# CUBLAS LIBRARY

DU-06702-001_v9.1 | November 2017

## User Guide

# Chapter 1. INTRODUCTION

The cuBLAS library is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA®CUDA™ runtime. It allows the user to access the computational resources of NVIDIA Graphics Processing Unit (GPU).

Starting with CUDA 6.0, the cuBLAS Library now exposes two sets of API, the regular cuBLAS API which is simply called cuBLAS API in this document and the CUBLASXT API.

To use the cuBLAS API, the application must allocate the required matrices and vectors in the GPU memory space, fill them with data, call the sequence of desired cuBLAS functions, and then upload the results from the GPU memory space back to the host. The cuBLAS API also provides helper functions for writing and retrieving data from the GPU.

To use the CUBLASXT API, the application must keep the data on the Host and the Library will take care of dispatching the operation to one or multiple GPUS present in the system, depending on the user request.

## 1.1. Data layout

For maximum compatibility with existing Fortran environments, the cuBLAS library uses column-major storage, and 1-based indexing. Since C and C++ use row-major storage, applications written in these languages can not use the native array semantics for two-dimensional arrays. Instead, macros or inline functions should be defined to implement matrices on top of one-dimensional arrays. For Fortran code ported to C in mechanical fashion, one may chose to retain 1-based indexing to avoid the need to transform loops. In this case, the array index of a matrix element in row "i" and column "j" can be computed via the following macro

```
#define IDX2F(i,j,ld) ((((j)-1)*(ld))+((i)-1))
```

Here, ld refers to the leading dimension of the matrix, which in the case of column-major storage is the number of rows of the allocated matrix (even if only a submatrix of it is being used). For natively written C and C++ code, one would most likely choose 0-based

indexing, in which case the array index of a matrix element in row "i" and column "j" can be computed via the following macro

```
#define IDX2C(i,j,ld) (((j)*(ld))+(i))
```

## 1.2. New and Legacy cuBLAS API

Starting with version 4.0, the cuBLAS Library provides a new updated API, in addition to the existing legacy API. This section discusses why a new API is provided, the advantages of using it, and the differences with the existing legacy API.

The new cuBLAS library API can be used by including the header file "cublas_v2.h". It has the following features that the legacy cuBLAS API does not have:

- the **`handle`** to the cuBLAS library context is initialized using the function and is explicitly passed to every subsequent library function call. This allows the user to have more control over the library setup when using multiple host threads and multiple GPUs. This also allows the cuBLAS APIs to be reentrant.
- the scalars $\alpha$ and $\beta$ can be passed by reference on the host or the device, instead of only being allowed to be passed by value on the host. This change allows library functions to execute asynchronously using streams even when $\alpha$ and $\beta$ are generated by a previous kernel.
- when a library routine returns a scalar result, it can be returned by reference on the host or the device, instead of only being allowed to be returned by value only on the host. This change allows library routines to be called asynchronously when the scalar result is generated and returned by reference on the device resulting in maximum parallelism.
- the error status **`cublasStatus_t`** is returned by all cuBLAS library function calls. This change facilitates debugging and simplifies software development. Note that **`cublasStatus`** was renamed **`cublasStatus_t`** to be more consistent with other types in the cuBLAS library.
- the **`cublasAlloc()`** and **`cublasFree()`** functions have been deprecated. This change removes these unnecessary wrappers around **`cudaMalloc()`** and **`cudaFree()`**, respectively.
- the function **`cublasSetKernelStream()`** was renamed **`cublasSetStream()`** to be more consistent with the other CUDA libraries.

The legacy cuBLAS API, explained in more detail in the Appendix A, can be used by including the header file "cublas.h". Since the legacy API is identical to the previously released cuBLAS library API, existing applications will work out of the box and automatically use this legacy API without any source code changes. In general, new applications should not use the legacy cuBLAS API, and existing existing applications should convert to using the new API if it requires sophisticated and optimal stream parallelism or if it calls cuBLAS routines concurrently from multiple threads. For the rest of the document, the new cuBLAS Library API will simply be referred to as the cuBLAS Library API.

As mentioned earlier the interfaces to the legacy and the cuBLAS library APIs are the header file "cublas.h" and "cublas_v2.h", respectively. In addition, applications using the cuBLAS library need to link against the DSO cublas.so (Linux), the DLL cublas.dll

(Windows), or the dynamic library cublas.dylib (Mac OS X). Note: the same dynamic library implements both the new and legacy cuBLAS APIs.

## 1.3. Example code

For sample code references please see the two examples below. They show an application written in C using the cuBLAS library API with two indexing styles

(Example 1. "Application Using C and CUBLAS: 1-based indexing" and Example 2. "Application Using C and CUBLAS: 0-based Indexing").

```
//Example 1. Application Using C and CUBLAS: 1-based indexing
//-----------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda_runtime.h>
#include "cublas_v2.h"
#define M 6
#define N 5
#define IDX2F(i,j,ld) ((((j)-1)*(ld))+((i)-1))

static __inline__ void modify (cublasHandle_t handle, float *m, int ldm, int
 n, int p, int q, float alpha, float beta){
    cublasSscal (handle, n-p+1, &alpha, &m[IDX2F(p,q,ldm)], ldm);
    cublasSscal (handle, ldm-p+1, &beta, &m[IDX2F(p,q,ldm)], 1);
}

int main (void){
    cudaError_t cudaStat;
    cublasStatus_t stat;
    cublasHandle_t handle;
    int i, j;
    float* devPtrA;
    float* a = 0;
    a = (float *)malloc (M * N * sizeof (*a));
    if (!a) {
        printf ("host memory allocation failed");
        return EXIT_FAILURE;
    }
    for (j = 1; j <= N; j++) {
        for (i = 1; i <= M; i++) {
            a[IDX2F(i,j,M)] = (float)((i-1) * M + j);
        }
    }
    cudaStat = cudaMalloc ((void**)&devPtrA, M*N*sizeof(*a));
    if (cudaStat != cudaSuccess) {
        printf ("device memory allocation failed");
        return EXIT_FAILURE;
    }
    stat = cublasCreate(&handle);
    if (stat != CUBLAS_STATUS_SUCCESS) {
        printf ("CUBLAS initialization failed\n");
        return EXIT_FAILURE;
    }
    stat = cublasSetMatrix (M, N, sizeof(*a), a, M, devPtrA, M);
    if (stat != CUBLAS_STATUS_SUCCESS) {
        printf ("data download failed");
        cudaFree (devPtrA);
        cublasDestroy(handle);
        return EXIT_FAILURE;
    }
    modify (handle, devPtrA, M, N, 2, 3, 16.0f, 12.0f);
    stat = cublasGetMatrix (M, N, sizeof(*a), devPtrA, M, a, M);
    if (stat != CUBLAS_STATUS_SUCCESS) {
        printf ("data upload failed");
        cudaFree (devPtrA);
        cublasDestroy(handle);
        return EXIT_FAILURE;
    }
    cudaFree (devPtrA);
    cublasDestroy(handle);
    for (j = 1; j <= N; j++) {
        for (i = 1; i <= M; i++) {
            printf ("%7.0f", a[IDX2F(i,j,M)]);
        }
        printf ("\n");
    }
    free(a);
    return EXIT_SUCCESS;
}
```

```
//Example 2. Application Using C and CUBLAS: 0-based indexing
//-----------------------------------------------------------
```

