CUDA

CUFFT Library

PG-00000-003_V3.0
February, 2010

Published by
NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

Notice

Trademarks

Copyright

Table of Contents

CUFFT Library

This document describes CUFFT, the NVIDIA® CUDA™ Fast Fourier Transform (FFT) library. The FFT is a divide-and-conquer algorithm for efficiently computing discrete Fourier transforms of complex or real-valued data sets, and it is one of the most important and widely used numerical algorithms, with applications that include computational physics and general signal processing. The CUFFT library provides a simple interface for computing parallel FFTs on an NVIDIA GPU, which allows users to leverage the floating-point power and parallelism of the GPU without having to develop a custom, GPU-based FFT implementation.

FFT libraries typically vary in terms of supported transform sizes and data types. For example, some libraries only implement Radix-2 FFTs, restricting the transform size to a power of two, while other implementations support arbitrary transform sizes. This version of the CUFFT library supports the following features:

- 1D, 2D, and 3D transforms of complex and real-valued data
- Batch execution for doing multiple transforms of any dimension in parallel
- 2D and 3D transform sizes in the range `[2, 16384]` in any dimension
- 1D transform sizes up to 8 million elements
- In-place and out-of-place transforms for real and complex data
- Double-precision transforms on compatible hardware (GT200 and later GPUs)
- Support for streamed execution, enabling simultaneous computation together with data movement

CUFFT Types and Definitions

The next sections describe the CUFFT types and transform directions:

Type cufftHandle

`typedef unsigned int cufftHandle;`

is a handle type used to store and access CUFFT plans (see "CUFFT API Functions" on page 6 for more information about plans). For example, the user receives a handle after creating a CUFFT plan and uses this handle to execute the plan.

Type cufftResult

`typedef enum cufftResult_t cufftResult;`

is an enumeration of values used exclusively as API function return values. The possible return values are defined as follows:

Return Values

CUFFT_SUCCESS	Any CUFFT operation is successful.
CUFFT_INVALID_PLAN	CUFFT is passed an invalid plan handle.
CUFFT_ALLOC_FAILED	CUFFT failed to allocate GPU memory.
CUFFT_INVALID_TYPE	The user requests an unsupported type.
CUFFT_INVALID_VALUE	The user specifies a bad memory pointer.
CUFFT_INTERNAL_ERROR	Used for all internal driver errors.
CUFFT_EXEC_FAILED	CUFFT failed to execute an FFT on the GPU.
CUFFT_SETUP_FAILED	The CUFFT library failed to initialize.

Return Values (continued)

CUFFT_SHUTDOWN_FAILED	The CUFFT library failed to shut down.
CUFFT_INVALID_SIZE	The user specifies an unsupported FFT size.

Type cufftReal

typedef float cufftReal;

is a single-precision, floating-point real data type.

Type cufftDoubleReal

typedef double cufftDoubleReal;

is a double-precision, floating-point real data type.

Type cufftComplex

typedef cuComplex cufftComplex;

is a single-precision, floating-point complex data type that consists of interleaved real and imaginary components.

Type cufftDoubleComplex

typedef cuDoubleComplex cufftDoubleComplex;

is a double-precision, floating-point complex data type that consists of interleaved real and imaginary components.

CUFFT Transform Types

The CUFFT library supports complex- and real-data transforms. The **cufftType** data type is an enumeration of the types of transform data supported by CUFFT:

```
typedef enum cufftType_t {
    CUFFT_R2C = 0x2a,  // Real to complex (interleaved)
    CUFFT_C2R = 0x2c,  // Complex (interleaved) to real
    CUFFT_C2C = 0x29,  // Complex to complex, interleaved
    CUFFT_D2Z = 0x6a,  // Double to double-complex
    CUFFT_Z2D = 0x6c,  // Double-complex to double
```

```
    CUFFT_Z2Z = 0x69   // Double-complex to double-complex
} cufftType;
```

For complex FFTs, the input and output arrays must interleave the real and imaginary parts (the **`cufftComplex`** type). The transform size in each dimension is the number of **`cufftComplex`** elements. The **`CUFFT_C2C`** constant can be passed to any plan creation function to configure a single-precision complex-to-complex FFT. Pass the **`CUFFT_Z2Z`** constant to configure a double-precision complex-to-complex FFT.

