

matCUDA User Guide

Version 0.1, November 2009

Contents

Contents	2
1 CUBLAS functions	5
2 CUFFT functions	9
3 Function Reference	11
3.1 Functions - by category	11
3.1.1 CUBLAS functions	11
3.1.2 CUFFT functions	12
3.2 Functions - alphabetical list	13
3.2.1 cublasAlloc	13
3.2.2 cublasCgemm	14
3.2.3 cublasCheckStatus	15
3.2.4 cublasError	15
3.2.5 cublasFree	16
3.2.6 cublasGetError	17
3.2.7 cublasGetVector	18
3.2.8 cublasInit	19
3.2.9 cublasIsamax	20
3.2.10 cublasIsamin	21
3.2.11 cublasResult	21
3.2.12 cublasSasum	22
3.2.13 cublasSaxpy	23
3.2.14 cublasScopy	24
3.2.15 cublasSdot	25
3.2.16 cublasSetVector	26
3.2.17 cublasSgemm	27
3.2.18 cublasShutdown	28
3.2.19 cublasSnrm2	28
3.2.20 cublasSrot	29

3.2.21	cublasSscal	29
3.2.22	cufftCheckStatus	30
3.2.23	cufftDestroy	31
3.2.24	cufftExecC2C	32
3.2.25	cufftExecC2R	33
3.2.26	cufftExecR2C	33
3.2.27	cufftPlan1d	34
3.2.28	cufftPlan2d	35
3.2.29	cufftResult	35
3.2.30	cufftTransformDirections	36
3.2.31	cufftType	36

The *matCUDA* project on Sourceforge(<http://sourceforge.net/projects/matcuda/>) was started by the *GP-you Group* as part of the Matlab toolbox GPUmat (<http://gp-you.org>). This project is a collection of Matlab wrappers to the CUDA CUBLAS and CUFFT libraries. To install the binaries, the user has to do the following:

- (Optional) Download and install GPUmat (<http://gp-you.org>).
- Download the binaries from Sourceforge. Extract them to the folder where GPUmat was installed or to any other folder. Make sure that the extracted folders are available in the Matlab path.

Although *matCUDA* wrappers can be used independently from GPUmat, next sections explain how to use them from GPUmat.

Chapter 1

CUBLAS functions

The following code shows how to use low level CUBLAS functions using `matCUDA` wrappers. The code can be found in the file `simpleCUBLAS.m` located in the examples folder `CUBLAS`. Make sure that the GPU environment was started using `GPUstart` before running the example.

```
function simpleCUBLAS
% This is the GPUmat translation of the code in the
% CUDA SDK projects called with the same name (simpleCUBLAS).
% The example shows how to access CUBLAS functions from GPUmat

SIZEOF_FLOAT = sizeof(float());

%% Allocate HOST arrays and initialize with random numbers
N = 500;

h_A = single(rand(N));
h_B = single(rand(N));
h_C = single(rand(N));

%% Allocate GPU arrays
d_A = GPUsingle(h_A);
d_B = GPUsingle(h_B);
d_C = GPUsingle(h_C);

% Although d_A was already initialized with h_A values, we can
% call cublasSetVector to do that again
status = cublasSetVector(N*N, SIZEOF_FLOAT, ...
                        h_A, 1, getPtr(d_A), 1);
cublasCheckStatus( status, '!!!! device access error (write A)');
```

```
% Calculate reference in Matlab
alpha = 2.0;
h_C_ref = alpha * h_A*h_B;

% Execute on GPU
cublasSgemm('n','n', N, N, N, alpha, getPtr(d_A), ...
           N, getPtr(d_B), N, 0.0, getPtr(d_C), N);
status = cublasGetError();
cublasCheckStatus( status, '!!!! kernel execution error.');
```

```
% Copy results back to HOST
h_C = single(d_C);

compareArrays(h_C_ref, h_C, 5e-6);

% Clean up GPU memory
% THERE IS NO NEED TO CLEAN UP MEMORY
% NEVERTHELESS, IF NECESSARY, ALWAYS USE
% CLEAR WITH GPUSINGLE
clear d_A
clear d_B
clear d_C

end
```

matCUDA defines wrappers to CUBLAS functions. The list of these functions can be found in the function reference Section (CUBLAS Function Reference). Some examples can be found in the **example** folder **CUBLAS**. In general CUBLAS wrappers have the same interface as the original CUBLAS functions. When a CUBLAS function needs a pointer to a GPU variable **A**, the pointer is obtained using **getPtr(A)**. For example:

```
A = GPUsingle(rand(1,100));
r = cublasIsamax(numel(A),getPtr(A),1)
```