# Chapter 2. USING THE CUBLAS API

## 2.1. General description

This section describes how to use the cuBLAS library API. It does not contain a detailed reference for all API datatypes and functions–those are provided in subsequent chapters. The Legacy cuBLAS API is also not covered in this section–that is handled in an Appendix.

### 2.1.1. Error status

All cuBLAS library function calls return the error status **`cublasStatus_t`**.

### 2.1.2. cuBLAS context

The application must initialize the **`handle`** to the cuBLAS library context by calling the **`cublasCreate()`** function. Then, the is explicitly passed to every subsequent library function call. Once the application finishes using the library, it must call the function **`cublasDestory()`** to release the resources associated with the cuBLAS library context.

This approach allows the user to explicitly control the library setup when using multiple host threads and multiple GPUs. For example, the application can use **`cudaSetDevice()`** to associate different devices with different host threads and in each of those host threads it can initialize a unique **`handle`** to the cuBLAS library context, which will use the particular device associated with that host thread. Then, the cuBLAS library function calls made with different **`handle`** will automatically dispatch the computation to different devices.

The device associated with a particular cuBLAS context is assumed to remain unchanged between the corresponding **`cublasCreate()`** and **`cublasDestory()`** calls. In order for the cuBLAS library to use a different device in the same host thread, the application must set the new device to be used by calling **`cudaSetDevice()`** and then create another cuBLAS context, which will be associated with the new device, by calling **`cublasCreate()`**.

### 2.1.3. Thread Safety

The library is thread safe and its functions can be called from multiple host threads, even with the same **handle**. When multiple threads share the same handle, extreme care needs to be taken when the handle configuration is changed because that change will affect potentially subsequent CUBLAS calls in all threads. It is even more true for the destruction of the handle. So it is not recommended that multiple thread share the same CUBLAS handle.

### 2.1.4. Results reproducibility

By design, all CUBLAS API routines from a given toolkit version, generate the same bit-wise results at every run when executed on GPUs with the same architecture and the same number of SMs. However, bit-wise reproducibility is not guaranteed across toolkit version because the implementation might differ due to some implementation changes.

For some routines such as **cublas<t>symv** and **cublas<t>hemv**, an alternate significantly faster routines can be chosen using the routine **cublasSetAtomicsMode()**. In that case, the results are not guaranteed to be bit-wise reproducible because atomics are used for the computation.

### 2.1.5. Scalar Parameters

There are two categories of the functions that use scalar parameters :

- functions that take **alpha** and/or **beta** parameters by reference on the host or the device as scaling factors, such as **gemm**
- functions that return a scalar result on the host or the device such as **amax()**, **amin**, **asum()**, **rotg()**, **rotmg()**, **dot()** and **nrm2()**.

For the functions of the first category, when the pointer mode is set to **CUBLAS_POINTER_MODE_HOST**, the scalar parameters **alpha** and/or **beta** can be on the stack or allocated on the heap. Underneath the CUDA kernels related to that functions will be launched with the value of **alpha** and/or **beta**. Therefore if they were allocated on the heap, they can be freed just after the return of the call even though the kernel launch is asynchronous. When the pointer mode is set to **CUBLAS_POINTER_MODE_DEVICE**, **alpha** and/or **beta** must be accessible on the device and their values should not be modified until the kernel is done. Note that since **cudaFree()** does an implicit **cudaDeviceSynchronize()**, **cudaFree()** can still be called on **alpha** and/or **beta** just after the call but it would defeat the purpose of using this pointer mode in that case.

For the functions of the second category, when the pointer mode is set to **CUBLAS_POINTER_MODE_HOST**, these functions blocks the CPU, until the GPU has completed its computation and the results has been copied back to the Host. When the pointer mode is set to **CUBLAS_POINTER_MODE_DEVICE**, these functions return immediately. In this case, similarly to matrix and vector results, the scalar result is ready only when execution of the routine on the GPU has completed. This requires proper synchronization in order to read the result from the host.

In either case, the pointer mode **CUBLAS_POINTER_MODE_DEVICE** allows the library functions to execute completely asynchronously from the Host even when **alpha** and/or **beta** are generated by a previous kernel. For example, this situation can arise when iterative methods for solution of linear systems and eigenvalue problems are implemented using the cuBLAS library.

## 2.1.6. Parallelism with Streams

If the application uses the results computed by multiple independent tasks, CUDA™ streams can be used to overlap the computation performed in these tasks.

The application can conceptually associate each stream with each task. In order to achieve the overlap of computation between the tasks, the user should create CUDA™ streams using the function **cudaStreamCreate()** and set the stream to be used by each individual cuBLAS library routine by calling **cublasSetStream()** just before calling the actual cuBLAS routine. Then, the computation performed in separate streams would be overlapped automatically when possible on the GPU. This approach is especially useful when the computation performed by a single task is relatively small and is not enough to fill the GPU with work.

We recommend using the new cuBLAS API with scalar parameters and results passed by reference in the device memory to achieve maximum overlap of the computation when using streams.

A particular application of streams, batching of multiple small kernels, is described below.

## 2.1.7. Batching Kernels

In this section we will explain how to use streams to batch the execution of small kernels. For instance, suppose that we have an application where we need to make many small independent matrix-matrix multiplications with dense matrices.

It is clear that even with millions of small independent matrices we will not be able to achieve the same *GFLOPS* rate as with a one large matrix. For example, a single $n \times n$ large matrix-matrix multiplication performs $n^3$ operations for $n^2$ input size, while 1024 $\frac{n}{32} \times \frac{n}{32}$ small matrix-matrix multiplications perform $1024\left(\frac{n}{32}\right)^3 = \frac{n^3}{32}$ operations for the same input size. However, it is also clear that we can achieve a significantly better performance with many small independent matrices compared with a single small matrix.