For real-to-complex FFTs, the output array holds only the non-redundant complex coefficients. So for an N-element transform, the output array holds `N/2+1` **`cufftComplex`** terms. For higher-dimensional real transforms of the form `N0×N1×...×Nn`, the last dimension is cut in half such that the output data is `N0×N1×...×(Nn/2+1)` complex elements. Therefore, in order to perform an in-place FFT, the user has to pad the input array in the last dimension to `(Nn/2+1)` complex elements or `2*(N/2+1)` real elements. Note that the real-to-complex transform is implicitly forward. Passing the **`CUFFT_R2C`** constant to any plan creation function configures a single-precision real-to-complex FFT. Passing the **`CUFFT_D2Z`** constant configures a double-precision real-to-complex FFT.

The requirements for complex-to-real FFTs are similar to those for real-to-complex. In this case, the input array holds only the non-redundant, `N/2+1` complex coefficients from a real-to-complex transform. The output is simply `N` elements of type **`cufftReal`**. However, for an in-place transform, the input size must be padded to `2*(N/2+1)` real elements. The complex-to-real transform is implicitly inverse. Passing the **`CUFFT_C2R`** constant to any plan creation function configures a single-precision complex-to-real FFT. Passing **`CUFFT_Z2D`** constant configures a double-precision complex-to-real FFT.

For 1D complex-to-complex transforms, the stride between signals in a batch is assumed to be the number of **`cufftComplex`** elements in the logical transform size. However, for real-data FFTs, the distance between signals in a batch depends on whether the transform is in-place or out-of-place. For in-place FFTs, the input stride is assumed to be `2*(N/2+1)` **`cufftReal`** elements or `N/2+1` **`cufftComplex`** elements.

For out-of-place transforms, input and output strides match the logical transform size (`N`) and the non-redundant size (`N/2+1`), respectively.

Starting with CUFFT version 3.0, batched transforms are supported through the **`cufftPlanMany()`** function. Although this function takes input parameters that specify input- and output-data strides, in version 3.0 data for each signal within the batch is assumed to immediately follow that of the previous one (a stride of 1).

CUFFT Transform Directions

The CUFFT library defines forward and inverse Fast Fourier Transforms according to the sign of the complex exponential term:

```
#define CUFFT_FORWARD -1
#define CUFFT_INVERSE  1
```

For higher-dimensional transforms (2D and 3D), CUFFT performs FFTs in row-major or C order. For example, if the user requests a 3D transform plan for sizes *X*, *Y*, and *Z*, CUFFT transforms along *Z*, *Y*, and then *X*. The user can configure column-major FFTs by simply changing the order of the size parameters to the plan creation API functions.

CUFFT performs un-normalized FFTs; that is, performing a forward FFT on an input data set followed by an inverse FFT on the resulting set yields data that is equal to the input scaled by the number of elements. Scaling either transform by the reciprocal of the size of the data set is left for the user to perform as seen fit.

Streamed CUFFT Transforms

Execution of a transform of a particular size and type may take several stages of processing. A plan for the transform is generated, in which CUFFT specifies the internal steps that need to be taken. These steps may include multiple kernel launches, memory copies, and so on.

Every CUFFT plan may be associated with a CUDA stream. Once so associated, all launches of the internal stages of that plan take place through the specified stream. Streaming of launches allows for potential overlap between transforms and memory copies—see the *NVIDIA CUDA Programming Guide* for more information on streams. If no stream is associated with a plan, launches take place in stream `0` (the default CUDA stream).

CUFFT API Functions

The CUFFT API is modeled after FFTW (see http://www.fftw.org), which is one of the most popular and efficient CPU-based FFT libraries. FFTW provides a simple configuration mechanism called a *plan* that completely specifies the optimal—that is, the minimum floating-point operation (flop)—plan of execution for a particular FFT size and data type. The advantage of this approach is that once the user creates a plan, the library stores whatever state is needed to execute the plan multiple times without recalculation of the configuration. The FFTW model works well for CUFFT because different kinds of FFTs require different thread configurations and GPU resources, and plans are a simple way to store and reuse configurations.

