALE,INPL,2,N,X,1,N,DUM,1,N,COMM,INFO)
        CALL CFFT1MX(-1,SCALE,INPL,2,N,X,1,N,DUM,1,N,COMM,INFO)
        DO 10 I = 1, N
           X(I,3) = X(I,1)*CONJG(X(I,2))/REAL(N)
           X(I,2) = CMPLX(0.0D0,1.0D0)*X(I,2)/REAL(N)
   10   CONTINUE
        CALL CFFT1MX(1,SCALE,INPL,2,N,X(1,2),1,N,DUM,1,N,COMM,INFO)
```

5.2.3 2D FFT of two-dimensional arrays of data

The routines documented here compute the two-dimensional discrete Fourier transforms (DFT) of a two-dimensional array of complex numbers in either single or double precision arithmetic. The 2D DFT is computed using a highly-efficient FFT algorithm.
There are two sets of interfaces available: simple drivers and expert drivers. The simple drivers perform in-place transforms on data held contiguously in memory using a fixed scaling factor; these are simpler to use and are sufficient for many problems. The expert drivers offer greater flexibility by including a number of additional arguments. These allow you to control: the scaling factor applied; whether the result should be output to a separate array; the increments used in storing successive elements in each dimension (for both input and output); and the facility to not perform a final transposition. This final facility is useful for those cases where a forward and backward transform are to be applied with some data manipulations in between; here two whole transpositions can be saved.

ZFFT2D Routine Documentation

ZFFT2D (*MODE*,*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the direction of transform to be performed by ZFFT2D.

On input:

- *MODE*=−1 : forward 2D transform is performed.
- *MODE*=1 : backward (reverse) 2D transform is performed.

INTEGER M [Input]

On input: M is the number of rows in the 2D array of data to be transformed. If X is declared as a 2D array then M is the first dimension of X.

INTEGER N [Input]

On input: N is the number of columns in the 2D array of data to be transformed. If X is declared as a 2D array then M is the second dimension of X.

COMPLEX*16 X(M*N) [Input/Output]

On input: X contains the M by N complex 2D array to be transformed. Element ij is stored in location $i + (j - 1) * M$ of X.

On output: X contains the transformed sequence.

COMPLEX*16 COMM(M*N+3*(M+N)+100) [Input/Output]

COMM is a communication array used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL ZFFT2D(-1,M,N,X,COMM,INFO)
      DO 20 J = 1, N
         DO 10 I = 1, MIN(J-1,M)
            X(I,J) = 0.5D0*(X(I,J) + X(J,I))
            X(J,I) = DCONJG(X(I,J))
10       CONTINUE
20    CONTINUE
      CALL ZFFT2D(1,M,N,X,COMM,INFO)
```

CFFT2D Routine Documentation

CFFT2D (*MODE*,*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the direction of transform to be performed by CFFT2D.

On input:

- *MODE*=−1 : a forward 2D transform is performed.
- *MODE*=1 : a backward (reverse) 2D transform is performed.

INTEGER M [Input]

On input: M is the number of rows in the 2D array of data to be transformed. If X is declared as a 2D array then M is the first dimension of X.

INTEGER N [Input]

On input: N is the number of columns in the 2D array of data to be transformed. If X is declared as a 2D array then M is the second dimension of X.

COMPLEX X(M*N) [Input/Output]

On input: X contains the M by N complex 2D array to be transformed. Element ij is stored in location $i + (j - 1) * M$ of X.

On output: X contains the transformed sequence.

COMPLEX COMM(M*N+5*(M+N)) [Input/Output]

COMM is a communication array used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL CFFT2D(-1,M,N,X,COMM,INFO)
      DO 20 J = 1, N
         DO 10 I = 1, MIN(J-1,M)
            X(I,J) = 0.5D0*(X(I,J) + X(J,I))
            X(J,I) = CONJG(X(I,J))
10       CONTINUE
20    CONTINUE
      CALL CFFT2D(1,M,N,X,COMM,INFO)
```

ZFFT2DX Routine Documentation

ZFFT2DX (*MODE*,*SCALE*,*LTRANS*,*INPL*,*M*,*N*,*X*,*INCX1*,*INCX2*, *Y*,*INCY1*,*INCY2*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZFFT2DX. On input:

- *MODE*=0 : only initializations (specific to the value of N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward 2D transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT2DX.
- *MODE*=1 : a backward (reverse) 2D transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT2DX.
- *MODE*=−2 : (default) initializations and a forward 2D transform are performed.
- *MODE*=2 : (default) initializations and a backward 2D transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the values of N and M) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the values of N and M.

DOUBLE PRECISION SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequences

LOGICAL LTRANS [Input]

On input: if *LTRANS* is .TRUE. then a normal final transposition is performed internally to return transformed data consistent with the values for arguments *INPL*, *INCX1*, *INCX2*, *INCY1* and *INCY2*. If *LTRANS* is .FALSE. then the final transposition is not performed explicitly; the storage format on output is determined by whether the output data is stored contiguously or not – please see the output specifications for X and Y for details.

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then X is overwritten by the output sequences; otherwise the output sequences are returned in Y.

INTEGER M [Input]

On input: M is the first dimension of the 2D transform.

INTEGER N [Input]

On input: N is the second dimension of the 2D transform.

`COMPLEX*16 X(1+(M-1)*INCX1+(N-1)*INCX2)` [Input/Output]

On input: *X* contains the *M* by *N* complex 2D data array to be transformed; the (ij)th element is stored in X(1+(i-1)*INCX1+(j-1)*INCX2).

On output: if *INPL* is .TRUE. then *X* contains the transformed data, either in the same locations as on input when *LTRANS*=.TRUE.; in locations X((i-1)*N+j) when *LTRANS*=.FALSE., *INCX1*=1 and *INCX2*=M; and otherwise in the same locations as on input. If *INPL* is .FALSE. *X* remains unchanged.

`INTEGER INCX1` [Input]

On input: *INCX1* is the increment used to store, in *X*, successive elements in the first dimension (*INCX1*=1 for contiguous data).

Constraint: *INCX1* > 0.

`INTEGER INCX2` [Input]

On input: *INCX2* is the increment used to store, in *X*, successive elements in the second dimension (*INCX2*=M for contiguous data).

Constraint: *INCX2* > 0;

INCX2 > (M-1)*INCX1 if *N* > 1.

`COMPLEX*16 Y(1+(M-1)*INCY1+(N-1)*INCY2)` [Output]

On output: if *INPL* is .FALSE. then *Y* contains the transformed data. If *LTRANS*=.TRUE. then the (ij)th data element is stored in Y(1+(i-1)*INCY1+(j-1)*INCY2); if *LTRANS*=.FALSE., *INCY1*=1 and *INCY2*=N then the (ij)th data element is stored in Y((i-1)*N+j); and otherwise the (ij)th element is stored in Y(1+(i-1)*INCY1+(j-1)*INCY2). If *INPL* is .TRUE. then *Y* is not referenced.

`INTEGER INCY1` [Input]

On input: *INCY1* is the increment used to store successive elements in the first dimension in *Y* (*INCY1*=1 for contiguous data). If *INPL* is .TRUE. then *INCY1* is not referenced.

Constraint: *INCY1* > 0.

`INTEGER INCY2` [Input]

On input: *INCY2* is the increment used to store successive elements in the second dimension in *Y* (for contiguous data, *INCY2*=*M* when *LTRANS* is .TRUE. or *INCY2*=*N* when *LTRANS* is .FALSE.). If *INPL* is .TRUE. then *INCY2* is not referenced.

Constraints: *INCY2* > 0;

INCY2 > (M-1)*INCY1 if *N* > 1 and *LTRANS* is .TRUE.;

INCY2 = N if *M* > 1 and *LTRANS* is .FALSE..

`COMPLEX*16 COMM(M*N+3*M+3*N+200)` [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same dimensions *M* and *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward 2D FFT is performed unscaled, without final transpose
C     and out-of-place on data stored in array X and output to Y.
C     Manipulations are stored in vector Y which is then transformed
C     back, with scaling, into the first M rows of X.
C
        COMPLEX *16 X(M,N), Y(N,M)
        SCALE = 1.0D0
        INPL = .FALSE.
        LTRANS = .FALSE.
        CALL ZFFT2DX(0,SCALE,LTRANS,INPL,M,N,X,1,M,Y,1,N,COMM,INFO)
        CALL ZFFT2DX(-1,SCALE,LTRANS,INPL,M,N,X,1,M,Y,1,N,COMM,INFO)
        DO 20 I = M
           DO 10 J = 1, N
              Y(J,I) = 0.5D0*Y(J,I)*EXP(0.001D0*(I+J-2))
   10      CONTINUE
   20   CONTINUE
        SCALE = 1.0D0/DBLE(M*N)
        CALL ZFFT2DX(1,SCALE,LTRANS,INPL,N,M,Y,1,N,X,1,M,COMM,INFO)
```

CFFT2DX Routine Documentation

CFFT2DX (*MODE*,*SCALE*,*LTRANS*,*INPL*,*M*,*N*,*X*,*INCX1*,*INCX2*, *Y*,*INCY1*,*INCY2*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CFFT2DX. On input:

- *MODE*=0 : only initializations (specific to the value of N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward 2D transform is performed. Initializations are assumed to have been performed by a prior call to CFFT2DX.
- *MODE*=1 : a backward (reverse) 2D transform is performed. Initializations are assumed to have been performed by a prior call to CFFT2DX.
- *MODE*=−2 : (default) initializations and a forward 2D transform are performed.
- *MODE*=2 : (default) initializations and a backward 2D transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the values of N and M) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the values of N and M.