The original declaration of the CUBLAS function **cublasIsamax** is:

```
int
cublasIsamax (int n, const float *x, int incx)
```

Note the mapping between variables in the above example:

```
int n          -> numel(A)
const float *x -> getPtr(A)
int incx       -> 1
```

The following code performs complex matrix-matrix multiplication using **cublasCgemm**:

```
N = 10;
I = sqrt(-1);
A = GPUsingle(rand(N,N) + I*rand(N,N));
B = GPUsingle(rand(N,N) + I*rand(N,N));

% C needs to be complex as well, thats why we multiply by I
C = zeros(N,N,GPUsingle)*I;

% alpha is complex
alpha = 2.0+I*3.0;
beta  = 0.0;

opA = 'n';
opB = 'n';

cublasCgemm(opA, opB, N, N, N, ...
    alpha, getPtr(A), N, getPtr(B), ...
    N, beta, getPtr(C), N);

status = cublasGetError();
ret = cublasCheckStatus( status, ...
    '!!!! kernel execution error.');
```

```
C_mat = alpha * single(A)*single(B);
compareArrays(C_mat, single(C), 1e-6);
```

The original declaration of the CUBLAS function **cublasCgemm** is:

```
void cublasCgemm (char transa, char transb, int m, int n, int k,
    cuComplex alpha, const cuComplex *A, int lda,
    const cuComplex *B, int ldb, cuComplex beta,
    cuComplex *C, int ldc)
```

Please note the mapping between variables in the above example:

```
char transa      -> 'n'  
char transb      -> 'n'  
int m            -> N  
int n            -> N  
int k            -> N  
cuComplex alpha  -> 2.0+I*3.0  
const cuComplex *A -> getPtr(d_A)  
int lda          -> N  
const cuComplex *B -> getPtr(d_B)  
int ldb          -> N  
cuComplex beta   -> 0.0  
cuComplex *C     -> getPtr(d_C)  
int ldc          -> N
```

Complex numbers are stored interleaving real and imaginary values on the GPU, the same format expected by the **cublasCgemm** function and other CUFFT functions. For a complete description of CUBLAS functions check the CUDA CUBLAS manual. For a complete list of implemented wrappers check the functions reference Section (CUBLAS Function Reference).

Chapter 2

CUFFT functions

The following code shows how to call low level CUFFT functions using `matCUDA` wrappers. The code can be found in the file `simpleCUFFT.m` located in the examples folder `CUFFT`. Make sure that the GPU environment was started using `GPUstart` before testing the example.

```
%% CUFFT example

%% Allocate HOST arrays and initialize with random numbers
N = 512;

h_A = single(rand(1,N)+i*rand(1,N));

d_A = GPUsingle(h_A);
d_B = GPUsingle(h_A);

fftType = cufftType;
fftDir = cufftTransformDirections;

% FFT plan
plan = 0;
[status, plan] = cufftPlan1d(plan, numel(d_A), ...
                             fftType.CUFFT_C2C, 1);
cufftCheckStatus(status, 'Error in cufftPlan1D');

% Run GPU FFT
[status] = cufftExecC2C(plan, getPtr(d_A), getPtr(d_B), ...
                        fftDir.CUFFT_FORWARD);
cufftCheckStatus(status, 'Error in cufftExecC2C');

% Run GPU IFFT
```

```
[status] = cufftExecC2C(plan, getPtr(d_B), getPtr(d_A), ...
                        fftDir.CUFFT_INVERSE);
cufftCheckStatus(status, 'Error in cufftExecC2C');

% results should be scaled by 1/N if compared to CPU
h_B = 1/N*single(d_A);

[status] = cufftDestroy(plan);
cufftCheckStatus(status, 'Error in cuffDestroyPlan');
```