The CUFFT library initializes internal data upon the first invocation of an API function. Therefore, all API functions could return the **`CUFFT_SETUP_FAILED`** error code if the library fails to initialize. CUFFT shuts down automatically when all user-created FFT plans are destroyed. The CUFFT functions are as follows:

- "Function cufftPlan1d()" on page 7
- "Function cufftPlan2d()" on page 7
- "Function cufftPlan3d()" on page 8
- "Function cufftPlanMany()" on page 9
- "Function cufftDestroy()" on page 10
- "Function cufftExecC2C()" on page 10
- "Function cufftExecR2C()" on page 11
- "Function cufftExecC2R()" on page 12
- "Function cufftExecZ2Z()" on page 13
- "Function cufftExecD2Z()" on page 13
- "Function cufftExecZ2D()" on page 14
- "Function cufftSetStream()" on page 15

Function cufftPlan1d()

```
cufftResult
cufftPlan1d( cufftHandle *plan, int nx, cufftType type,
             int batch );
```

creates a 1D FFT plan configuration for a specified signal size and data type. The `batch` input parameter tells CUFFT how many 1D transforms to configure.

Input

`plan`	Pointer to a `cufftHandle` object
`nx`	The transform size (e.g., 256 for a 256-point FFT)
`type`	The transform data type (e.g., `CUFFT_C2C` for complex to complex)
`batch`	Number of transforms of size `nx`

Output

`plan`	Contains a CUFFT 1D plan handle value

Return Values

`CUFFT_SETUP_FAILED`	CUFFT library failed to initialize.
`CUFFT_INVALID_SIZE`	The `nx` parameter is not a supported size.
`CUFFT_INVALID_TYPE`	The `type` parameter is not supported.
`CUFFT_ALLOC_FAILED`	Allocation of GPU resources for the plan failed.
`CUFFT_SUCCESS`	CUFFT successfully created the FFT plan.

Function cufftPlan2d()

```
cufftResult
cufftPlan2d( cufftHandle *plan, int nx, int ny,
             cufftType type );
```

creates a 2D FFT plan configuration according to specified signal sizes and data type. This function is the same as **`cufftPlan1d()`** except that it takes a second size parameter, `ny`, and does not support batching.

Input

`plan`	Pointer to a `cufftHandle` object
`nx`	The transform size in the *X* dimension (number of rows)
`ny`	The transform size in the *Y* dimension (number of columns)
`type`	The transform data type (e.g., `CUFFT_C2R` for complex to real)

Output

`plan`	Contains a CUFFT 2D plan handle value

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_INVALID_SIZE	The `nx` or `ny` parameter is not a supported size.
CUFFT_INVALID_TYPE	The `type` parameter is not supported.
CUFFT_ALLOC_FAILED	Allocation of GPU resources for the plan failed.
CUFFT_SUCCESS	CUFFT successfully created the FFT plan.

Function cufftPlan3d()

```
cufftResult
cufftPlan3d( cufftHandle *plan, int nx, int ny, int nz,
             cufftType type );
```

creates a 3D FFT plan configuration according to specified signal sizes and data type. This function is the same as **`cufftPlan2d()`** except that it takes a third size parameter `nz`. :

Input

`plan`	Pointer to a `cufftHandle` object
`nx`	The transform size in the *X* dimension
`ny`	The transform size in the *Y* dimension
`nz`	The transform size in the *Z* dimension
`type`	The transform data type (e.g., `CUFFT_R2C` for real to complex)

Output

`plan`	Contains a CUFFT 3D plan handle value

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_INVALID_SIZE	Parameter `nx`, `ny`, or `nz` is not a supported size.
CUFFT_INVALID_TYPE	The `type` parameter is not supported.
CUFFT_ALLOC_FAILED	Allocation of GPU resources for the plan failed.
CUFFT_SUCCESS	CUFFT successfully created the FFT plan.