REAL SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequences

LOGICAL LTRANS [Input]

On input: if *LTRANS* is .TRUE. then a normal final transposition is performed internally to return transformed data consistent with the values for arguments *INPL*, *INCX1*, *INCX2*, *INCY1* and *INCY2*. If *LTRANS* is .FALSE. then the final transposition is not performed explicitly; the storage format on output is determined by whether the output data is stored contiguously or not – please see the output specifications for X and Y for details.

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then X is overwritten by the output sequences; otherwise the output sequences are returned in Y.

INTEGER M [Input]

On input: M is the first dimension of the 2D transform.

INTEGER N [Input]

On input: N is the second dimension of the 2D transform.

`COMPLEX X(1+(M-1)*INCX1+(N-1)*INCX2)` [Input/Output]

On input: *X* contains the *M* by *N* complex 2D data array to be transformed; the (ij)th element is stored in X(1+(i-1)*INCX1+(j-1)*INCX2).

On output: if *INPL* is .TRUE. then *X* contains the transformed data, either in the same locations as on input when *LTRANS*=.TRUE.; in locations X((i-1)*N+j) when *LTRANS*=.FALSE., *INCX1*=1 and *INCX2*=M; and otherwise in the same locations as on input. If *INPL* is .FALSE. *X* remains unchanged.

`INTEGER INCX1` [Input]

On input: *INCX1* is the increment used to store, in *X*, successive elements in the first dimension (*INCX1*=1 for contiguous data).

Constraint: *INCX1* > 0.

`INTEGER INCX2` [Input]

On input: *INCX2* is the increment used to store, in *X*, successive elements in the second dimension (*INCX2*=M for contiguous data).

Constraint: *INCX2* > 0;

INCX2 > (M-1)*INCX1 if *N* > 1.

`COMPLEX Y(1+(M-1)*INCY1+(N-1)*INCY2)` [Output]

On output: if *INPL* is .FALSE. then *Y* contains the transformed data. If *LTRANS*=.TRUE. then the (ij)th data element is stored in Y(1+(i-1)*INCY1+(j-1)*INCY2); if *LTRANS*=.FALSE., *INCY1*=1 and *INCY2*=N then the (ij)th data element is stored in Y((i-1)*N+j); and otherwise the (ij)th element is stored in Y(1+(i-1)*INCY1+(j-1)*INCY2). If *INPL* is .TRUE. then *Y* is not referenced.

`INTEGER INCY1` [Input]

On input: *INCY1* is the increment used to store successive elements in the first dimension in *Y* (*INCY1*=1 for contiguous data). If *INPL* is .TRUE. then *INCY1* is not referenced.

Constraint: *INCY1* > 0.

`INTEGER INCY2` [Input]

On input: *INCY2* is the increment used to store successive elements in the second dimension in *Y* (for contiguous data, *INCY2*=*M* when *LTRANS* is .TRUE. or *INCY2*=*N* when *LTRANS* is .FALSE.). If *INPL* is .TRUE. then *INCY2* is not referenced.

Constraints: *INCY2* > 0;

INCY2 > (M-1)*INCY1 if *N* > 1 and *LTRANS* is .TRUE.;

INCY2 = N if *M* > 1 and *LTRANS* is .FALSE..

`COMPLEX COMM(M*N+5*M+5*N+200)` [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same dimensions *M* and *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward 2D FFT is performed unscaled, without final transpose
C     and out-of-place on data stored in array X and output to Y.
C     Manipulations are stored in vector Y which is then transformed
C     back, with scaling, into the first M rows of X.
C
        COMPLEX X(M,N), Y(N,M)
        SCALE = 1.0
        INPL = .FALSE.
        LTRANS = .FALSE.
        CALL CFFT2DX(0,SCALE,LTRANS,INPL,M,N,X,1,M,Y,1,N,COMM,INFO)
        CALL CFFT2DX(-1,SCALE,LTRANS,INPL,M,N,X,1,M,Y,1,N,COMM,INFO)
        DO 20 I = M
           DO 10 J = 1, N
              Y(J,I) = 0.5*Y(J,I)*EXP(-0.001*REAL(I+J-2))
              IY = IY + 1
   10      CONTINUE
   20   CONTINUE
        SCALE = 1.0/REAL(M*N)
        CALL CFFT2DX(1,SCALE,LTRANS,INPL,N,M,Y,1,N,X,1,M,COMM,INFO)
```

5.2.4 3D FFT of three-dimensional arrays of data

The routines documented here compute the three-dimensional discrete Fourier transforms (DFT) of a three-dimensional array of complex numbers in either single or double precision arithmetic. The 3D DFT is computed using a highly-efficient FFT algorithm.

ZFFT3D Routine Documentation

ZFFT3D (*MODE*,*L*,*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the direction of transform to be performed by ZFFT3D.

On input:

- *MODE*=−1 : forward 3D transform is performed.
- *MODE*=1 : backward (reverse) 3D transform is performed.

INTEGER L [Input]

On input: the length of the first dimension of the 3D array of data to be transformed. If X is declared as a 3D array then L is the first dimension of X.

INTEGER M [Input]

On input: the length of the second dimension of the 3D array of data to be transformed. If X is declared as a 3D array then M is the second dimension of X.

INTEGER N [Input]

On input: the length of the third dimension of the 3D array of data to be transformed. If X is declared as a 3D array then N is the third dimension of X.

COMPLEX*16 X(L*M*N) [Input/Output]

On input: X contains the L by M by N complex 3D array to be transformed. Element ijk is stored in location $i + (j-1) * L + (k-1) * L * M$ of X.

On output: X contains the transformed sequence.

COMPLEX*16 COMM(L*M*N+3*(L+M+N)) [Input/Output]

COMM is a communication array used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL ZFFT3D(-1,L,M,N,X,COMM,INFO)
      DO 30 K = 1, N
        DO 20 J = 1, M
          DO 10 I = 1, L
            X(I,J) = X(I,J)*EXP(-0.001D0*DBLE(I+J+K))
10        CONTINUE
20      CONTINUE
30    CONTINUE
      CALL ZFFT3D(1,L,M,N,X,COMM,INFO)
```

CFFT3D Routine Documentation

CFFT3D (*MODE*,*L*,*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the direction of transform to be performed by CFFT3D.

On input:

- *MODE*=−1 : forward 3D transform is performed.
- *MODE*=1 : backward (reverse) 3D transform is performed.

INTEGER L [Input]

On input: the length of the first dimension of the 3D array of data to be transformed. If X is declared as a 3D array then L is the first dimension of X.

INTEGER M [Input]

On input: the length of the second dimension of the 3D array of data to be transformed. If X is declared as a 3D array then M is the second dimension of X.

INTEGER N [Input]

On input: the length of the third dimension of the 3D array of data to be transformed. If X is declared as a 3D array then N is the third dimension of X.

COMPLEX X(L*M*N) [Input/Output]

On input: X contains the L by M by N complex 3D array to be transformed. Element ijk is stored in location $i + (j-1) * L + (k-1) * L * M$ of X.

On output: X contains the transformed sequence.

COMPLEX COMM(5*(L+M+N)+4) [Input/Output]

COMM is a communication array used as temporary store. Some further workspace will be allocated internally; the amount of allocated memory requested will be MAX(L*N+N, L*M + L + M, N).