Chapter 3

Function Reference

3.1 Functions - by category

3.1.1 CUBLAS functions

Name	Description
<code>cublasAlloc</code>	Wrapper to CUBLAS <code>cublasAlloc</code> function
<code>cublasCgemm</code>	Wrapper to CUBLAS <code>cublasCgemm</code> function
<code>cublasCheckStatus</code>	Check the CUBLAS status.
<code>cublasError</code>	Returns a structure with CUBLAS result codes
<code>cublasFree</code>	Wrapper to CUBLAS <code>cublasFree</code> function
<code>cublasGetError</code>	Wrapper to CUBLAS <code>cublasGetError</code> function
<code>cublasGetVector</code>	Wrapper to CUBLAS <code>cublasGetVector</code> function
<code>cublasInit</code>	Wrapper to CUBLAS <code>cublasInit</code> function
<code>cublasIsamax</code>	Wrapper to CUBLAS <code>cublasIsamax</code> function
<code>cublasIsamin</code>	Wrapper to CUBLAS <code>cublasIsamin</code> function
<code>cublasResult</code>	Returns a structure with CUBLAS error results
<code>cublasSasum</code>	Wrapper to CUBLAS <code>cublasSasum</code> function
<code>cublasSaxpy</code>	Wrapper to CUBLAS <code>cublasSaxpy</code> function
<code>cublasScopy</code>	Wrapper to CUBLAS <code>cublasScopy</code> function
<code>cublasSdot</code>	Wrapper to CUBLAS <code>cublasSdot</code> function
<code>cublasSetVector</code>	Wrapper to CUBLAS <code>cublasSetVector</code> function
<code>cublasSgemm</code>	Wrapper to CUBLAS <code>cublasSgemm</code> function

<code>cublasShutdown</code>	Wrapper to CUBLAS <code>cublasShutdown</code> function
<code>cublasSnrm2</code>	Wrapper to CUBLAS <code>cublasSnrm2</code> function
<code>cublasSrot</code>	Wrapper to CUBLAS <code>cublasSrot</code> function
<code>cublasSscal</code>	Wrapper to CUBLAS <code>cublasSscal</code> function

3.1.2 CUFFT functions

Name	Description
<code>cufftCheckStatus</code>	Checks the CUFFT status
<code>cufftDestroy</code>	Wrapper to CUFFT <code>cufftDestroy</code> function
<code>cufftExecC2C</code>	Wrapper to CUFFT <code>cufftExecC2C</code> function
<code>cufftExecC2R</code>	Wrapper to CUFFT <code>cufftExecC2R</code> function
<code>cufftExecR2C</code>	Wrapper to CUFFT <code>cufftExecR2C</code> function
<code>cufftPlan1d</code>	Wrapper to CUFFT <code>cufftPlan1d</code> function
<code>cufftPlan2d</code>	Wrapper to CUFFT <code>cufftPlan2d</code> function
<code>cufftResult</code>	Returns a structure with CUFFT result codes
<code>cufftTransformDirections</code>	Returns a structure with CUFFT transform direction codes
<code>cufftType</code>	Returns a structure with CUFFT transform type codes

3.2 Functions - alphabetical list

3.2.1 cublasAlloc

cublasAlloc - Wrapper to CUBLAS cublasAlloc function

SYNTAX

```
[status d_A] = cublasAlloc(N,SIZE,d_A);  
N - number of elements to allocate  
SIZE - size of the elements to allocate  
d_A - pointer to GPU memory  
status - CUBLAS status  
d_A - pointer to GPU memory
```

DESCRIPTION

Wrapper to CUBLAS cublasAlloc function.

Original function declaration:

```
cublasStatus  
cublasAlloc (int n, int elemSize, void **devicePtr)
```

Mapping:

```
[status d_A] = cublasAlloc(N, SIZE, d_A)  
N -> int n  
SIZE -> int elemSize  
d_A -> void **devicePtr
```

```
status -> cublasStatus
```

EXAMPLE

```
N = 10;  
SIZEOF_FLOAT = sizeof(float);  
% GPU variable d_A  
d_A = 0;  
[status d_A] = cublasAlloc(N,SIZEOF_FLOAT,d_A);  
ret = cublasCheckStatus( status, ...  
    '!!!! device memory allocation error (d_A)');
```

3.2.2 cublasCgemm

cublasCgemm - Wrapper to CUBLAS cublasCgemm function

DESCRIPTION

Wrapper to CUBLAS cublasCgemm function. Original function declaration:

```
void cublasCgemm
(char transa, char transb, int m, int n, int k,
 cuComplex alpha, const cuComplex *A, int lda,
 const cuComplex *B, int ldb, cuComplex beta,
 cuComplex *C, int ldc)
```