The architecture family of GPUs allows us to execute multiple kernels simultaneously. Hence, in order to batch the execution of independent kernels, we can run each of them in a separate stream. In particular, in the above example we could create 1024 CUDA™ streams using the function **cudaStreamCreate()**, then preface each call to **cublas<t>gemm()** with a call to **cublasSetStream()** with a different stream for each of the matrix-matrix multiplications. This will ensure that when possible the different computations will be executed concurrently. Although the user can create many streams, in practice it is not possible to have more than 16 concurrent kernels executing at the same time.

## 2.1.8. Cache configuration

On some devices, L1 cache and shared memory use the same hardware resources. The cache configuration can be set directly with the CUDA Runtime function cudaDeviceSetCacheConfig. The cache configuration can also be set specifically for some functions using the routine cudaFuncSetCacheConfig. Please refer to the CUDA Runtime API documentation for details about the cache configuration settings.

Because switching from one configuration to another can affect kernels concurrency, the cuBLAS Library does not set any cache configuration preference and relies on the current setting. However, some cuBLAS routines, especially Level-3 routines, rely heavily on shared memory. Thus the cache preference setting might affect adversely their performance.

## 2.1.9. Device API Library

Starting with release 5.0, the CUDA Toolkit now provides a static cuBLAS Library cublas_device.a that contains device routines with the same API as the regular cuBLAS Library. Those routines use internally the Dynamic Parallelism feature to launch kernel from within and thus is only available for device with compute capability at least equal to 3.5.

In order to use those library routines from the device the user must include the header file "cublas_v2.h" corresponding to the new cuBLAS API and link against the static cuBLAS library cublas_device.a.

Those device cuBLAS library routines are called from the device in exactly the same way they are called from the host, with the following exceptions:

- The legacy cuBLAS API is not supported on the device.
- The pointer mode is not supported on the device, in other words, scalar input and output parameters must be allocated on the device memory.

Furthermore, the input and output scalar parameters must be allocated and released on the device using the cudaMalloc and cudaFree routines from the Host respectively or malloc and free routines from the device, in other words, they can not be passed by reference from the local memory to the routines.

## 2.1.10. Static Library support

Starting with release 6.5, the cuBLAS Library is also delivered in a static form as libcublas_static.a on Linux and Mac OSes. The static cuBLAS library and all others static maths libraries depend on a common thread abstraction layer library called libculibos.a.

For example, on Linux, to compile a small application using cuBLAS, against the dynamic library, the following command can be used:

```
nvcc myCublasApp.c  -lcublas  -o myCublasApp
```

Whereas to compile against the static cuBLAS library, the following command has to be used:

```
    nvcc myCublasApp.c  -lcublas_static   -lculibos -o myCublasApp
```

It is also possible to use the native Host C++ compiler. Depending on the Host Operating system, some additional libraries like **pthread** or **dl** might be needed on the linking line. The following command on Linux is suggested :

```
    g++ myCublasApp.c  -lcublas_static   -lculibos -lcudart_static -lpthread -
ldl -I <cuda-toolkit-path>/include -L <cuda-toolkit-path>/lib64 -o myCublasApp
```

Note that in the latter case, the library **cuda** is not needed. The CUDA Runtime will try to open explicitly the **cuda** library if needed. In the case of a system which does not have the CUDA driver installed, this allows the application to gracefully manage this issue and potentially run if a CPU-only path is available.

# 2.2. cuBLAS Datatypes Reference

## 2.2.1. cublasHandle_t

The **cublasHandle_t** type is a pointer type to an opaque structure holding the cuBLAS library context. The cuBLAS library context must be initialized using **cublasCreate()** and the returned handle must be passed to all subsequent library function calls. The context should be destroyed at the end using **cublasDestroy()**.

## 2.2.2. cublasStatus_t

The type is used for function status returns. All cuBLAS library functions return their status, which can have the following values.

| Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | The operation completed successfully. |
| **CUBLAS_STATUS_NOT_INITIALIZED** | The cuBLAS library was not initialized.<br>This is usually caused by the lack of a prior **cublasCreate()** call, an error in the CUDA Runtime API called by the cuBLAS routine, or an error in the hardware setup.<br>To correct: call **cublasCreate()** prior to the function call; and check that the hardware, an appropriate version of the driver, and the cuBLAS library are correctly installed. |
| **CUBLAS_STATUS_ALLOC_FAILED** | Resource allocation failed inside the cuBLAS library. This is usually caused by a **cudaMalloc()** failure.<br>To correct: prior to the function call, deallocate previously allocated memory as much as possible. |

| Value | Meaning |
|---|---|
| **CUBLAS_STATUS_INVALID_VALUE** | An unsupported value or parameter was passed to the function (a negative vector size, for example).<br>To correct: ensure that all the parameters being passed have valid values. |
| **CUBLAS_STATUS_ARCH_MISMATCH** | The function requires a feature absent from the device architecture; usually caused by the lack of support for double precision.<br>To correct: compile and run the application on a device with appropriate compute capability, which is 1.3 for double precision. |
| **CUBLAS_STATUS_MAPPING_ERROR** | An access to GPU memory space failed, which is usually caused by a failure to bind a texture.<br>To correct: prior to the function call, unbind any previously bound textures. |
| **CUBLAS_STATUS_EXECUTION_FAILED** | The GPU program failed to execute. This is often caused by a launch failure of the kernel on the GPU, which can be caused by multiple reasons.<br>To correct: check that the hardware, an appropriate version of the driver, and the cuBLAS library are correctly installed. |
| **CUBLAS_STATUS_INTERNAL_ERROR** | An internal cuBLAS operation failed. This error is usually caused by a **cudaMemcpyAsync()** failure.<br>To correct: check that the hardware, an appropriate version of the driver, and the cuBLAS library are correctly installed. Also, check that the memory passed as a parameter to the routine is not being deallocated prior to the routine's completion. |
| **CUBLAS_STATUS_NOT_SUPPORTED** | The functionnality requested is not supported |
| **CUBLAS_STATUS_LICENSE_ERROR** | The functionnality requested requires some license and an error was detected when trying to check the current licensing. This error can happen if the license is not present or is expired or if the environment variable NVIDIA_LICENSE_FILE is not set properly. |

## 2.2.3. cublasOperation_t

The **cublasOperation_t** type indicates which operation needs to be performed with the dense matrix. Its values correspond to Fortran characters **'N'** or **'n'** (non-transpose), **'T'** or **'t'** (transpose) and **'C'** or **'c'** (conjugate transpose) that are often used as parameters to legacy BLAS implementations.