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL CFFT3D(-1,L,M,N,X,COMM,INFO)
      DO 30 K = 1, N
        DO 20 J = 1, M
          DO 10 I = 1, L
            X(I,J) = X(I,J)*EXP(-0.001D0*REAL(I+J+K))
10        CONTINUE
20      CONTINUE
30    CONTINUE
      CALL CFFT3D(1,L,M,N,X,COMM,INFO)
```

ZFFT3DX Routine Documentation

ZFFT3DX (*MODE*,*SCALE*,*LTRANS*,*INPL*,*L*,*M*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZFFT3DX. On input:

- *MODE*=0 : only initializations (specific to the values of L, M and N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward 3D transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT3DX.
- *MODE*=1 : a backward (reverse) 3D transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT3DX.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the values of L, M and M) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the values of L, M and N.

DOUBLE PRECISION SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequences

LOGICAL LTRANS [Input]

On input: if *LTRANS* is .TRUE. then a normal final transposition is performed internally to return transformed data using the same storage format as the input data. If *LTRANS* is .FALSE. then the final transposition is not performed and transformed data is stored, in X or Y, in transposed form.

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then X is overwritten by the output sequences; otherwise the output sequences are returned in Y.

INTEGER L [Input]

On input: L is the first dimension of the 3D transform.

INTEGER M [Input]

On input: M is the second dimension of the 3D transform.

INTEGER N [Input]

On input: N is the third dimension of the 3D transform.

COMPLEX*16 X(L*M*N) [Input/Output]

On input: X contains the L by M by N complex 3D data array to be transformed; the (ijk)th element is stored in X(i+(j-1)*L+(k-1)*L*M).

On output: if *INPL* is .TRUE. then X contains the transformed data, either in the same locations as on input when *LTRANS*=.TRUE.; or in locations X(k+(j-1)*N+(i-1)*N*M) when *LTRANS*=.FALSE. If *INPL* is .FALSE. X remains unchanged.

`COMPLEX*16 Y(L*M*N)` [Output]

On output: if *INPL* is .FALSE. then *Y* contains the three-dimensional transformed data. If *LTRANS*=.TRUE. then the (ijk)th data element is stored in Y(i+(j-1)*L+(k-1)*L*M); otherwise, the (ijk)th data element is stored in Y(k+(j-1)*N+(i-1)*N*M). If *INPL* is .TRUE. then *Y* is not referenced.

`COMPLEX*16 COMM(L*M*N+3*(L+M+N)+300)` [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence dimensions. The remainder is used as temporary store.

`INTEGER INFO` [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward 3D FFT is performed unscaled, without final transpose
C     and out-of-place on data stored in array X and output to Y.
C     Manipulations are stored in vector Y which is then transformed
C     back, with scaling, into the first M rows of X.
C
        COMPLEX *16 X(L*M*N), Y(L*M*N)
        SCALE = 1.0D0
        INPL = .FALSE.
        LTRANS = .FALSE.
        CALL ZFFT3DX(0,SCALE,LTRANS,INPL,L,M,N,X,Y,COMM,INFO)
        CALL ZFFT3DX(-1,SCALE,LTRANS,INPL,L,M,N,X,Y,COMM,INFO)
        IY = 1
        DO 20 I = 1, L
           DO 40 J = 1, M
              DO 10 K = 1, N
                 Y(IY) = Y(IY)*EXP(-0.001D0*DBLE(I+J+K-3))
                 IY = IY + 1
   10         CONTINUE
   20      CONTINUE
           SCALE = 1.0D0/DBLE(L*M*N)
           CALL ZFFT3DX(1,SCALE,LTRANS,INPL,N,M,L,Y,X,COMM,INFO)
```

CFFT3DX Routine Documentation

CFFT3DX (*MODE*,*SCALE*,*LTRANS*,*INPL*,*L*,*M*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CFFT3DX.
On input:

- *MODE*=0 : only initializations (specific to the values of L, M and N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward 3D transform is performed. Initializations are assumed to have been performed by a prior call to CFFT3DX.
- *MODE*=1 : a backward (reverse) 3D transform is performed. Initializations are assumed to have been performed by a prior call to CFFT3DX.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the values of L, M and M) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the values of L, M and N.

REAL SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequences

LOGICAL LTRANS [Input]

On input: if *LTRANS* is .TRUE. then a normal final transposition is performed internally to return transformed data using the same storage format as the input data. If *LTRANS* is .FALSE. then the final transposition is not performed and transformed data is stored, in X or Y, in transposed form.

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then X is overwritten by the output sequences; otherwise the output sequences are returned in Y.

INTEGER L [Input]

On input: L is the first dimension of the 3D transform.

INTEGER M [Input]

On input: M is the second dimension of the 3D transform.

INTEGER N [Input]

On input: N is the third dimension of the 3D transform.

COMPLEX X(L*M*N) [Input/Output]

On input: X contains the L by M by N complex 3D data array to be transformed; the (ijk)th element is stored in X(i+(j-1)*L+(k-1)*L*M).
On output: if *INPL* is .TRUE. then X contains the transformed data, either in the same locations as on input when *LTRANS*=.TRUE.; or in locations X(k+(j-1)*N+(i-1)*N*M) when *LTRANS*=.FALSE. If *INPL* is .FALSE. X remains unchanged.

`COMPLEX Y(L*M*N)` [Output]

On output: if *INPL* is .FALSE. then *Y* contains the three-dimensional transformed data. If *LTRANS*=.TRUE. then the (ijk)th data element is stored in Y(i+(j-1)*L+(k-1)*L*M); otherwise, the (ijk)th data element is stored in Y(k+(j-1)*N+(k-1)*N*M). If *INPL* is .TRUE. then *Y* is not referenced.

`COMPLEX COMM(*)` [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence dimensions. The remainder is used as temporary store. The amount of store required depends on the values of the arguments *L*, *M*, *N* and *LTRANS*. If *LTRANS*=.TRUE. then for a genuine 3D transform (all of *L*, *M*, *N* greater than 1) the dimension of *COMM* need only be 5*(L+M+N) + 150; in this case some further workspace will be allocated internally; the amount of allocated memory requested will be MAX(L*N+N, L*M + L + M, N). If *LTRANS*=.FALSE. then for a genuine 3D transform the workspace requirement is considerably more to allow for the storage of an intermediate 3D transposed array; in this case the dimension of *COMM* must be at least L*M*N+5*(L+M+N)+150. It is recommended that the appropriate 1D or 2D FFT routine be called when at least one of *L*, *M* or *N* is 1.