Function cufftPlanMany()

```
cufftResult
cufftPlanMany( cufftHandle *plan, int rank, int *n,
               int *inembed, int istride, int idist,
               int *onembed, int ostride, int odist,
               cufftType type, int batch );
```

creates a FFT plan configuration of dimension **rank**, with sizes specified in the array **n**. The **batch** input parameter tells CUFFT how many transforms to configure in parallel. With this function, batched plans of any dimension may be created.

Input parameters **inembed**, **istride**, and **idist** and output parameters **onembed**, **ostride**, and **odist** will allow setup of non-contiguous input data in a future version. Note that for CUFFT 3.0, these parameters are ignored and the layout of batched data must be side-by-side and not interleaved. :

Input

`plan`	Pointer to a `cufftHandle` object
`rank`	Dimensionality of the transform (`1`, `2`, or `3`)
`n`	An array of size **rank**, describing the size of each dimension
`inembed`	Unused: pass `NULL`
`istride`	Unused: pass `1`
`idist`	Unused: pass `0`
`onembed`	Unused: pass `NULL`
`ostride`	Unused: pass `1`
`odist`	Unused: pass `0`
`type`	Transform data type (e.g., `CUFFT_C2C,` as per other CUFFT calls)
`batch`	Batch size for this transform

Output

`plan`	Contains a CUFFT plan handle

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_INVALID_SIZE	Parameter is not a supported size.
CUFFT_INVALID_TYPE	The `type` parameter is not supported.

Return Values (continued)

CUFFT_ALLOC_FAILED	Allocation of GPU resources for the plan failed.
CUFFT_SUCCESS	CUFFT successfully created the FFT plan.

Function cufftDestroy()

```
cufftResult
cufftDestroy( cufftHandle plan );
```

frees all GPU resources associated with a CUFFT plan and destroys the internal plan data structure. This function should be called once a plan is no longer needed to avoid wasting GPU memory.

Input

`plan`	The `cufftHandle` object of the plan to be destroyed.

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_SHUTDOWN_FAILED	CUFFT library failed to shut down.
CUFFT_INVALID_PLAN	The `plan` parameter is not a valid handle.
CUFFT_SUCCESS	CUFFT successfully destroyed the FFT plan.

Function cufftExecC2C()

```
cufftResult
cufftExecC2C( cufftHandle plan, cufftComplex *idata,
              cufftComplex *odata, int direction );
```

executes a CUFFT single-precision complex-to-complex transform plan as specified by `direction`. CUFFT uses as input data the GPU memory pointed to by the `idata` parameter. This function stores the Fourier coefficients in the `odata` array. If `idata` and `odata` are the same, this method does an in-place transform.

Input

`plan`	The `cufftHandle` object for the plan to update
`idata`	Pointer to the single-precision complex input data (in GPU memory) to transform
`odata`	Pointer to the single-precision complex output data (in GPU memory)
`direction`	The transform direction: `CUFFT_FORWARD` or `CUFFT_INVERSE`

Output

odata	Contains the complex Fourier coefficients

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_INVALID_PLAN	The plan parameter is not a valid handle.
CUFFT_INVALID_VALUE	The idata, odata, and/or direction parameter is not valid.
CUFFT_EXEC_FAILED	CUFFT failed to execute the transform on GPU.
CUFFT_SUCCESS	CUFFT successfully executed the FFT plan.

Function cufftExecR2C()

```
cufftResult
cufftExecR2C( cufftHandle plan, cufftReal *idata,
              cufftComplex *odata );
```

executes a CUFFT single‐precision real‐to‐complex (implicitly forward) transform plan. CUFFT uses as input data the GPU memory pointed to by the idata parameter. This function stores the non‐redundant Fourier coefficients in the odata array. If idata and odata are the same, this method does an in‐place transform (See "CUFFT Transform Types" on page 3 for details on real data FFTs.)

Input

plan	The cufftHandle object for the plan to update
idata	Pointer to the single-precision real input data (in GPU memory) to transform
odata	Pointer to the single-precision complex output data (in GPU memory)

Output

odata	Contains the complex Fourier coefficients

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_INVALID_PLAN	The plan parameter is not a valid handle.
CUFFT_INVALID_VALUE	The idata and/or odata parameter is not valid.