| Value | Meaning |
|---|---|
| **CUBLAS_OP_N** | the non-transpose operation is selected |
| **CUBLAS_OP_T** | the transpose operation is selected |

| Value | Meaning |
|---|---|
| CUBLAS_OP_C | the conjugate transpose operation is selected |

## 2.2.4. cublasFillMode_t

The type indicates which part (lower or upper) of the dense matrix was filled and consequently should be used by the function. Its values correspond to Fortran characters 'L' or 'l' (lower) and 'U' or 'u' (upper) that are often used as parameters to legacy BLAS implementations.

| Value | Meaning |
|---|---|
| CUBLAS_FILL_MODE_LOWER | the lower part of the matrix is filled |
| CUBLAS_FILL_MODE_UPPER | the upper part of the matrix is filled |

## 2.2.5. cublasDiagType_t

The type indicates whether the main diagonal of the dense matrix is unity and consequently should not be touched or modified by the function. Its values correspond to Fortran characters 'N' or 'n' (non-unit) and 'U' or 'u' (unit) that are often used as parameters to legacy BLAS implementations.

| Value | Meaning |
|---|---|
| CUBLAS_DIAG_NON_UNIT | the matrix diagonal has non-unit elements |
| CUBLAS_DIAG_UNIT | the matrix diagonal has unit elements |

## 2.2.6. cublasSideMode_t

The type indicates whether the dense matrix is on the left or right side in the matrix equation solved by a particular function. Its values correspond to Fortran characters 'L' or 'l' (left) and 'R' or 'r' (right) that are often used as parameters to legacy BLAS implementations.

| Value | Meaning |
|---|---|
| CUBLAS_SIDE_LEFT | the matrix is on the left side in the equation |
| CUBLAS_SIDE_RIGHT | the matrix is on the right side in the equation |

## 2.2.7. cublasPointerMode_t

The **cublasPointerMode_t** type indicates whether the scalar values are passed by reference on the host or device. It is important to point out that if several scalar values are present in the function call, all of them must conform to the same single pointer mode. The pointer mode can be set and retrieved using **cublasSetPointerMode()** and **cublasGetPointerMode()** routines, respectively.

| Value | Meaning |
|---|---|
| CUBLAS_POINTER_MODE_HOST | the scalars are passed by reference on the host |
| CUBLAS_POINTER_MODE_DEVICE | the scalars are passed by reference on the device |

## 2.2.8. cublasAtomicsMode_t

The type indicates whether cuBLAS routines which has an alternate implementation using atomics can be used. The atomics mode can be set and queried using and routines, respectively.

| Value | Meaning |
|---|---|
| CUBLAS_ATOMICS_NOT_ALLOWED | the usage of atomics is not allowed |
| CUBLAS_ATOMICS_ALLOWED | the usage of atomics is allowed |

## 2.2.9. cublasGemmAlgo_t

cublasGemmAlgo_t type is an enumerant to specify the algorithm for matrix-matrix multiplication. It is used to run cublasGemmEx routine with specific algorithm. CUBLAS has the following algorithm options.

| Value | Meaning |
|---|---|
| CUBLAS_GEMM_DEFAULT | Apply Heuristics to select the GEMM algorithm |
| CUBLAS_GEMM_ALGO0 to CUBLAS_GEMM_ALGO17 | Explicitly choose an Algorithm [0,17] |
| CUBLAS_GEMM_DEFAULT_TENSOR_OP | Apply Heuristics to select the GEMM algorithm, and allow the use of Tensor Core operations when possible |
| CUBLAS_GEMM_ALGO0_TENSOR_OP to CUBLAS_GEMM_ALGO2_TENSOR_OP | Explicitly choose a GEMM Algorithm [0,2] while allowing the use of Tensor Core operations when possible |

## 2.2.10. cublasMath_t

**cublasMath_t** enumerate type is used in cublasSetMathMode to choose whether or not to use Tensor Core operations in the library by setting the math mode to either CUBLAS_TENSOR_OP_MATH or CUBLAS_DEFAULT_MATH.

| Value | Meaning |
|---|---|
| CUBLAS_DEFAULT_MATH | Prevent the library from using Tensor Core operations |
| CUBLAS_TENSOR_OP_MATH | Allows the library to use Tensor Core operations whenever possible |

# 2.3. CUDA Datatypes Reference

The chapter describes types shared by multiple CUDA Libraries and defined in the header file **library_types.h**.

## 2.3.1. cudaDataType_t

The **cudaDataType_t** type is an enumerant to specify the data precision. It is used when the data reference does not carry the type itself (e.g void *)

For example, it is used in the routine **cublasSgemmEx**.

| Value | Meaning |
|---|---|
| CUDA_R_16F | the data type is 16-bit floating-point |
| CUDA_C_16F | the data type is 16-bit complex floating-point |
| CUDA_R_32F | the data type is 32-bit floating-point |
| CUDA_C_32F | the data type is 32-bit complex floating-point |
| CUDA_R_64F | the data type is 64-bit floating-point |
| CUDA_C_64F | the data type is 64-bit complex floating-point |
| CUDA_R_8I | the data type is 8-bit signed integer |
| CUDA_C_8I | the data type is 8-bit complex signed integer |
| CUDA_R_8U | the data type is 8-bit unsigned integer |
| CUDA_C_8U | the data type is 8-bit complex unsigned integer |

## 2.3.2. libraryPropertyType_t

The **libraryPropertyType_t** is used as a parameter to specify which property is requested when using the routine **cublasGetProperty**

| Value | Meaning |
|---|---|
| MAJOR_VERSION | enumerant to query the major version |
| MINOR_VERSION | enumerant to query the minor version |
| PATCH_LEVEL | number to identify the patch level |

# 2.4. cuBLAS Helper Function Reference

## 2.4.1. cublasCreate()

```
cublasStatus_t
cublasCreate(cublasHandle_t *handle)
```

This function initializes the CUBLAS library and creates a handle to an opaque structure holding the CUBLAS library context. It allocates hardware resources on the host and device and must be called prior to making any other CUBLAS library calls. The CUBLAS library context is tied to the current CUDA device. To use the library on multiple devices, one CUBLAS handle needs to be created for each device. Furthermore, for a given device, multiple CUBLAS handles with different configuration can be created. Because **cublasCreate** allocates some internal resources and the release of those resources by calling **cublasDestroy** will implicitly call **cublasDeviceSynchronize**, it is recommended to minimize the number of **cublasCreate/cublasDestroy** occurences. For multi-threaded applications that use the same device from different threads, the recommended programming model is to create one CUBLAS handle per thread and use that CUBLAS handle for the entire life of the thread.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the initialization succeeded |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the CUDA™ Runtime initialization failed |
| **CUBLAS_STATUS_ALLOC_FAILED** | the resources could not be allocated |