`INTEGER INFO` [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward 3D FFT is performed unscaled, without final transpose
C     and out-of-place on data stored in array X and output to Y.
C     Manipulations are stored in vector Y which is then transformed
C     back, with scaling, into the first M rows of X.
C
        SCALE = 1.0
        INPL = .FALSE.
        LTRANS = .FALSE.
        CALL CFFT3DX(0,SCALE,LTRANS,INPL,L,M,N,X,Y,COMM,INFO)
        CALL CFFT3DX(-1,SCALE,LTRANS,INPL,L,M,N,X,Y,COMM,INFO)
        IY = 1
        DO 20 I = 1, L
           DO 40 J = 1, M
              DO 10 K = 1, N
                 Y(IY) = Y(IY)*EXP(-0.001*REAL(I+J+K-3))
                 IY = IY + 1
   10      CONTINUE
   20   CONTINUE
        SCALE = 1.0/REAL(L*M*N)
        CALL CFFT3DX(1,SCALE,LTRANS,INPL,N,M,L,Y,X,COMM,INFO)
```

ZFFT3DY Routine Documentation

ZFFT3DY (*MODE*,*SCALE*,*INPL*,*L*,*M*,*N*,*X*,*INCX1*,*INCX2*,*INCX3*,*Y*, *INCY1*,*INCY2*,*INCY3*,*COMM*,*LCOMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZFFT3DY. On input:

- *MODE*=0 : only initializations (specific to the values of L, M and N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward 3D transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT3DY.
- *MODE*=1 : a backward (reverse) 3D transform is performed. Initializations are assumed to have been performed by a prior call to ZFFT3DY.
- *MODE*=−2 : (default) initializations and a forward 3D transform are performed.
- *MODE*=2 : (default) initializations and a backward 3D transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the values of L, M and M) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the values of L, M and N.

REAL SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequences

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then X is overwritten by the output sequences; otherwise the output sequences are returned in Y.

INTEGER L [Input]

On input: L is the first dimension of the 3D transform.

INTEGER M [Input]

On input: M is the second dimension of the 3D transform.

INTEGER N [Input]

On input: N is the third dimension of the 3D transform.

COMPLEX*16 X(*) [Input/Output]

On input: X contains the L by M by N complex 3D data array to be transformed; the (ijk)th element is stored in X(1+(i-1)**INCX1*+(j-1)**INCX2*+(k-1)**INCX3*).

On output: if *INPL* is .TRUE. then X contains the transformed data in the same locations as on input. If *INPL* is .FALSE. X remains unchanged.

`INTEGER INCX1` [Input]

On input: *INCX1* is the step in index of X between successive data elements in the first dimension of the 3D data. Usually *INCX1*=1 so that succesive elements in the first dimension are stored contiguously.
Constraint: *INCX1* > 0.

`INTEGER INCX2` [Input]

On input: *INCX2* is the step in index of X between successive data elements in the second dimension of the 3D data. For completely contiguous data (no gaps in X) *INCX2* should be set to L.
Constraint: *INCX2* > 0;
INCX2 > (L-1)*INCX1 if max(M,N) > 1.

`INTEGER INCX3` [Input]

On input: *INCX3* is the step in index of X between successive data elements in the third dimension of the 3D data. For completely contiguous data (no gaps in X) *INCX3* should be set to $L*M$.
Constraint: *INCX3* > 0;
INCX3 > (L-1)**INCX1*+(M-1)**INCX2* if N > 1.

`COMPLEX*16 Y(*)` [Output]

On output: if *INPL* is .FALSE. then Y contains the three-dimensional transformed data. If *LTRANS*=.TRUE. then the the (ijk)th element is stored in Y(1+(i-1)**INCY1*+(j-1)**INCY2*+(k-1)**INCY3*).
If *INPL* is .TRUE. then Y is not referenced.

`INTEGER INCY1` [Input]

On input: if *INPL* is .FALSE. then *INCY1* is the step in index of Y between successive data elements in the first dimension of the 3D transformed data. Usually *INCY1*=1 so that succesive elements in the first dimension are stored contiguously.
If *INPL* is .TRUE. then *INCY1* is not referenced. Constraint: If *INPL* is .FALSE. then *INCY1* > 0.

`INTEGER INCY2` [Input]

On input: if *INPL* is .FALSE. then *INCY2* is the step in index of Y between successive data elements in the second dimension of the 3D transformed data. For completely contiguous data (no gaps in Y) *INCY2* should be set to L.
Constraint: *INCY2* > 0 if *INPL* is .FALSE.;
INCY2 > (L-1)**INCY1*, if *INPL* is .FALSE. and max(M,N) > 1.

`INTEGER INCY3` [Input]

On input: if *INPL* is .FALSE. then *INCY3* is the step in index of Y between successive data elements in the third dimension of the 3D transformed data. For completely contiguous data (no gaps in Y) *INCY3* should be set to $L*M$.
Constraint: *INCY3* > 0 if *INPL* is .FALSE.;
INCY3 > (L-1)**INCY1*+(M-1)**INCY2*, if *INPL* is .FALSE. and N > 1.

COMPLEX*16 COMM(*LCOMM*) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence dimensions. The remainder is used as temporary store; if this is not sufficient for the requirements of the routine then temporary storage space will be dynamically allocated internally.

INTEGER LCOMM [Input]

On input: *LCOMM* is the length of the communication array *COMM*. The amount of internal dynamic allocation of temporary storage can be reduced significantly by declaring *COMM* to be of length at least L*M*N + 4*(L+M+N) + 300.

Constraint: *LCOMM* > 3*(L+M+N) + 150.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward 3D FFT is performed unscaled and in-place, on the leading
C     10x10x10 submatrix of a larger 100x100x100 array of data.
C     The result is transformed back with scaling.
C
        SCALE = 1.0D0
        INPL = .TRUE.
        L = 10
        M = 10
        N = 10
        LCOMM = 2000000
        CALL ZFFT3DY(0,SCALE,INPL,L,M,N,X,1,100,10000,Y,1,1,1,
     *               COMM,LCOMM,INFO)
        CALL ZFFT3DY(-1,SCALE,INPL,L,M,N,X,1,100,10000,Y,1,1,1,
     *               COMM,LCOMM,INFO)
        IY = 1
        DO 20 I = 1, L
           DO 40 J = 1, M
              DO 10 K = 1, N
                 X(I,J,K) = X(I,J,K)*EXP(-1.0D-3*DBLE(I+J+K-3))
   10      CONTINUE
   20   CONTINUE
        SCALE = 1.0/DBLE(L*M*N)
        CALL ZFFT3DY(1,SCALE,INPL,L,M,N,X,1,100,10000,Y,1,1,1,
     *               COMM,LCOMM,INFO)
```

CFFT3DY Routine Documentation

CFFT3DY (*MODE*,*SCALE*,*INPL*,*L*,*M*,*N*,*X*,*INCX1*,*INCX2*,*INCX3*,*Y*, *INCY1*,*INCY2*,*INCY3*,*COMM*,*LCOMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CFFT3DY. On input:

- *MODE*=0 : only initializations (specific to the values of L, M and N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=−1 or 1.
- *MODE*=−1 : a forward 3D transform is performed. Initializations are assumed to have been performed by a prior call to CFFT3DY.
- *MODE*=1 : a backward (reverse) 3D transform is performed. Initializations are assumed to have been performed by a prior call to CFFT3DY.
- *MODE*=−2 : (default) initializations and a forward 3D transform are performed.
- *MODE*=2 : (default) initializations and a backward 3D transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the values of L, M and M) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the values of L, M and N.

REAL SCALE [Input]

On input: *SCALE* is the scaling factor to apply to the output sequences

LOGICAL INPL [Input]

On input: if *INPL* is .TRUE. then X is overwritten by the output sequences; otherwise the output sequences are returned in Y.

INTEGER L [Input]

On input: L is the first dimension of the 3D transform.

INTEGER M [Input]

On input: M is the second dimension of the 3D transform.

INTEGER N [Input]

On input: N is the third dimension of the 3D transform.

COMPLEX X(*) [Input/Output]

On input: X contains the L by M by N complex 3D data array to be transformed; the (ijk)th element is stored in X(1+(i-1)**INCX1*+(j-1)**INCX2*+(k-1)**INCX3*).

On output: if *INPL* is .TRUE. then X contains the transformed data in the same locations as on input. If *INPL* is .FALSE. X remains unchanged.

`INTEGER INCX1` [Input]

On input: *INCX1* is the step in index of X between successive data elements in the first dimension of the 3D data. Usually *INCX1*=1 so that succesive elements in the first dimension are stored contiguously.
Constraint: *INCX1* > 0.

`INTEGER INCX2` [Input]

On input: *INCX2* is the step in index of X between successive data elements in the second dimension of the 3D data. For completely contiguous data (no gaps in X) *INCX2* should be set to L.
Constraint: *INCX2* > 0;
INCX2 > (L-1)*INCX1 if max(M,N) > 1.

`INTEGER INCX3` [Input]

On input: *INCX3* is the step in index of X between successive data elements in the third dimension of the 3D data. For completely contiguous data (no gaps in X) *INCX3* should be set to $L*M$.
Constraint: *INCX3* > 0;
INCX3 > (L-1)**INCX1*+(M-1)**INCX2* if N > 1.

`COMPLEX Y(*)` [Output]

On output: if *INPL* is .FALSE. then Y contains the three-dimensional transformed data. If *LTRANS*=.TRUE. then the the (ijk)th element is stored in Y(1+(i-1)**INCY1*+(j-1)**INCY2*+(k-1)**INCY3*).
If *INPL* is .TRUE. then Y is not referenced.

`INTEGER INCY1` [Input]

On input: if *INPL* is .FALSE. then *INCY1* is the step in index of Y between successive data elements in the first dimension of the 3D transformed data. Usually *INCY1*=1 so that succesive elements in the first dimension are stored contiguously.
If *INPL* is .TRUE. then *INCY1* is not referenced. Constraint: If *INPL* is .FALSE. then *INCY1* > 0.

`INTEGER INCY2` [Input]

On input: if *INPL* is .FALSE. then *INCY2* is the step in index of Y between successive data elements in the second dimension of the 3D transformed data. For completely contiguous data (no gaps in Y) *INCY2* should be set to L.
Constraint: *INCY2* > 0 if *INPL* is .FALSE.;
INCY2 > (L-1)**INCY1*, if *INPL* is .FALSE. and max(M,N) > 1.