EXAMPLE

```
I = sqrt(-1);
N = 10;
A = GPUsingle(rand(N,N) + I*rand(N,N));
B = GPUsingle(rand(N,N) + I*rand(N,N));
% C needs to be complex as well
C = zeros(N,N,GPUsingle)*I;

% alpha is complex
alpha = 2.0+I*3.0;
beta = 0.0;

opA = 'n';
opB = 'n';

cublasCgemm(opA, opB, N, N, N, ...
  alpha, getPtr(A), N, getPtr(B), ...
  N, beta, getPtr(C), N);

status = cublasGetError();
ret = cublasCheckStatus( status, ...
  '!!!! kernel execution error.');
```

3.2.3 `cublasCheckStatus`

`cublasCheckStatus` - Check the CUBLAS status.

DESCRIPTION

`cublasCheckStatus(STATUS,MSG)` returns `EXIT_FAILURE(1)` or `EXIT_SUCCESS(0)` depending on `STATUS` value, and throws an error with message 'MSG'.

EXAMPLE

```
status = cublasGetError();  
cublasCheckStatus( status, 'Kernel execution error');
```

3.2.4 `cublasError`

`cublasError` - Returns a structure with CUBLAS result codes

DESCRIPTION

Returns a structure with CUBLAS result codes.

3.2.5 cublasFree

cublasFree - Wrapper to CUBLAS cublasFree function

DESCRIPTION

Wrapper to CUBLAS cublasFree function.

Original function declaration:

```
cublasStatus  
cublasFree (const void *devicePtr)
```

Mapping:

```
status = cublasFree(d_A)  
d_A -> const void *devicePtr
```

```
status -> cublasStatus
```

EXAMPLE

```
N = 10;  
SIZEOF_FLOAT = sizeof(float());  
  
% GPU variable d_A  
d_A = 0;  
[status d_A] = cublasAlloc(N,SIZEOF_FLOAT,d_A);  
ret = cublasCheckStatus( status, ...  
    '!!!! device memory allocation error (d_A)');  
  
% Clean up memory  
status = cublasFree(d_A);  
ret = cublasCheckStatus( status, ...  
    '!!!! memory free error (d_A)');
```

3.2.6 `cublasGetError`

`cublasGetError` - Wrapper to CUBLAS `cublasGetError` function

DESCRIPTION

Wrapper to CUBLAS `cublasGetError` function. Original function declaration:

```
cublasStatus  
cublasGetError (void)
```

EXAMPLE

```
status = cublasGetError();  
cublasCheckStatus( status, 'Kernel execution error');
```

3.2.7 cublasGetVector

cublasGetVector - Wrapper to CUBLAS cublasGetVector function

DESCRIPTION

Wrapper to CUBLAS cublasGetVector function. Original function declaration:

```
cublasStatus  
cublasGetVector  
(int n, int elemSize, const void *x, int incx,  
 void *y, int incy)
```

EXAMPLE

```
A = GPUsingle([1 2 3 4]);  
  
% Ah should be of the correct type. GPUsingle is single  
% precision floating point, also Ah should be single  
% precision  
Ah = single(zeros(size(A)));  
  
% The function getSizeOf returns the size of the  
% stored elements in A (for example float or complex)  
[status Ah] = cublasGetVector (numel(A), getSizeOf(A),...  
                             getPtr(A), 1, Ah, 1);  
cublasCheckStatus( status, 'Error.');
```

3.2.8 cublasInit

cublasInit - Wrapper to CUBLAS cublasInit function

DESCRIPTION

Wrapper to CUBLAS cublasInit function. Original function declaration:

```
cublasStatus  
cublasInit (void)
```

EXAMPLE

```
status = cublasInit;  
cublasCheckStatus(status, 'Error.');
```

3.2.9 cublasIsamax

cublasIsamax - Wrapper to CUBLAS cublasIsamax function

DESCRIPTION

Wrapper to CUBLAS cublasIsamax function. Original function declaration:

```
int  
cublasIsamax (int n, const float *x, int incx)
```