## 2.4.2. cublasDestroy()

```
cublasStatus_t
cublasDestroy(cublasHandle_t handle)
```

This function releases hardware resources used by the CUBLAS library. This function is usually the last call with a particular handle to the CUBLAS library. Because **cublasCreate** allocates some internal resources and the release of those resources by calling **cublasDestroy** will implicitly call **cublasDeviceSynchronize**, it is recommended to minimize the number of **cublasCreate/cublasDestroy** occurences.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the shut down succeeded |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

## 2.4.3. cublasGetVersion()

```
cublasStatus_t
cublasGetVersion(cublasHandle_t handle, int *version)
```

This function returns the version number of the cuBLAS library.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

## 2.4.4. cublasGetProperty()

```
cublasStatus_t
cublasGetProperty(libraryPropertyType type, int *value)
```

This function returns the value of the requested property in memory pointed to by value. Refer to **libraryPropertyType** for supported types.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | The operation completed successfully |
| **CUBLAS_STATUS_INVALID_VALUE** | Invalid type value |

## 2.4.5. cublasSetStream()

```
cublasStatus_t
cublasSetStream(cublasHandle_t handle, cudaStream_t streamId)
```

This function sets the cuBLAS library stream, which will be used to execute all subsequent calls to the cuBLAS library functions. If the cuBLAS library stream is not set, all kernels use the *default* **NULL** stream. In particular, this routine can be used to change the stream between kernel launches and then to reset the cuBLAS library stream back to **NULL**.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the stream was set successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

## 2.4.6. cublasGetStream()

```
cublasStatus_t
cublasGetStream(cublasHandle_t handle, cudaStream_t *streamId)
```

This function gets the cuBLAS library stream, which is being used to execute all calls to the cuBLAS library functions. If the cuBLAS library stream is not set, all kernels use the *default* **NULL** stream.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the stream was returned successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

## 2.4.7. cublasGetPointerMode()

```
cublasStatus_t
cublasGetPointerMode(cublasHandle_t handle, cublasPointerMode_t *mode)
```

This function obtains the pointer mode used by the cuBLAS library. Please see the section on the **cublasPointerMode_t** type for more details.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the pointer mode was obtained successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

## 2.4.8. cublasSetPointerMode()

```
cublasStatus_t
cublasSetPointerMode(cublasHandle_t handle, cublasPointerMode_t mode)
```

This function sets the pointer mode used by the cuBLAS library. The *default* is for the values to be passed by reference on the host. Please see the section on the **cublasPointerMode_t** type for more details.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the pointer mode was set successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

## 2.4.9. cublasSetVector()

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

This function copies **n** elements from a vector **x** in host memory space to a vector **y** in GPU memory space. Elements in both vectors are assumed to have a size of **elemSize** bytes. The storage spacing between consecutive elements is given by **incx** for the source vector **x** and by **incy** for the destination vector **y**.

In general, **y** points to an object, or part of an object, that was allocated via **cublasAlloc()**. Since column-major format for two-dimensional matrices is assumed, if a vector is part of a matrix, a vector increment equal to **1** accesses a (partial) column of that matrix. Similarly, using an increment equal to the leading dimension of the matrix results in accesses to a (partial) row of that matrix.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **incx**, **incy**, **elemSize<=0** |
| **CUBLAS_STATUS_MAPPING_ERROR** | there was an error accessing GPU memory |

## 2.4.10. cublasGetVector()

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

This function copies **n** elements from a vector **x** in GPU memory space to a vector **y** in host memory space. Elements in both vectors are assumed to have a size of **elemSize** bytes. The storage spacing between consecutive elements is given by **incx** for the source vector and **incy** for the destination vector **y**.

In general, **x** points to an object, or part of an object, that was allocated via **cublasAlloc()**. Since column-major format for two-dimensional matrices is assumed, if a vector is part of a matrix, a vector increment equal to **1** accesses a (partial) column of

that matrix. Similarly, using an increment equal to the leading dimension of the matrix results in accesses to a (partial) row of that matrix.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **incx**, **incy**, **elemSize<=0** |
| **CUBLAS_STATUS_MAPPING_ERROR** | there was an error accessing GPU memory |

## 2.4.11. cublasSetMatrix()

```
cublasStatus_t
cublasSetMatrix(int rows, int cols, int elemSize,
                const void *A, int lda, void *B, int ldb)
```

This function copies a tile of **rows x cols** elements from a matrix **A** in host memory space to a matrix **B** in GPU memory space. It is assumed that each element requires storage of **elemSize** bytes and that both matrices are stored in column-major format, with the leading dimension of the source matrix **A** and destination matrix **B** given in **lda** and **ldb**, respectively. The leading dimension indicates the number of rows of the allocated matrix, even if only a submatrix of it is being used. In general, **B** is a device pointer that points to an object, or part of an object, that was allocated in GPU memory space via **cublasAlloc()**.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **rows, cols<0** or **elemSize, lda, ldb<=0** |
| **CUBLAS_STATUS_MAPPING_ERROR** | there was an error accessing GPU memory |

## 2.4.12. cublasGetMatrix()

```
cublasStatus_t
cublasGetMatrix(int rows, int cols, int elemSize,
                const void *A, int lda, void *B, int ldb)
```

This function copies a tile of **rows x cols** elements from a matrix **A** in GPU memory space to a matrix **B** in host memory space. It is assumed that each element requires storage of **elemSize** bytes and that both matrices are stored in column-major format, with the leading dimension of the source matrix **A** and destination matrix **B** given in **lda** and **ldb**, respectively. The leading dimension indicates the number of rows of the allocated matrix, even if only a submatrix of it is being used. In general, **A** is a device pointer that points to an object, or part of an object, that was allocated in GPU memory space via **cublasAlloc()**.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **rows, cols<0** or **elemSize, lda, ldb<=0** |
| **CUBLAS_STATUS_MAPPING_ERROR** | there was an error accessing GPU memory |

## 2.4.13. cublasSetVectorAsync()

```
cublasStatus_t
cublasSetVectorAsync(int n, int elemSize, const void *hostPtr, int incx,
                     void *devicePtr, int incy, cudaStream_t stream)
```