`INTEGER INCY3` [Input]

On input: if *INPL* is .FALSE. then *INCY3* is the step in index of Y between successive data elements in the third dimension of the 3D transformed data. For completely contiguous data (no gaps in Y) *INCY3* should be set to $L*M$.
Constraint: *INCY3* > 0 if *INPL* is .FALSE.;
INCY3 > (L-1)**INCY1*+(M-1)**INCY2*, if *INPL* is .FALSE. and N > 1.

`COMPLEX COMM(`*`LCOMM`*`)` [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence dimensions. The remainder is used as temporary store; if this is not sufficient for the requirements of the routine then temporary storage space will be dynamically allocated internally.

`INTEGER LCOMM` [Input]

On input: *LCOMM* is the length of the communication array *COMM*. The amount of internal dynamic allocation of temporary storage is dependent on the values of the increment arguments for arrays *X* and *Y*. The amount is minimized when the increments for the output array are 1, *L* and *L*M* respectively, since this represents a contiguous array in which calculations can be performedd in-place.

Constraint: If min(*L*,*M*,*N*) > 1, *LCOMM* >= 5*(L+M+N) + 150 (otherwise see see [CFFT2DX], page 56).

`INTEGER INFO` [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
C     Forward 3D FFT is performed unscaled and in-place, on the leading
C     10x10x10 submatrix of a larger 100x100x100 array of data.
C     The result is transformed back with scaling.
C
        SCALE = 1.0
        INPL = .TRUE.
        L = 10
        M = 10
        N = 10
        LCOMM = 2000000
        CALL CFFT3DY(0,SCALE,INPL,L,M,N,X,1,100,10000,Y,1,1,1,
     *               COMM,LCOMM,INFO)
        CALL CFFT3DY(-1,SCALE,INPL,L,M,N,X,1,100,10000,Y,1,1,1,
     *               COMM,LCOMM,INFO)
        IY = 1
        DO 20 I = 1, L
           DO 40 J = 1, M
              DO 10 K = 1, N
                 X(I,J,K) = X(I,J,K)*EXP(-0.001*REAL(I+J+K-3))
   10      CONTINUE
   20   CONTINUE
        SCALE = 1.0/REAL(L*M*N)
        CALL CFFT3DY(1,SCALE,INPL,L,M,N,X,1,100,10000,Y,1,1,1,
     *               COMM,LCOMM,INFO)
```

5.3 FFTs on Real and Hermitian Data Sequences

The routines documented here compute discrete Fourier transforms (DFTs) of sequences of real numbers or of Hermitian sequences in either single or double precision arithmetic. The DFTs are computed using a highly-efficient FFT algorithm. Hermitian sequences are represented in one of the two formats that is described in Section 5.1 [Introduction to FFTs], page 28. The DFT of a real sequence results in a Hermitian sequence; the DFT of a Hermitian sequence is a real sequence.

5.3.1 1D Real-To-Complex FFT (Hermitian-Packed Storage)

DZFFT Routine Documentation

DZFFT (*MODE*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by DZFFT.
On input:

- *MODE*=0 : only default initializations (specific to *N*) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real transform is performed. Initializations are assumed to have been performed by a prior call to DZFFT.
- *MODE*=2 : (default) initializations and a real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER N [Input]

On input: *N* is the length of the real sequence *X*

DOUBLE PRECISION X(N) [Input/Output]

On input: *X* contains the real sequence of length *N* to be transformed.
On output: *X* contains the transformed Hermitian sequence.

DOUBLE PRECISION COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL DZFFT(0,N,X,COMM,INFO)
      CALL DZFFT(1,N,X,COMM,INFO)
      DO 10 I = N/2+2, N
         X(I) = -X(I)
10    CONTINUE
      CALL ZDFFT(2,N,X,COMM,INFO)
```

SCFFT Routine Documentation

SCFFT (*MODE*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by SCFFT.
On input:

- *MODE*=0 : only default initializations (specific to *N*) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real transform is performed. Initializations are assumed to have been performed by a prior call to SCFFT.
- *MODE*=2 : (default) initializations and a real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER N [Input]

On input: *N* is the length of the real sequence *X*

REAL X(N) [Input/Output]

On input: *X* contains the real sequence of length *N* to be transformed.
On output: *X* contains the transformed Hermitian sequence.

REAL COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL SCFFT(0,N,X,COMM,INFO)
      CALL SCFFT(1,N,X,COMM,INFO)
      DO 10 I = N/2+2, N
         X(I) = -X(I)
10    CONTINUE
      CALL CSFFT(2,N,X,COMM,INFO)
```

5.3.2 Multiple 1D Real-To-Complex FFT (Hermitian-Packed Storage)

DZFFTM Routine Documentation

DZFFTM (*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER M [Input]

On input: M is the number of sequences to be transformed.

INTEGER N [Input]

On input: N is the length of the real sequences in X

DOUBLE PRECISION X(N*M) [Input/Output]

On input: X contains the M real sequences of length N to be transformed. Element i of sequence j is stored in location $i + (j-1) * N$ of X.
On output: X contains the transformed Hermitian sequences.

DOUBLE PRECISION COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL DZFFTM(1,N,X,COMM,INFO)
      CALL DZFFTM(2,N,X,COMM,INFO)
      DO 10 I = 1, N
         X(I,3) = X(I,1)*X(N-I+1,2)
10    CONTINUE
      CALL ZDFFTM(2,N,X(1,3),COMM,INFO)
```

SCFFTM Routine Documentation

SCFFTM (*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER M [Input]

On input: M is the number of sequences to be transformed.

INTEGER N [Input]

On input: N is the length of the real sequences in X

REAL X(N*M) [Input/Output]

On input: X contains the M real sequences of length N to be transformed. Element i of sequence j is stored in location $i + (j - 1) * N$ of X.
On output: X contains the transformed Hermitian sequences.

REAL COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If $INFO = -$i on exit, the i-th argument had an illegal value.

```
      CALL SCFFTM(1,N,X,COMM,INFO)
      CALL SCFFTM(2,N,X,COMM,INFO)
      DO 10 I = 1, N
         X(I,3) = X(I,1)*X(N-I+1,2)
10    CONTINUE
      CALL CSFFTM(1,N,X(1,3),COMM,INFO)
```

5.3.3 1D Complex-To-Real FFT (Hermitian-Packed Storage)

ZDFFT Routine Documentation

ZDFFT (*MODE,N,X,COMM,INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZDFFT. On input:

- *MODE*=0 : only initializations (specific to the values of *N*) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real transform is performed. Initializations are assumed to have been performed by a prior call to ZDFFT.
- *MODE*=2 : (default) initializations and a real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of *N*) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of *N*.

INTEGER N [Input]

On input: *N* is length of the sequence in *X*

DOUBLE PRECISION X(N) [Input/Output]

On input: *X* contains the Hermitian sequence of length *N* to be transformed. On output: *X* contains the transformed real sequence.

DOUBLE PRECISION COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL DZFFT(0,N,X,COMM,INFO)
      CALL DZFFT(1,N,X,COMM,INFO)
      DO 10 I = N/2+2, N
         X(I) = -X(I)
10    CONTINUE
      CALL ZDFFT(2,N,X,COMM,INFO)
```

CSFFT Routine Documentation

CSFFT (*MODE*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CSFFT. On input:

- *MODE*=0 : only initializations (specific to the values of *N*) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real transform is performed. Initializations are assumed to have been performed by a prior call to CSFFT.
- *MODE*=2 : (default) initializations and a real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of *N*) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of *N*.

INTEGER N [Input]

On input: *N* is the length of the sequence in *X*

REAL X(N) [Input/Output]

On input: *X* contains the Hermitian sequence of length *N* to be transformed.
On output: *X* contains the transformed real sequence.

REAL COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL SCFFT(0,N,X,COMM,INFO)
      CALL SCFFT(1,N,X,COMM,INFO)
      DO 10 I = N/2+2, N
         X(I) = -X(I)
10    CONTINUE
      CALL CSFFT(2,N,X,COMM,INFO)
```

5.3.4 Multiple 1D Complex-To-Real FFT (Hermitian-Packed Storage)

ZDFFTM Routine Documentation

ZDFFTM (*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER M [Input]

On input: M is the number of sequences to be transformed.

INTEGER N [Input]

On input: N is the length of the sequences in X

DOUBLE PRECISION X(N*M) [Input/Output]

On input: X contains the M Hermitian sequences of length N to be transformed. Element i of sequence j is stored in location $i + (j - 1) * N$ of X.
On output: X contains the transformed real sequences.