Return Values (continued)

CUFFT_EXEC_FAILED	CUFFT failed to execute the transform on GPU.
CUFFT_SUCCESS	CUFFT successfully executed the FFT plan.

Function cufftExecC2R()

```
cufftResult
cufftExecC2R( cufftHandle plan, cufftComplex *idata,
              cufftReal *odata );
```

executes a CUFFT single-precision complex-to-real (implicitly inverse) transform plan. CUFFT uses as input data the GPU memory pointed to by the `idata` parameter. The input array holds only the non-redundant complex Fourier coefficients. This function stores the real output values in the `odata` array. If `idata` and `odata` are the same, this method does an in-place transform. (See “CUFFT Transform Types” on page 3 for details on real data FFTs.)

Input

`plan`	The `cufftHandle` object for the plan to update
`idata`	Pointer to the single-precision complex input data (in GPU memory) to transform
`odata`	Pointer to the single-precision real output data (in GPU memory)

Output

`odata`	Contains the real-valued output data

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_INVALID_PLAN	The `plan` parameter is not a valid handle.
CUFFT_INVALID_VALUE	The `idata` and/or `odata` parameter is not valid.
CUFFT_EXEC_FAILED	CUFFT failed to execute the transform on GPU.
CUFFT_SUCCESS	CUFFT successfully executed the FFT plan.

Function cufftExecZ2Z()

```
cufftResult
cufftExecZ2Z( cufftHandle plan,
              cufftDoubleComplex *idata,
              cufftDoubleComplex *odata, int direction );
```

executes a CUFFT double-precision complex-to-complex transform plan as specified by `direction`. CUFFT uses as input data the GPU memory pointed to by the `idata` parameter. This function stores the Fourier coefficients in the `odata` array. If `idata` and `odata` are the same, this method does an in-place transform.

Input

`plan`	The `cufftHandle` object for the plan to update
`idata`	Pointer to the double-precision complex input data (in GPU memory) to transform
`odata`	Pointer to the double-precision complex output data (in GPU memory)
`direction`	The transform direction: `CUFFT_FORWARD` or `CUFFT_INVERSE`

Output

`odata`	Contains the complex Fourier coefficients

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_INVALID_PLAN	The `plan` parameter is not a valid handle.
CUFFT_INVALID_VALUE	The `idata`, `odata`, and/or `direction` parameter is not valid.
CUFFT_EXEC_FAILED	CUFFT failed to execute the transform on GPU.
CUFFT_SUCCESS	CUFFT successfully executed the FFT plan.

Function cufftExecD2Z()

```
cufftResult
cufftExecD2Z( cufftHandle plan, cufftDoubleReal *idata,
              cufftDoubleComplex *odata );
```

executes a CUFFT double-precision real-to-complex (implicitly forward) transform plan. CUFFT uses as input data the GPU memory pointed to by the `idata` parameter. This function stores the non-redundant Fourier coefficients in the `odata` array. If `idata` and `odata` are the same, this method does an in-place transform (See “CUFFT Transform Types” on page 3 for details on real data FFTs.)

Input

`plan`	The `cufftHandle` object for the plan to update
`idata`	Pointer to the double-precision real input data (in GPU memory) to transform
`odata`	Pointer to the double-precision complex output data (in GPU memory)

Output

`odata`	Contains the complex Fourier coefficients

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_INVALID_PLAN	The `plan` parameter is not a valid handle.
CUFFT_INVALID_VALUE	The `idata` and/or `odata` parameter is not valid.
CUFFT_EXEC_FAILED	CUFFT failed to execute the transform on GPU.
CUFFT_SUCCESS	CUFFT successfully executed the FFT plan.

Function cufftExecZ2D()

```
cufftResult
cufftExecZ2D( cufftHandle plan,
              cufftDoubleComplex *idata,
              cufftDoubleReal *odata );
```

executes a CUFFT double-precision complex-to-real (implicitly inverse) transform plan. CUFFT uses as input data the GPU memory pointed to by the `idata` parameter. The input array holds only the non-redundant complex Fourier coefficients. This function stores the real output values in the `odata` array. If `idata` and `odata` are the same, this method does an in-place transform. (See “CUFFT Transform Types” on page 3 for details on real data FFTs.)