This function has the same functionality as **cublasSetVector()**, with the exception that the data transfer is done asynchronously (with respect to the host) using the given CUDA™ stream parameter.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **incx, incy, elemSize<=0** |
| **CUBLAS_STATUS_MAPPING_ERROR** | there was an error accessing GPU memory |

## 2.4.14. cublasGetVectorAsync()

```
cublasStatus_t
cublasGetVectorAsync(int n, int elemSize, const void *devicePtr, int incx,
                     void *hostPtr, int incy, cudaStream_t stream)
```

This function has the same functionality as **cublasGetVector()**, with the exception that the data transfer is done asynchronously (with respect to the host) using the given CUDA™ stream parameter.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **incx, incy, elemSize<=0** |
| **CUBLAS_STATUS_MAPPING_ERROR** | there was an error accessing GPU memory |

## 2.4.15. cublasSetMatrixAsync()

```
cublasStatus_t
cublasSetMatrixAsync(int rows, int cols, int elemSize, const void *A,
                     int lda, void *B, int ldb, cudaStream_t stream)
```

This function has the same functionality as **cublasSetMatrix()**, with the exception that the data transfer is done asynchronously (with respect to the host) using the given CUDA™ stream parameter.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **rows, cols<0** or **elemSize, lda, ldb<=0** |
| **CUBLAS_STATUS_MAPPING_ERROR** | there was an error accessing GPU memory |

## 2.4.16. cublasGetMatrixAsync()

```
cublasStatus_t
cublasGetMatrixAsync(int rows, int cols, int elemSize, const void *A,
                     int lda, void *B, int ldb, cudaStream_t stream)
```

This function has the same functionality as **cublasGetMatrix()**, with the exception that the data transfer is done asynchronously (with respect to the host) using the given CUDA™ stream parameter.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **rows, cols<0** or **elemSize, lda, ldb<=0** |
| **CUBLAS_STATUS_MAPPING_ERROR** | there was an error accessing GPU memory |

## 2.4.17. cublasSetAtomicsMode()

```
cublasStatus_t cublasSetAtomicsMode(cublasHandlet handle, cublasAtomicsMode_t
 mode)
```

Some routines like cublas<t>symv and cublas<t>hemv have an alternate implementation that use atomics to cumulate results. This implementation is generally significantly faster but can generate results that are not strictly identical from one run to the others. Mathematically, those different results are not significant but when debugging those differences can be prejudicial.

This function allows or disallows the usage of atomics in the cuBLAS library for all routines which have an alternate implementation. When not explicitly specified in the documentation of any cuBLAS routine, it means that this routine does not have an alternate implementation that use atomics. When atomics mode is disabled, each cuBLAS routine should produce the same results from one run to the other when called with identical parameters on the same Hardware.

The value of the atomics mode is CUBLASATOMICSNOTALLOWED. Please see the section on the type for more details.

| Return Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the atomics mode was set successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |

## 2.4.18. cublasGetAtomicsMode()

```
cublasStatus_t cublasGetAtomicsMode(cublasHandle_t handle, cublasAtomicsMode_t
 *mode)
```

This function queries the atomic mode of a specific cuBLAS context.

The value of the atomics mode is CUBLASATOMICSNOTALLOWED. Please see the section on the type for more details.

| Return Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the atomics mode was queried successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |

## 2.4.19. cublasSetMathMode()

```
cublasStatus_t cublasSetMathMode(cublasHandle_t handle, cublasMath_t mode)
```

The cublasSetMathMode function enables you to choose whether or not to use Tensor Core operations in the library by setting the math mode to either CUBLAS_TENSOR_OP_MATH or CUBLAS_DEFAULT_MATH. Tensor Core operations perform parallel floating point accumulation of multiple floating point products. Setting the math mode to CUBLAS_TENSOR_OP_MATH indicates that the library will use Tensor Core operations in the functions: cublasHgemm(), cublasGemmEx, cublasSgemmEx(), cublasHgemmBatched() and cublasHgemmStridedBatched(). The math mode default is CUBLAS_DEFAULT_MATH, this default indicates that the Tensor Core operations will be avoided by the library. The default mode is a serialized operation, the Tensor Core operations are parallelized, thus the two might result in slight different numerical results due to the different sequencing of operations. Note: The library falls back to the default math mode when Tensor Core operations are not supported or not permitted.

| Atype/ Btype | Ctype | computeTyp | alpha / beta | Supported Functions when CUBLAS_TENSOR_OP_MATH is set |
|---|---|---|---|---|
| CUDA_R_16F | CUDA_R_32F | CUDA_R_32F | CUDA_R_32F | cublasGemmEx(), cublasSgemmEx(), cublasGemmBatchedEx(), cublasGemmStridedBatchedEx() |
| CUDA_R_16F | CUDA_R_16F | CUDA_R_32F | CUDA_R_32F | cublasGemmEx(), cublasSgemmEx(), cublasGemmBatchedEx(), cublasGemmStridedBatchedEx() |
| CUDA_R_16F | CUDA_R_16F | CUDA_R_16F | CUDA_R_16F | cublasHgemm(), cublasHgemmBatched() , cublasHgemmStridedBatched() |
| CUDA_R_32F | CUDA_R_32F | CUDA_R_32F | CUDA_R_32F | cublasSgemm(), cublasGemmEx(), cublasSgemmEx(), |

| Atype/ Btype | Ctype | computeTyp | alpha / beta | Supported Functions when CUBLAS_TENSOR_OP_MATH is set |
|---|---|---|---|---|
| | | | | cublasGemmBatchedEx(), cublasGemmStridedBatchedEx() NOTE: A conversion from CUDA_R_32F to CUDA_R_16F with round to nearest on the input values A/B is performed when Tensor Core operations are used |

| Return Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the math mode was set successfully. |
| CUBLAS_STATUS_INVALID_VALUE | an invalid value for mode was specified. |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized. |

## 2.4.20. cublasGetMathMode()

```
cublasStatus_t cublasGetMathMode(cublasHandle_t handle, cublasMath_t *mode)
```

This function returns the math mode used by the library routines.

| Return Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the math type was returned successfully. |
| CUBLAS_STATUS_INVALID_VALUE | if mode is NULL. |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized. |

# 2.5. cuBLAS Level-1 Function Reference

In this chapter we describe the Level-1 Basic Linear Algebra Subprograms (BLAS1) functions that perform scalar and vector based operations. We will use abbreviations <*type*> for type and <*t*> for the corresponding short type to make a more concise and clear presentation of the implemented functions. Unless otherwise specified <*type*> and <*t*> have the following meanings:

| <type> | <t> | Meaning |
|---|---|---|
| float | ‘s’ or ‘S’ | real single-precision |
| double | ‘d’ or ‘D’ | real double-precision |
| cuComplex | ‘c’ or ‘C’ | complex single-precision |
| cuDoubleComplex | ‘z’ or ‘Z’ | complex double-precision |

When the parameters and returned values of the function differ, which sometimes happens for complex input, the <t> can also have the following meanings ‘Sc’, ‘Cs’, ‘Dz’ and ‘Zd’.