Mapping:

```
RET = cublasIsamax(N, d_A, INCX)  
N    -> int n  
d_A  -> void **devicePtr  
INCX -> int incx
```

RET -> cublasIsamax result

EXAMPLE

```
N = 10;  
A = GPUsingle(rand(1,N));  
  
Isamax = cublasIsamax(N, getPtr(A), 1);  
status = cublasGetError();  
ret = cublasCheckStatus( status, ...  
    '!!!! kernel execution error.');
```

```
[value, Isamax_mat] = max(single(A));  
compareArrays(Isamax, Isamax_mat, 1e-6);
```

3.2.10 cublasIsamin

cublasIsamin - Wrapper to CUBLAS cublasIsamin function

DESCRIPTION

Wrapper to CUBLAS cublasIsamin function. Original function declaration:

```
int  
cublasIsamin (int n, const float *x, int incx)
```

EXAMPLE

```
N = 10;  
A = GPUsingle(rand(1,N));  
  
Isamin = cublasIsamin(N, getPtr(A), 1);  
status = cublasGetError();  
ret = cublasCheckStatus( status, ...  
    '!!!! kernel execution error.');
```

```
[value, Isamin_mat] = min(single(A));  
compareArrays(Isamin, Isamin_mat, 1e-6);
```

3.2.11 cublasResult

cublasResult - Returns a structure with CUBLAS error results

DESCRIPTION

Returns a structure with CUBLAS error results.

3.2.12 cublasSasum

cublasSasum - Wrapper to CUBLAS cublasSasum function

DESCRIPTION

Wrapper to CUBLAS cublasSasum function.

Original function declaration:

```
float  
cublasSasum (int n, const float *x, int incx)
```

EXAMPLE

```
N = 10;  
A = GPUsingle(rand(1,N));  
  
Sasum = cublasSasum( N, getPtr(A), 1);  
status = cublasGetError();  
ret = cublasCheckStatus( status, ...  
    '!!!! kernel execution error.');
```

```
Sasum_mat = sum(abs(single(A)));  
compareArrays(Sasum, Sasum_mat, 1e-6);
```

3.2.13 cublasSaxpy

cublasSaxpy - Wrapper to CUBLAS cublasSaxpy function

DESCRIPTION

Wrapper to CUBLAS cublasSaxpy function. Original function declaration:

```
void  
cublasSaxpy  
(int n, float alpha, const float *x, int incx, float *y,  
 int incy)
```

EXAMPLE

```
N = 10;  
A = GPUsingle(rand(1,N));  
B = GPUsingle(rand(1,N));  
  
alpha = 2.0;  
Saxpy_mat = alpha * single(A) + single(B);  
  
cublasSaxpy(N, alpha, getPtr(A), 1, getPtr(B), 1);  
  
status = cublasGetError();  
ret = cublasCheckStatus( status, ...  
    '!!!! kernel execution error.');
```

```
compareArrays(Saxpy_mat, single(B), 1e-6);
```

3.2.14 cublasScopy

cublasScopy - Wrapper to CUBLAS cublasScopy function

DESCRIPTION

Wrapper to CUBLAS cublasScopy function. Original function declaration:

```
void  
cublasScopy  
(int n, const float *x, int incx, float *y, int incy)
```

EXAMPLE

```
N = 10;  
A = GPUsingle(rand(1,N));  
B = GPUsingle(rand(1,N));  
  
cublasScopy(N, getPtr(A), 1, getPtr(B), 1);  
status = cublasGetError();  
ret = cublasCheckStatus( status, ...  
    '!!!! kernel execution error.');
```

```
compareArrays(single(A), single(B), 1e-6);
```

3.2.15 cublasSdot

cublasSdot - Wrapper to CUBLAS cublasSdot function

DESCRIPTION

Wrapper to CUBLAS cublasSdot function. Original function declaration:

```
float  
cublasSdot  
(int n, const float *x, int incx, const float *y, int incy)
```

EXAMPLE

```
N = 10;  
A = GPUsingle(rand(1,N));  
B = GPUsingle(rand(1,N));  
  
Sdot_mat = sum(single(A).*single(B));  
Sdot = cublasSdot(N, getPtr(A), 1, getPtr(B), 1);  
  
status = cublasGetError();  
ret = cublasCheckStatus( status, ...  
    '!!!! kernel execution error.');
```

```
compareArrays(Sdot_mat, Sdot, 1e-6);
```