DOUBLE PRECISION COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL DZFFTM(1,N,X,COMM,INFO)
      CALL DZFFTM(2,N,X,COMM,INFO)
      DO 10 I = 1, N
         X(I,3) = X(I,1)*X(N-I+1,2)
10    CONTINUE
      CALL ZDFFTM(1,N,X(1,3),COMM,INFO)
```

CSFFTM Routine Documentation

CSFFTM (*M*,*N*,*X*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER M [Input]

On input: M is the number of sequences to be transformed.

INTEGER N [Input]

On input: N is the length of the sequences in X

REAL X(N*M) [Input/Output]

On input: X contains the M Hermitian sequences of length N to be transformed. Element i of sequence j is stored in location $i + (j - 1) * N$ of X.

On output: X contains the transformed real sequences.

REAL COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
      CALL SCFFTM(1,N,X,COMM,INFO)
      CALL SCFFTM(2,N,X,COMM,INFO)
      DO 10 I = 1, N
         X(I,3) = X(I,1)*X(N-I+1,2)
10    CONTINUE
      CALL CSFFTM(1,N,X(1,3),COMM,INFO)
```

5.3.5 1D Real-To-Complex FFT (Complex-Hermitian Storage)

DZFFT1D Routine Documentation

DZFFT1D (*MODE*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by DZFFT1D. On input:

- *MODE*=0 : only default initializations (specific to N) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real-to-complex transform is performed. Initializations are assumed to have been performed by a prior call to DZFFT1D.
- *MODE*=2 : (default) initializations and a real-to-complex transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER N [Input]

On input: N is length of the sequence in X

DOUBLE PRECISION X(N) [Input]

On input: X contains the real sequence of length N to be transformed.

DOUBLE COMPLEX Y(N/2+1) [Output]

On output: Y contains the tansformed complex sequence with roughly half redundant information due to complex conjugate removed.

DOUBLE PRECISION COMM(4*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL DZFFT1D(0,N,X,Y,COMM,INFO)
CALL DZFFT1D(1,N,X,Y,COMM,INFO)
DO 10 I = 1, N/2+1
   Y(I) = -Y(I)*EXP(-DBLE(I-1)/DBLE(N))
10 CONTINUE
CALL ZDFFT1D(2,N,Y,X,COMM,INFO)
```

SCFFT1D Routine Documentation

SCFFT1D (*MODE*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by SCFFT1D. On input:

- *MODE*=0 : only default initializations (specific to N) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real-to-complex transform is performed. Initializations are assumed to have been performed by a prior call to SCFFT1D.
- *MODE*=2 : (default) initializations and a real-to-complex transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER N [Input]

On input: N is length of the sequence in X

REAL X(N) [Input]

On input: X contains the real sequence of length N to be transformed.

COMPLEX Y(N/2+1) [Output]

On output: Y contains the tansformed complex sequence with roughly half redundant information due to complex conjugate removed.

REAL COMM(4*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL SCFFT1D(0,N,X,Y,COMM,INFO)
CALL SCFFT1D(1,N,X,Y,COMM,INFO)
DO 10 I = 1, N/2+1
   Y(I) = -Y(I)*EXP(-REAL(I-1)/REAL(N))
10 CONTINUE
CALL CSFFT1D(2,N,Y,X,COMM,INFO)
```

5.3.6 1D Complex-To-Real FFT (Complex-Hermitian Storage)

ZDFFT1D Routine Documentation

ZDFFT1D (*MODE*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZDFFT1D. On input:

- *MODE*=0 : only initializations (specific to the values of *N*) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a complex-to-real transform is performed. Initializations are assumed to have been performed by a prior call to ZDFFT1D.
- *MODE*=2 : (default) initializations and a complex-to-real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of *N*) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of *N*.

INTEGER N [Input]

On input: *N* is length of the sequence in *Y*

DOUBLE COMPLEX X(N/2+1) [Input]

On input: *X* contains the complex sequence to be transformed.

DOUBLE PRECISION Y(N) [Output]

On output: *Y* contains the transformed real sequence of length *N*.

DOUBLE PRECISION COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL DZFFT1D(0,N,X,Y,COMM,INFO)
CALL DZFFT1D(1,N,X,Y,COMM,INFO)
DO 10 I = 1, N/2+1
   Y(I) = -Y(I)*EXP(-DBLE(I-1)/DBLE(N))
10 CONTINUE
CALL ZDFFT1D(2,N,Y,X,COMM,INFO)
```

CSFFT1D Routine Documentation

CSFFT1D (*MODE*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CSFFT1D. On input:

- *MODE*=0 : only initializations (specific to the values of N) are performed using a default plan; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a complex-to-real transform is performed. Initializations are assumed to have been performed by a prior call to CSFFT1D.
- *MODE*=2 : (default) initializations and a complex-to-real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations (specific to the value of N) are performed, but these are based on a plan that is first generated by timing a subset of all possible plans and choosing the quickest (i.e. the FFT computation was timed as fastest based on the chosen plan). The plan generation phase may take a significant amount of time depending on the value of N.

INTEGER N [Input]

On input: N is length of the sequence in Y

COMPLEX X(N/2+1) [Input]

On input: X contains the complex sequence to be transformed.

REAL Y(N) [Output]

On output: Y contains the transformed real sequence of length N.

REAL COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length N. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL SCFFT1D(0,N,X,Y,COMM,INFO)
CALL SCFFT1D(1,N,X,Y,COMM,INFO)
DO 10 I = 1, N/2+1
   Y(I) = -Y(I)*EXP(-REAL(I-1)/REAL(N))
10 CONTINUE
CALL CSFFT1D(2,N,Y,X,COMM,INFO)
```

5.3.7 Multiple 1D Real-To-Complex FFT (Complex-Hermitian Storage)

DZFFT1M Routine Documentation

DZFFT1M (*MODE,M,N,X,Y,COMM,INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by DZFFT1M. On input:

- *MODE*=0 : only default initializations (specific to *N*) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real-to-complex transform is performed. Initializations are assumed to have been performed by a prior call to DZFFT1M.
- *MODE*=2 : (default) initializations and a real-to-complex transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER M [Input]

On input: *M* is the number of sequences to be transformed.

INTEGER N [Input]

On input: *N* is length of the sequence in *X*

DOUBLE PRECISION X(N*M) [Input]

On input: *X* contains the *M* real sequences of length *N* to be transformed. Element *i* of sequence *j* is stored in location *i + (j - 1) * N* of *X*.

DOUBLE COMPLEX Y((N/2+1)*M) [Output]

On output: *Y* contains the transformed Hermitian sequences in complex-Hermitian storage.

DOUBLE PRECISION COMM(4*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL DZFFT1M(0,M,N,X,Y,COMM,INFO)
CALL DZFFT1M(1,M,N,X,Y,COMM,INFO)
DO J = 1, M
   DO I = 1, N/2+1
      Y(I,J) = -Y(I,J)*EXP(-DBLE(I-1)/DBLE(N))
   END DO
EMD DO
CALL ZDFFT1M(2,M,N,Y,X,COMM,INFO)
```

SCFFT1M Routine Documentation

SCFFT1M (*MODE,M,N,X,Y,COMM,INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by SCFFT1M. On input:

- *MODE*=0 : only default initializations (specific to *N*) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real-to-complex transform is performed. Initializations are assumed to have been performed by a prior call to SCFFT1M.
- *MODE*=2 : (default) initializations and a real-to-complex transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER M [Input]

On input: *M* is the number of sequences to be transformed.

INTEGER N [Input]

On input: *N* is length of the sequence in *X*

REAL X(N*M) [Input]

On input: *X* contains the *M* real sequences of length *N* to be transformed. Element *i* of sequence *j* is stored in location *i + (j - 1) * N* of *X*.

COMPLEX Y((N/2+1)*M) [Output]

On output: *Y* contains the transformed Hermitian sequences in complex-Hermitian storage.

REAL COMM(4*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL SCFFT1M(0,M,N,X,Y,COMM,INFO)
CALL SCFFT1M(1,M,N,X,Y,COMM,INFO)
DO J = 1, M
   DO I = 1, N/2+1
      Y(I,J) = -Y(I,J)*EXP(-REAL(I-1)/REAL(N))
   END DO
EMD DO
CALL CSFFT1M(2,M,N,Y,X,COMM,INFO)
```

5.3.8 Multiple 1D Complex-To-Real FFT (Complex-Hermitian Storage)

ZDFFT1M Routine Documentation

ZDFFT1M (*MODE,M,N,X,Y,COMM,INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZDFFT1M. On input:

- *MODE*=0 : only default initializations (specific to *N*) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a complex-to-real transform is performed. Initializations are assumed to have been performed by a prior call to ZDFFT1M.
- *MODE*=2 : (default) initializations and a complex-to-real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER M [Input]

On input: *M* is the number of sequences to be transformed.