The abbreviation **Re**(.) and **Im**(.) will stand for the real and imaginary part of a number, respectively. Since imaginary part of a real number does not exist, we will consider it to be zero and can usually simply discard it from the equation where it is being used. Also, the $\bar{\alpha}$ will denote the complex conjugate of $\alpha$ .

In general throughout the documentation, the lower case Greek symbols $\alpha$ and $\beta$ will denote scalars, lower case English letters in bold type $\mathbf{x}$ and $\mathbf{y}$ will denote vectors and capital English letters $A$ , $B$ and $C$ will denote matrices.

## 2.5.1. cublasI<t>amax()

```
cublasStatus_t cublasIsamax(cublasHandle_t handle, int n,
                            const float *x, int incx, int *result)
cublasStatus_t cublasIdamax(cublasHandle_t handle, int n,
                            const double *x, int incx, int *result)
cublasStatus_t cublasIcamax(cublasHandle_t handle, int n,
                            const cuComplex *x, int incx, int *result)
cublasStatus_t cublasIzamax(cublasHandle_t handle, int n,
                            const cuDoubleComplex *x, int incx, int *result)
```

This function finds the (smallest) index of the element of the maximum magnitude. Hence, the result is the first $i$ such that $|\mathbf{Im}(x[j])|+|\mathbf{Re}(x[j])|$ is maximum for $i = 1, \ldots, n$ and $j = 1+(i-1)*$ incx . Notice that the last equation reflects 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vector **x**. |
| x | device | input | <type> vector with elements. |
| incx | | input | stride between consecutive elements of **x**. |
| result | host or device | output | the resulting index, which is 0 if **n,incx<=0**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ALLOC_FAILED** | the reduction buffer could not be allocated |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

isamax, idamax, icamax, izamax

## 2.5.2. cublasI<t>amin()

```
cublasStatus_t cublasIsamin(cublasHandle_t handle, int n,
                            const float *x, int incx, int *result)
cublasStatus_t cublasIdamin(cublasHandle_t handle, int n,
                            const double *x, int incx, int *result)
cublasStatus_t cublasIcamin(cublasHandle_t handle, int n,
                            const cuComplex *x, int incx, int *result)
cublasStatus_t cublasIzamin(cublasHandle_t handle, int n,
                            const cuDoubleComplex *x, int incx, int *result)
```

This function finds the (smallest) index of the element of the minimum magnitude. Hence, the result is the first $i$ such that $|\mathrm{Im}(x[j])|+|\mathrm{Re}(x[j])|$ is minimum for $i = 1, \ldots, n$ and $j = 1+(i-1)*$ incx Notice that the last equation reflects 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vector **x**. |
| x | device | input | <type> vector with elements. |
| incx | | input | stride between consecutive elements of **x**. |
| result | host or device | output | the resulting index, which is 0 if **n,incx<=0**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ALLOC_FAILED** | the reduction buffer could not be allocated |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

isamin

## 2.5.3. cublas<t>asum()

```
cublasStatus_t  cublasSasum(cublasHandle_t handle, int n,
                            const float           *x, int incx, float  *result)
cublasStatus_t  cublasDasum(cublasHandle_t handle, int n,
                            const double          *x, int incx, double *result)
cublasStatus_t cublasScasum(cublasHandle_t handle, int n,
                            const cuComplex       *x, int incx, float  *result)
cublasStatus_t cublasDzasum(cublasHandle_t handle, int n,
                            const cuDoubleComplex *x, int incx, double *result)
```

This function computes the sum of the absolute values of the elements of vector **x**. Hence, the result is $\sum_{i=1}^{n} |\mathbf{Im}(x[j])| + |\mathbf{Re}(x[j])|$ where $j = 1 + (i-1)*$ incx . Notice that the last equation reflects 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vector **x**. |
| x | device | input | <type> vector with elements. |
| incx | | input | stride between consecutive elements of **x**. |
| result | host or device | output | the resulting index, which is `0.0` if `n,incx<=0`. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ALLOC_FAILED** | the reduction buffer could not be allocated |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sasum, dasum, scasum, dzasum

## 2.5.4. cublas<t>axpy()

```
cublasStatus_t cublasSaxpy(cublasHandle_t handle, int n,
                           const float           *alpha,
                           const float           *x, int incx,
                           float                 *y, int incy)
cublasStatus_t cublasDaxpy(cublasHandle_t handle, int n,
                           const double          *alpha,
                           const double          *x, int incx,
                           double                *y, int incy)
cublasStatus_t cublasCaxpy(cublasHandle_t handle, int n,
                           const cuComplex       *alpha,
                           const cuComplex       *x, int incx,
                           cuComplex             *y, int incy)
cublasStatus_t cublasZaxpy(cublasHandle_t handle, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *x, int incx,
                           cuDoubleComplex       *y, int incy)
```

This function multiplies the vector **x** by the scalar $\alpha$ and adds it to the vector **y** overwriting the latest vector with the result. Hence, the performed operation is $\mathbf{y}[j] = \alpha \times \mathbf{x}[k] + \mathbf{y}[j]$ for $i = 1, \ldots, n$, $k = 1 + (i-1)*$ incx and $j = 1 + (i-1)*$ incy . Notice that the last two equations reflect 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| n | | input | number of elements in the vector **x** and **y**. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | in/out | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

saxpy, daxpy, caxpy, zaxpy

## 2.5.5. cublas<t>copy()

```
cublasStatus_t cublasScopy(cublasHandle_t handle, int n,
                           const float           *x, int incx,
                           float                 *y, int incy)
cublasStatus_t cublasDcopy(cublasHandle_t handle, int n,
                           const double          *x, int incx,
                           double                *y, int incy)
cublasStatus_t cublasCcopy(cublasHandle_t handle, int n,
                           const cuComplex       *x, int incx,
                           cuComplex             *y, int incy)
cublasStatus_t cublasZcopy(cublasHandle_t handle, int n,
                           const cuDoubleComplex *x, int incx,
                           cuDoubleComplex       *y, int incy)
```