Input

`plan`	The `cufftHandle` object for the plan to update

Input (continued)

idata	Pointer to the double-precision complex input data (in GPU memory) to transform
odata	Pointer to the double-precision real output data (in GPU memory)

Output

odata	Contains the real-valued output data

Return Values

CUFFT_SETUP_FAILED	CUFFT library failed to initialize.
CUFFT_INVALID_PLAN	The `plan` parameter is not a valid handle.
CUFFT_INVALID_VALUE	The `idata` and/or `odata` parameter is not valid.
CUFFT_EXEC_FAILED	CUFFT failed to execute the transform on GPU.
CUFFT_SUCCESS	CUFFT successfully executed the FFT plan.

Function cufftSetStream()

```
cufftResult
cufftSetStream( cufftHandle plan, cudaStream_t stream );
```

associates a CUDA stream with a CUFFT plan. All kernel launches made during plan execution are now done through the associated stream, enabling overlap with activity in other streams (for example, data copying). The association remains until the plan is destroyed or the stream is changed with another call to **`cufftSetStream()`**.

Input

plan	The `cufftHandle` object to associate with the stream
stream	A valid CUDA stream created with **`cudaStreamCreate()`** (or `0` for the default stream)

Output

odata	Contains the real-valued output data

Return Values

CUFFT_INVALID_PLAN	The `plan` parameter is not a valid handle.
CUFFT_SUCCESS	The stream was successfully associated with the plan.

Accuracy and Performance

The CUFFT library implements several FFT algorithms, each having different performance and accuracy. The best performance paths correspond to transform sizes that meet two criteria:

- Fit in CUDA's shared memory
- Are powers of a single factor (for example, powers of two)

These transforms are also the most accurate due to the numeric stability of the chosen FFT algorithm. For transform sizes that meet the first criterion but not second, CUFFT uses a more general mixed-radix FFT algorithm that is usually slower and less numerically accurate. Therefore, if possible it is best to use sizes that are powers of two or four, or powers of other small primes (such as, three, five, or seven). In addition, the power-of-two FFT algorithm in CUFFT makes maximum use of shared memory by blocking sub-transforms for signals that do not meet the first criterion.

For transform sizes that do not meet either criteria above, CUFFT uses an out-of-place, mixed-radix algorithm that stores all intermediate results in CUDA's global GPU memory. Although this algorithm uses optimized transform modules for many factors, it has generally lower performance because global memory has less bandwidth than shared memory. The one exception is large 1D transforms, where CUFFT uses a distributed algorithm that performs a 1D FFT using a 2D FFT, where the dimensions of the 2D transform are factors of the 1D size. This path attempts to utilize the faster transforms mentioned above even if the signal size is too large to fit in CUDA's shared memory.

Many FFT algorithms for real data exploit the conjugate symmetry property to reduce computation and memory cost by roughly half. However, CUFFT does not implement any specialized algorithms for real data, and so there is no direct performance benefit to using real-to-complex (or complex-to-real) plans instead of complex-to-complex. For this release, the real data API exists primarily for convenience, so that users do not have to build interleaved complex data from a real data source before using the library. For 1D transforms, the performance for real data will either match or be less than the complex equivalent (due to an extra copy in come cases). However, there is usually a performance benefit to using real data for 2D and 3D FFTs, since all transforms but the last dimension operate on roughly half the logical signal size

CUFFT Code Examples

This section provides simple examples of 1D, 2D, and 3D complex and real data transforms that use the CUFFT to perform forward and inverse FFTs.