3.2.16 `cublasSetVector`

`cublasSetVector` - Wrapper to CUBLAS `cublasSetVector` function

DESCRIPTION

Wrapper to CUBLAS `cublasSetVector` function. Original function declaration:

```
cublasStatus  
cublasSetVector  
(int n, int elemSize, const void *x, int incx,  
 void *y, int incy)
```

EXAMPLE

```
B =single( [1 2 3 4]);  
  
% Create empty GPU variable A  
A = GPUsingle();  
setSize(A, size(B));  
GPUallocVector(A);  
  
status = cublasSetVector(numel(A), getSizeOf(A), ...  
B, 1, getPtr(A), 1);  
cublasCheckStatus( status, 'Error.');
```

```
disp(single(A));
```

3.2.17 cublasSgemm

cublasSgemm - Wrapper to CUBLAS cublasSgemm function

DESCRIPTION

Wrapper to CUBLAS cublasSgemm function. Original function declaration:

```
void  
cublasSgemm  
(char transa, char transb, int m, int n, int k,  
 float alpha, const float *A, int lda,  
 const float *B, int ldb, float beta,  
 float *C, int ldc)
```

EXAMPLE

```
N = 10;  
A = GPUsingle(rand(N,N));  
B = GPUsingle(rand(N,N));  
C = zeros(N,N,GPUsingle);  
  
alpha = 2.0;  
beta = 0.0;  
  
opA = 'n';  
opB = 'n';  
  
cublasSgemm(opA, opB, N, N, N, ...  
  alpha, getPtr(A), N, getPtr(B), ...  
  N, beta, getPtr(C), N);  
  
status = cublasGetError();  
ret = cublasCheckStatus( status, ...  
  '!!!! kernel execution error.');
```

3.2.18 `cublasShutdown`

`cublasShutdown` - Wrapper to CUBLAS `cublasShutdown` function

DESCRIPTION

Wrapper to CUBLAS `cublasShutdown` function. Original function declaration:

```
cublasStatus  
cublasShutdown (void)
```

3.2.19 `cublasSnrm2`

`cublasSnrm2` - Wrapper to CUBLAS `cublasSnrm2` function

DESCRIPTION

Wrapper to CUBLAS `cublasSnrm2` function. Original function declaration:

```
float  
cublasSnrm2 (int n, const float *x, int incx)
```

EXAMPLE

```
N = 10;  
A = GPUsingle(rand(1,N));  
  
Snrm2_mat = sqrt(sum(single(A).*single(A)));  
Snrm2 = cublasSnrm2(N, getPtr(A),1);  
  
status = cublasGetError();  
ret = cublasCheckStatus( status, ...  
    '!!!! kernel execution error.');
```

3.2.20 **cublasSrot**

cublasSrot - Wrapper to CUBLAS cublasSrot function

DESCRIPTION

Wrapper to CUBLAS cublasSrot function.

Original function declaration:

```
void
cublasSrot (int n, float *x, int incx,
            float *y, int incy, float sc,
            float ss)
```

3.2.21 **cublasSscal**

cublasSscal - Wrapper to CUBLAS cublasSscal function

DESCRIPTION

Wrapper to CUBLAS cublasSscal function.

Original function declaration:

```
void
sscal (int n, float alpha, float *x, int incx)
```

EXAMPLE

```
N = 10;
A = GPUsingle(rand(1,N));

alpha = 1/10.0;
A_mat = single(A)*alpha;
cublasSscal(N, alpha, getPtr(A), 1);

status = cublasGetError();
ret = cublasCheckStatus( status, ...
    '!!!! kernel execution error.');
```

3.2.22 cufftCheckStatus

cufftCheckStatus - Checks the CUFFT status

DESCRIPTION

`cufftCheckStatus(STATUS,MSG)` returns `EXIT_FAILURE(1)` or `EXIT_SUCCESS(0)` depending on `STATUS` value, and throws an error with message `'MSG'`. `STATUS` is compared to CUFFT possible results.