INTEGER N [Input]

On input: *N* is length of the sequence in *Y*

DOUBLE COMPLEX X((N/2+1)*M) [Input]

On input: *X* contains the Hermitian sequences to be transformed in complex-Hermitian storage.

DOUBLE PRECISION Y(N*M) [Output]

On output: *Y* contains the *M* real sequences of length *N* transformed. Element *i* of sequence *j* is stored in location *i + (j - 1) * N* of *Y*.

DOUBLE PRECISION COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL DZFFT1M(0,M,N,X,Y,COMM,INFO)
CALL DZFFT1M(1,M,N,X,Y,COMM,INFO)
DO J = 1, M
   DO I = 1, N/2+1
      Y(I,J) = -Y(I,J)*EXP(-DBLE(I-1)/DBLE(N))
   END DO
EMD DO
CALL ZDFFT1M(2,M,N,Y,X,COMM,INFO)
```

CSFFT1M Routine Documentation

CSFFT1M (*MODE,M,N,X,Y,COMM,INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CSFFT1M. On input:

- *MODE*=0 : only default initializations (specific to *N*) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a complex-to-real transform is performed. Initializations are assumed to have been performed by a prior call to CSFFT1M.
- *MODE*=2 : (default) initializations and a complex-to-real transform a re performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER M [Input]

On input: *M* is the number of sequences to be transformed.

INTEGER N [Input]

On input: *N* is length of the sequence in *Y*

COMPLEX X((N/2+1)*M) [Input]

On input: *X* contains the Hermitian sequences to be transformed in complex-Hermitian storage.

REAL Y(N*M) [Output]

On output: *Y* contains the *M* real sequences of length *N* transformed. Element *i* of sequence *j* is stored in location *i + (j - 1) * N* of *Y*.

REAL COMM(3*N+100) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL SCFFT1M(0,M,N,X,Y,COMM,INFO)
CALL SCFFT1M(1,M,N,X,Y,COMM,INFO)
DO J = 1, M
   DO I = 1, N/2+1
      Y(I,J) = -Y(I,J)*EXP(-REAL(I-1)/REAL(N))
   END DO
EMD DO
CALL CSFFT1M(2,M,N,Y,X,COMM,INFO)
```

5.3.9 2D Real-To-Complex FFT

DZFFT2D Routine Documentation

DZFFT2D (*MODE*,*M*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by DZFFT2D. On input:

- *MODE*=0 : only default initializations (specific to M and N) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real-to-complex transform is performed. Initializations are assumed to have been performed by a prior call to DZFFT2D.
- *MODE*=2 : (default) initializations and a real-to-complex transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER M [Input]

On input: M is the number of rows in the 2D array of data to be transformed. If X is declared as a 2D array then M is the first dimension of X.

INTEGER N [Input]

On input: N is the number of columns in the 2D array of data to be transformed. If X is declared as a 2D array then N is the second dimension of X.

DOUBLE PRECISION X(M*N) [Input]

On input: X contains the M by N real 2D array to be transformed. Element *ij* is stored in location *i + (j - 1) * M* of X.

DOUBLE COMPLEX Y((M/2+1)*N) [Output]

On output: Y contains the transformed Hermitian sequences in complex-Hermitian storage.

DOUBLE PRECISION COMM(4*M+6*N+300) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *M*N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL DZFFT2D(0,M,N,X,Y,COMM,INFO)
CALL DZFFT2D(1,M,N,X,Y,COMM,INFO)
DO J = 1, N
   DO I = 1, M/2+1
      Y(I,J) = -Y(I,J)/SQRT(DBLE(M*N))
   END DO
EMD DO
CALL ZDFFT2D(2,M,N,Y,X,COMM,INFO)
```

SCFFT2D Routine Documentation

SCFFT2D (*MODE*,*M*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by SCFFT2D. On input:

- *MODE*=0 : only default initializations (specific to M and N) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real-to-complex transform is performed. Initializations are assumed to have been performed by a prior call to SCFFT2D.
- *MODE*=2 : (default) initializations and a real-to-complex transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER M [Input]

On input: M is the number of rows in the 2D array of data to be transformed. If X is declared as a 2D array then M is the first dimension of X.

INTEGER N [Input]

On input: N is the number of columns in the 2D array of data to be transformed. If X is declared as a 2D array then N is the second dimension of X.

REAL X(M*N) [Input]

On input: X contains the M by N real 2D array to be transformed. Element ij is stored in location $i + (j - 1) * M$ of X.

COMPLEX Y((M/2+1)*N) [Output]

On output: Y contains the transformed Hermitian sequences in complex-Hermitian storage.

REAL COMM(4*M+10*N+300) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *M*N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL SCFFT2D(0,M,N,X,Y,COMM,INFO)
CALL SCFFT2D(1,M,N,X,Y,COMM,INFO)
DO J = 1, N
   DO I = 1, M/2+1
      Y(I,J) = -Y(I,J)/SQRT(REAL(M*N))
   END DO
EMD DO
CALL CSFFT2D(2,M,N,Y,X,COMM,INFO)
```

5.3.10 2D Complex-To-Real FFT

ZDFFT2D Routine Documentation

ZDFFT2D (*MODE*,*M*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZDFFT2D. On input:

- *MODE*=0 : only default initializations (specific to M and N) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a complex-to-real transform is performed. Initializations are assumed to have been performed by a prior call to ZDFFT2D.
- *MODE*=2 : (default) initializations and a complex-to-real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER M [Input]

On input: M is the number of rows in the 2D array of the real data obtained from the transform. If Y is declared as a 2D array then M is the first dimension of Y.

INTEGER N [Input]

On input: N is the number of columns in the 2D array of the real data obtained from the transform. If Y is declared as a 2D array then N is the second dimension of Y.

DOUBLE COMPLEX X((M/2+1)*N) [Input]

On input: X contains the Hermitian sequences in complex-Hermitian storage to be transformed.

DOUBLE PRECISION Y(M*N) [Output]

On output: Y contains the M by N real 2D array obtained from the transform. Element *ij* is stored in location *i + (j - 1) * M* of Y.

DOUBLE PRECISION COMM(4*M+6*N+300) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *M*N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL DZFFT2D(0,M,N,X,Y,COMM,INFO)
CALL DZFFT2D(1,M,N,X,Y,COMM,INFO)
DO J = 1, N
   DO I = 1, M/2+1
      Y(I,J) = -Y(I,J)/SQRT(DBLE(M*N))
   END DO
EMD DO
CALL ZDFFT2D(2,M,N,Y,X,COMM,INFO)
```

CSFFT2D Routine Documentation

CSFFT2D (*MODE,M,N,X,Y,COMM,INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CSFFT2D. On input:

- *MODE*=0 : only default initializations (specific to *M* and *N*) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a complex-to-real transform is performed. Initializations are assumed to have been performed by a prior call to CSFFT2D.
- *MODE*=2 : (default) initializations and a complex-to-real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER M [Input]

On input: *M* is the number of rows in the 2D array of the real data obtained from the transform. If *Y* is declared as a 2D array then *M* is the first dimension of *Y*.

INTEGER N [Input]

On input: *N* is the number of columns in the 2D array of the real data obtained from the transform. If *Y* is declared as a 2D array then *N* is the second dimension of *Y*.

COMPLEX X((M/2+1)*N) [Input]

On input: *X* contains the Hermitian sequences in complex-Hermitian storage to be transformed.

REAL Y(M*N) [Output]

On output: *Y* contains the *M* by *N* real 2D array obtained from the transform. Element *ij* is stored in location *i + (j - 1) * M* of *Y*.

REAL COMM(4*M+10*N+300) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *M*N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL SCFFT2D(0,M,N,X,Y,COMM,INFO)
CALL SCFFT2D(1,M,N,X,Y,COMM,INFO)
DO J = 1, N
   DO I = 1, M/2+1
      Y(I,J) = -Y(I,J)/SQRT(REAL(M*N))
   END DO
EMD DO
CALL CSFFT2D(2,M,N,Y,X,COMM,INFO)
```

5.3.11 3D Real-To-Complex FFT

DZFFT3D Routine Documentation

DZFFT3D (*MODE,L,M,N,X,Y,COMM,INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by DZFFT3D. On input:

- *MODE*=0 : only default initializations (specific to *L*, *M* and *N*) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real-to-complex transform is performed. Initializations are assumed to have been performed by a prior call to DZFFT3D.
- *MODE*=2 : (default) initializations and a real-to-complex transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER L [Input]

On input: *L* is the length of the first dimension of the 3D array of data to be transformed. If *X* is declared as a 3D array then *L* is the first dimension of *X*.