1D Complex-to-Complex Transforms

```
#define NX 256
#define BATCH 10

cufftHandle plan;
cufftComplex *data;
cudaMalloc((void**)&data, sizeof(cufftComplex)*NX*BATCH);

/* Create a 1D FFT plan. */
cufftPlan1d(&plan, NX, CUFFT_C2C, BATCH);

/* Use the CUFFT plan to transform the signal in place. */
cufftExecC2C(plan, data, data, CUFFT_FORWARD);

/* Inverse transform the signal in place. */
cufftExecC2C(plan, data, data, CUFFT_INVERSE);

/* Note:
(1) Divide by number of elements in data set to get back original data
(2) Identical pointers to input and output arrays implies in-place
    transformation
*/

/* Destroy the CUFFT plan. */
cufftDestroy(plan);
cudaFree(data);
```

1D Real-to-Complex Transforms

```
#define NX 256
#define BATCH 10

cufftHandle plan;
cufftComplex *data;
cudaMalloc((void**)&data, sizeof(cufftComplex)*(NX/2+1)*BATCH);

/* Create a 1D FFT plan. */
cufftPlan1d(&plan, NX, CUFFT_R2C, BATCH);

/* Use the CUFFT plan to transform the signal in place. */
cufftExecR2C(plan, (cufftReal*)data, data);

/* Destroy the CUFFT plan. */
cufftDestroy(plan);
cudaFree(data);
```

2D Complex-to-Complex Transforms

```
#define NX 256
#define NY 128

cufftHandle plan;
cufftComplex *idata, *odata;
cudaMalloc((void**)&idata, sizeof(cufftComplex)*NX*NY);
cudaMalloc((void**)&odata, sizeof(cufftComplex)*NX*NY);

/* Create a 2D FFT plan. */
cufftPlan2d(&plan, NX, NY, CUFFT_C2C);

/* Use the CUFFT plan to transform the signal out of place. */
cufftExecC2C(plan, idata, odata, CUFFT_FORWARD);

/* Note: idata != odata indicates an out-of-place transformation
         to CUFFT at execution time. */
```

```
/* Inverse transform the signal in place */
cufftExecC2C(plan, odata, odata, CUFFT_INVERSE);

/* Destroy the CUFFT plan. */
cufftDestroy(plan);
cudaFree(idata); cudaFree(odata);
```

2D Complex-to-Real Transforms

```
#define NX 256
#define NY 128

cufftHandle plan;
cufftComplex *idata;
cufftReal *odata;
cudaMalloc((void**)&idata, sizeof(cufftComplex)*NX*NY);
cudaMalloc((void**)&odata, sizeof(cufftReal)*NX*NY);

/* Create a 2D FFT plan. */
cufftPlan2d(&plan, NX, NY, CUFFT_C2R);

/* Use the CUFFT plan to transform the signal out of place. */
cufftExecC2R(plan, idata, odata);

/* Destroy the CUFFT plan. */
cufftDestroy(plan);
cudaFree(idata); cudaFree(odata);
```

3D Complex-to-Complex Transforms

```
#define NX 64
#define NY 64
#define NZ 128

cufftHandle plan;
cufftComplex *data1, *data2;
cudaMalloc((void**)&data1, sizeof(cufftComplex)*NX*NY*NZ);
cudaMalloc((void**)&data2, sizeof(cufftComplex)*NX*NY*NZ);

/* Create a 3D FFT plan. */
cufftPlan3d(&plan, NX, NY, NZ, CUFFT_C2C);

/* Transform the first signal in place. */
cufftExecC2C(plan, data1, data1, CUFFT_FORWARD);

/* Transform the second signal using the same plan. */
cufftExecC2C(plan, data2, data2, CUFFT_FORWARD);

/* Destroy the CUFFT plan. */
cufftDestroy(plan);
cudaFree(data1); cudaFree(data2);
```

Batched 2D Complex-to-Complex Transforms

```
#define NX 128
#define NY 256
#define BATCHSIZE 1000

int datalen;
cufftHandle plan;
cufftComplex *indata, *outdata;

datalen = NX * NY * BATCHSIZE;
cudaMalloc((void **)&indata, sizeof(cufftComplex)*datalen);
cudaMalloc((void **)&outdata, sizeof(cufftComplex)*datalen);

/* Create a batched 2D plan */
cufftPlanMany(plan,{ NX, NY },NULL,1,0,NULL,1,0,CUFFT_C2C,BATCHSIZE);

/* Execute the transform out-of-place */
cufftExecC2C(plan, indata, outdata, CUFFT_FORWARD);

/* Destroy the CUFFT plan */
cufftDestroy(plan);
cudaFree(indata);
cudaFree(outdata);
```