CUDA

CUFFT Library

PG-00000-003_V03
February, 2007

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

Notice

This source code is subject to NVIDIA ownership rights under U.S. and international Copyright laws.

This software and the information contained herein is PROPRIETARY and CONFIDENTIAL to NVIDIA and is being provided under the terms and conditions of a Non-Disclosure Agreement. Any reproduction or disclosure to any third party without the express written consent of NVIDIA is prohibited.

NVIDIA MAKES NO REPRESENTATION ABOUT THE SUITABILITY OF THIS SOURCE CODE FOR ANY PURPOSE. IT IS PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY OF ANY KIND. NVIDIA DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOURCE CODE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL NVIDIA BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOURCE CODE.

U.S. Government End Users. This source code is a "commercial item" as that term is defined at 48 C.F.R. 2.101 (OCT 1995), consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (SEPT 1995) and is provided to the U.S. Government only as a commercial end item. Consistent with 48 C.F.R.12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (JUNE 1995), all U.S. Government End Users acquire the source code with only those rights set forth herein.

Trademarks

NVIDIA, CUDA, and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

Table of Contents

CUFFT Library

This document describes CUFFT, the NVIDIA® CUDA™ (compute unified device architecture) 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-valued signal data.
- Batch execution for doing multiple 1D transforms in parallel.
- Transform sizes (in any dimension) in the range `[2, 16384]`.

CUFFT Types and Definitions

There are three CUFFT types, as well as transform direction definitions:

- "Type cufftHandle" on page 2
- "Type cufftResult" on page 2
- "Type cufftComplex" on page 2
- "CUFFT Transform Directions" on page 2

Type cufftHandle

`typedef unsigned int cufftHandle;`

is a handle type used to store and access CUFFT plans. For example, the user receives a handle after creating a CUFFT plan and uses this handle to execute the plan.

Type cufftResult

`typedef unsigned int cufftResult;`

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

Type cufftComplex

`typedef float cufftComplex[2];`

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

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 size parameters to the plan creation API functions.

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 possibly 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 4
- ❑ "Function cufftPlan2d()" on page 4
- ❑ "Function cufftPlan3d()" on page 5
- ❑ "Function cufftDestroy()" on page 6
- ❑ "Function cufftExecute()" on page 6

Function cufftPlan1d()

```
cufftResult
cufftPlan1d( cufftHandle *plan, int nx, int 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_DATA_C2C` for 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,
             int 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
`ny`	The transform size in the *Y* dimension
`type`	The transform data type (e.g., `CUFFT_DATA_C2C` for complex)

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,
             int 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_DATA_C2C` for 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 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 shutdown.
`CUFFT_INVALID_PLAN`	The `plan` parameter is not a valid handle.
`CUFFT_SUCCESS`	CUFFT successfully destroyed the FFT plan.

Function cufftExecute()

```
cufftResult
cufftExecute( cufftHandle plan, void *idata, void *odata,
              int sign );
```

executes a CUFFT transform plan. 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 input data (in GPU memory) to transform
`odata`	Pointer to the output data (in GPU memory)
`sign`	The transform direction: `CUFFT_FORWARD` or `CUFFT_INVERSE`

Output

`odata`	Contains the Fourier coefficients

Return Values

`CUFFT_SETUP_FAILED`	CUFFT library failed to initialize.
`CUFFT_INVALID_PLAN`	The `plan` parameter is not a valid handle.

Return Values (continued)

CUFFT_INVALID_VALUE	The data and/or sign parameter is not valid.
CUFFT_EXEC_FAILED	CUFFT failed to execute the transform on GPU.
CUFFT_SUCCESS	CUFFT successfully executed the FFT plan.

CUFFT Code Examples

This section provides simple examples of 1D, 2D, and 3D complex transforms that use the CUFFT to perform forward and inverse FFTs. In the examples, pointers are assumed to point to signal data previously allocated on the GPU.

1D Complex Transforms

```
#define NX 256
#define BATCH 10

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

/* Create a 1D FFT plan. */
cufftPlan1d(&plan, NX, CUFFT_DATA_C2C, BATCH);

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

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

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

2D Complex Transforms

```
#define NX 200
#define NY 100

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

/* Create a 2D FFT plan. */
cufftPlan2d(&plan, NX, NY, CUFFT_DATA_C2C);

/* Use the CUFFT plan to transform the signal out of place.
*/
cufftExecute(plan, data1, data2, CUFFT_FORWARD);

/* Inverse transform the signal in place */
cufftExecute(plan, data2, data2, CUFFT_INVERSE);

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

3D Complex Transforms

```
#define NX 64
#define NY 80
#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_DATA_C2C);

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

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

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