INTEGER M [Input]

On input: *M* is the length of the second dimension of the 3D array of data to be transformed. If *X* is declared as a 3D array then *L* is the second dimension of *X*.

INTEGER N [Input]

On input: *N* is the length of the third dimension of the 3D array of data to be transformed. If *X* is declared as a 3D array then *L* is the third dimension of *X*.

DOUBLE PRECISION X(L*M*N) [Input]

On input: *X* contains the *L* by *M* by *N* real 3D array to be transformed. Element *ijk* is stored in location *i + (j - 1)* L + (k - 1) * L * M* of *X*.

DOUBLE COMPLEX Y((L/2+1)*M*N) [Output]

On output: *Y* contains the transformed Hermitian sequences in complex-Hermitian storage.

DOUBLE PRECISION COMM(4*L+6*M+6*N+500) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *L*M*N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL DZFFT3D(0,L,M,N,X,Y,COMM,INFO)
CALL DZFFT3D(1,L,M,N,X,Y,COMM,INFO)
DO K = 1, N
   DO J = 1, M
      DO I = 1, L/2+1
         Y(I,J,K) = -Y(I,J,K)/SQRT(DBLE(L*M*N))
      END DO
   END DO
EMD DO
CALL ZDFFT3D(2,L,M,N,Y,X,COMM,INFO)
```

SCFFT3D Routine Documentation

SCFFT3D (*MODE*,*L*,*M*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by SCFFT3D. On input:

- *MODE*=0 : only default initializations (specific to L, M and N) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a real-to-complex transform is performed. Initializations are assumed to have been performed by a prior call to SCFFT3D.
- *MODE*=2 : (default) initializations and a real-to-complex transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER L [Input]

On input: L is the length of the first dimension of the 3D array of data to be transformed. If X is declared as a 3D array then L is the first dimension of X.

INTEGER M [Input]

On input: M is the length of the second dimension of the 3D array of data to be transformed. If X is declared as a 3D array then L is the second dimension of X.

INTEGER N [Input]

On input: N is the length of the third dimension of the 3D array of data to be transformed. If X is declared as a 3D array then L is the third dimension of X.

REAL X(L*M*N) [Input]

On input: X contains the L by M by N real 3D array to be transformed. Element *ijk* is stored in location *i + (j - 1)* L + (k - 1) * L * M* of X.

COMPLEX Y((L/2+1)*M*N) [Output]

On output: Y contains the transformed Hermitian sequences in complex-Hermitian storage.

REAL COMM(4*L+10*M+10*N+500) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *L*M*N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL SCFFT3D(0,L,M,N,X,Y,COMM,INFO)
CALL SCFFT3D(1,L,M,N,X,Y,COMM,INFO)
DO K = 1, N
   DO J = 1, M
      DO I = 1, L/2+1
         Y(I,J,K) = -Y(I,J,K)/SQRT(REAL(L*M*N))
      END DO
   END DO
EMD DO
CALL CSFFT3D(2,L,M,N,Y,X,COMM,INFO)
```

5.3.12 3D Complex-To-Real FFT

ZDFFT3D Routine Documentation

ZDFFT3D (*MODE,L,M,N,X,Y,COMM,INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by ZDFFT3D. On input:

- *MODE*=0 : only default initializations (specific to L, M and N) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a complex-to-real transform is performed. Initializations are assumed to have been performed by a prior call to ZDFFT3D.
- *MODE*=2 : (default) initializations and a complex-to-real transform are performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER L [Input]

On input: L is the length of the first dimension of the 3D array of data obtained from the transform. If Y is declared as a 3D array then L is the first dimension of Y.

INTEGER M [Input]

On input: M is the length of the second dimension of the 3D array of data obtained from the transform. If Y is declared as a 3D array then L is the second dimension of Y.

INTEGER N [Input]

On input: N is the length of the third dimension of the 3D array of data obtained from the transform. If Y is declared as a 3D array then L is the third dimension of Y.

DOUBLE COMPLEX X((L/2+1)*M*N) [Input]

On input: X contains the Hermitian sequences in complex-Hermitian storage to be transformed.

DOUBLE PRECISION Y(L*M*N) [Output]

On output: Y contains the L by M by N real 3D array obtained from the transform. Element *ijk* is stored in location *i + (j - 1) * L + (k - 1) * L * M* of Y.

DOUBLE PRECISION COMM(4*L+6*M+6*N+500) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *L*M*N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL DZFFT3D(0,L,M,N,X,Y,COMM,INFO)
CALL DZFFT3D(1,L,M,N,X,Y,COMM,INFO)
DO K = 1, N
   DO J = 1, M
      DO I = 1, L/2+1
         Y(I,J,K) = -Y(I,J,K)/SQRT(DBLE(L*M*N))
      END DO
   END DO
EMD DO
CALL ZDFFT3D(2,L,M,N,Y,X,COMM,INFO)
```

CSFFT3D Routine Documentation

CSFFT3D (*MODE*,*L*,*M*,*N*,*X*,*Y*,*COMM*,*INFO*) [SUBROUTINE]

INTEGER MODE [Input]

The value of *MODE* on input determines the operation performed by CSFFT3D. On input:

- *MODE*=0 : only default initializations (specific to L, M and N) are performed; this is usually followed by calls to the same routine with *MODE*=1.
- *MODE*=1 : a complex-to-real transform is performed. Initializations are assumed to have been performed by a prior call to CSFFT3D.
- *MODE*=2 : (default) initializations and a complex-to-real transform a re performed.
- *MODE*=100 : similar to *MODE*=0; only initializations are performed, but first a plan is generated. This plan is chosen based on the fastest FFT computation for a subset of all possible plans.

INTEGER L [Input]

On input: L is the length of the first dimension of the 3D array of data obtained from the transform. If Y is declared as a 3D array then L is the first dimension of Y.

INTEGER M [Input]

On input: M is the length of the second dimension of the 3D array of data obtained from the transform. If Y is declared as a 3D array then L is the second dimension of Y.

INTEGER N [Input]

On input: N is the length of the third dimension of the 3D array of data obtained from the transform. If Y is declared as a 3D array then L is the third dimension of Y.

COMPLEX X((L/2+1)*M*N) [Input]

On input: X contains the Hermitian sequences in complex-Hermitian storage to be transformed.

REAL Y(L*M*N) [Output]

On output: Y contains the L by M by N real 3D array obtained from the transform. Element *ijk* is stored in location *i + (j - 1) * L + (k - 1) * L * M* of Y.

REAL COMM(4*L+10*M+10*N+500) [Input/Output]

COMM is a communication array. Some portions of the array are used to store initializations for subsequent calls with the same sequence length *L*M*N*. The remainder is used as temporary store.

INTEGER INFO [Output]

On output: *INFO* is an error indicator. On successful exit, *INFO* contains 0. If *INFO* = −i on exit, the i-th argument had an illegal value.

```
CALL SCFFT3D(0,L,M,N,X,Y,COMM,INFO)
CALL SCFFT3D(1,L,M,N,X,Y,COMM,INFO)
DO K = 1, N
   DO J = 1, M
      DO I = 1, L/2+1
         Y(I,J,K) = -Y(I,J,K)/SQRT(REAL(L*M*N))
      END DO
   END DO
EMD DO
CALL CSFFT3D(2,L,M,N,Y,X,COMM,INFO)
```

6 References

- [1] C.L. Lawson, R.J. Hanson, D. Kincaid, and F.T. Krogh, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Maths. Soft., 5 (1979), pp. 308–323.
- [2] J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson, *An extended set of FORTRAN basic linear algebra subroutines*, ACM Trans. Math. Soft., 14 (1988), pp. 1–17.
- [3] J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling, *A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Soft., 16 (1990), pp. 1–17.
- [4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK User's Guide*, SIAM, Philidelphia, (1999).
- [5] IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985)

Subject Index

Routine Index

A

C

D

S

Z