EXAMPLE

```
fftType = cufftType;
A = GPUsingle(rand(1,128));
plan = 0;
type = fftType.CUFFT_C2C;
[status, plan] = cufftPlan1d(plan, numel(A), type, 1);
cufftCheckStatus(status, 'Error in cufftPlan1D');
```

3.2.23 cufftDestroy

cufftDestroy - Wrapper to CUFFT cufftDestroy function

DESCRIPTION

Wrapper to CUFFT cufftDestroy function. Original function declaration:

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

EXAMPLE

```
fftType = cufftType;  
I = sqrt(-1);  
A = GPUsingle(rand(1,128)+I*rand(1,128));  
plan = 0;  
type = fftType.CUFFT_C2C;  
[status, plan] = cufftPlan1d(plan, numel(A), type, 1);  
cufftCheckStatus(status, 'Error in cufftPlan1D');  
  
[status] = cufftDestroy(plan);  
cufftCheckStatus(status, 'Error in cuffDestroyPlan');
```

3.2.24 cufftExecC2C

cufftExecC2C - Wrapper to CUFFT cufftExecC2C function

DESCRIPTION

Wrapper to CUFFT cufftExecC2C function. Original function declaration:

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

EXAMPLE

```
fftType = cufftType;  
fftDir  = cufftTransformDirections;  
  
I = sqrt(-1);  
  
A = GPUsingle(rand(1,128)+I*rand(1,128));  
plan = 0;  
type = fftType.CUFFT_C2C;  
[status, plan] = cufftPlan1d(plan, numel(A), type, 1);  
cufftCheckStatus(status, 'Error in cufftPlan1D');  
  
dir = fftDir.CUFFT_FORWARD;  
[status] = cufftExecC2C(plan, getPtr(A),getPtr(A), dir);  
cufftCheckStatus(status, 'Error in cufftExecC2C');  
  
[status] = cufftDestroy(plan);  
cufftCheckStatus(status, 'Error in cuffDestroyPlan');
```

3.2.25 cufftExecC2R

cufftExecC2R - Wrapper to CUFFT cufftExecC2R function

DESCRIPTION

Wrapper to CUFFT cufftExecC2R function. Original function declaration:

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

3.2.26 cufftExecR2C

cufftExecR2C - Wrapper to CUFFT cufftExecR2C function

DESCRIPTION

Wrapper to CUFFT cufftExecR2C function. Original function declaration:

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

3.2.27 cufftPlan1d

cufftPlan1d - Wrapper to CUFFT cufftPlan1d function

DESCRIPTION

Wrapper to CUFFT cufftPlan1d function. Original function declaration:

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

Original function returns only a cufftResult, whereas wrapper returns also the plan.

EXAMPLE

```
fftType = cufftType;  
I = sqrt(-1);  
A = GPUsingle(rand(1,128)+I*rand(1,128));  
plan = 0;  
type = fftType.CUFFT_C2C;  
[status, plan] = cufftPlan1d(plan, numel(A), type, 1);  
cufftCheckStatus(status, 'Error in cufftPlan1D');  
  
[status] = cufftDestroy(plan);  
cufftCheckStatus(status, 'Error in cuffDestroyPlan');
```

3.2.28 cufftPlan2d

cufftPlan2d - Wrapper to CUFFT cufftPlan2d function

DESCRIPTION

Wrapper to CUFFT cufftPlan2d function. Original function declaration:

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

EXAMPLE

```
fftType = cufftType;  
I = sqrt(-1);  
A = GPUSingle(rand(128,128)+I*rand(128,128));  
plan = 0;  
% Vectors stored in column major format (FORTRAN)  
s = size(A);  
type = fftType.CUFFT_C2C;  
[status, plan] = cufftPlan2d(plan, s(2), s(1),type);  
cufftCheckStatus(status, 'Error in cufftPlan2D');  
  
[status] = cufftDestroy(plan);  
cufftCheckStatus(status, 'Error in cuffDestroyPlan');
```

3.2.29 cufftResult

cufftResult - Returns a structure with CUFFT result codes

DESCRIPTION

Returns a structure with CUFFT result codes

3.2.30 `cufftTransformDirections`

`cufftTransformDirections` - Returns a structure with CUFFT transform direction codes

DESCRIPTION

Returns a structure with CUFFT transform direction codes.

CUFFT_FORWARD = -1; Forward FFT

CUFFT_INVERSE = 1; Inverse FFT

3.2.31 `cufftType`

`cufftType` - Returns a structure with CUFFT transform type codes

DESCRIPTION

Returns a structure with CUFFT transform type codes.