This function copies the vector **x** into the vector **y**. Hence, the performed operation is $\mathbf{y}[j] = \mathbf{x}[k]$ for $i = 1, \ldots, n$, $k = 1 + (i-1)*$ incx and $j = 1 + (i-1)*$ incy. Notice that the last two equations reflect 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vector **x** and **y**. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| y | device | output | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

scopy, dcopy, ccopy, zcopy

## 2.5.6. cublas<t>dot()

```
cublasStatus_t cublasSdot (cublasHandle_t handle, int n,
                           const float           *x, int incx,
                           const float           *y, int incy,
                           float           *result)
cublasStatus_t cublasDdot (cublasHandle_t handle, int n,
                           const double          *x, int incx,
                           const double          *y, int incy,
                           double          *result)
cublasStatus_t cublasCdotu(cublasHandle_t handle, int n,
                           const cuComplex       *x, int incx,
                           const cuComplex       *y, int incy,
                           cuComplex       *result)
cublasStatus_t cublasCdotc(cublasHandle_t handle, int n,
                           const cuComplex       *x, int incx,
                           const cuComplex       *y, int incy,
                           cuComplex       *result)
cublasStatus_t cublasZdotu(cublasHandle_t handle, int n,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *y, int incy,
                           cuDoubleComplex *result)
cublasStatus_t cublasZdotc(cublasHandle_t handle, int n,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *y, int incy,
                           cuDoubleComplex       *result)
```

This function computes the dot product of vectors **x** and **y**. Hence, the result is $\sum_{i=1}^{n}(\mathbf{x}[k]\times\mathbf{y}[j])$ where $k=1+(i-1)*$ incx and $j=1+(i-1)*$ incy . Notice that in the first equation the conjugate of the element of vector should be used if the function name ends in character 'c' and that the last two equations reflect 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vectors **x** and **y**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | input | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |
| result | host or device | output | the resulting dot product, which is **0.0** if **n<=0**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ALLOC_FAILED** | the reduction buffer could not be allocated |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sdot, ddot, cdotu, cdotc, zdotu, zdotc

## 2.5.7. cublas<t>nrm2()

```
cublasStatus_t  cublasSnrm2(cublasHandle_t handle, int n,
                            const float           *x, int incx, float  *result)
cublasStatus_t  cublasDnrm2(cublasHandle_t handle, int n,
                            const double          *x, int incx, double *result)
cublasStatus_t cublasScnrm2(cublasHandle_t handle, int n,
                            const cuComplex       *x, int incx, float  *result)
cublasStatus_t cublasDznrm2(cublasHandle_t handle, int n,
                            const cuDoubleComplex *x, int incx, double *result)
```

This function computes the Euclidean norm of the vector **x**. The code uses a multiphase model of accumulation to avoid intermediate underflow and overflow, with the result being equivalent to $\sqrt{\sum_{i=1}^{n}(\mathbf{x}[j]\times\mathbf{x}[j])}$ where $j = 1+(i-1)*$ incx in exact arithmetic. Notice that the last equation reflects 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vector **x**. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| result | host or device | output | the resulting norm, which is **0.0** if **n,incx<=0**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ALLOC_FAILED** | the reduction buffer could not be allocated |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

snrm2, snrm2, dnrm2, dnrm2, scnrm2, scnrm2, dznrm2

## 2.5.8. cublas<t>rot()

```
cublasStatus_t  cublasSrot(cublasHandle_t handle, int n,
                           float           *x, int incx,
                           float           *y, int incy,
                           const float  *c, const float           *s)
cublasStatus_t  cublasDrot(cublasHandle_t handle, int n,
                           double          *x, int incx,
                           double          *y, int incy,
                           const double *c, const double          *s)
cublasStatus_t  cublasCrot(cublasHandle_t handle, int n,
                           cuComplex       *x, int incx,
                           cuComplex       *y, int incy,
                           const float  *c, const cuComplex       *s)
cublasStatus_t cublasCsrot(cublasHandle_t handle, int n,
                           cuComplex       *x, int incx,
                           cuComplex       *y, int incy,
                           const float  *c, const float           *s)
cublasStatus_t  cublasZrot(cublasHandle_t handle, int n,
                           cuDoubleComplex *x, int incx,
                           cuDoubleComplex *y, int incy,
                           const double *c, const cuDoubleComplex *s)
cublasStatus_t cublasZdrot(cublasHandle_t handle, int n,
                           cuDoubleComplex *x, int incx,
                           cuDoubleComplex *y, int incy,
                           const double *c, const double          *s)
```

This function applies Givens rotation matrix

$$G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

to vectors **x** and **y**.

Hence, the result is $\mathbf{x}[k] = c \times \mathbf{x}[k] + s \times \mathbf{y}[j]$ and $\mathbf{y}[j] = -s \times \mathbf{x}[k] + c \times \mathbf{y}[j]$ where $k = 1 + (i-1)*$ incx and $j = 1 + (i-1)*$ incy . Notice that the last two equations reflect 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vectors **x** and **y**. |
| x | device | in/out | <type> vector with **n** elements. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| incx | | input | stride between consecutive elements of **x**. |
| y | device | in/out | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |
| c | host or device | input | cosine element of the rotation matrix. |
| s | host or device | input | sine element of the rotation matrix. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

srot, drot, crot, csrot, zrot, zdrot

## 2.5.9. cublas<t>rotg()

```
cublasStatus_t cublasSrotg(cublasHandle_t handle,
                           float          *a, float          *b,
                           float  *c, float          *s)
cublasStatus_t cublasDrotg(cublasHandle_t handle,
                           double          *a, double          *b,
                           double *c, double          *s)
cublasStatus_t cublasCrotg(cublasHandle_t handle,
                           cuComplex       *a, cuComplex       *b,
                           float  *c, cuComplex       *s)
cublasStatus_t cublasZrotg(cublasHandle_t handle,
                           cuDoubleComplex *a, cuDoubleComplex *b,
                           double *c, cuDoubleComplex *s)
```

This function constructs the Givens rotation matrix

$$G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

that zeros out the second entry of a $2 \times 1$ vector $(a, b)^T$.

Then, for real numbers we can write

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$$

where $c^2 + s^2 = 1$ and $r = a^2 + b^2$. The parameters $a$ and $b$ are overwritten with $r$ and $z$, respectively. The value of $z$ is such that $c$ and $s$ may be recovered using the following rules:

$$(c, s) = \begin{cases} \left(\sqrt{1-z^2}, z\right) & \text{if } |z| < 1 \\ (0.0, 1.0) & \text{if } |z| = 1 \\ \left(1/z, \sqrt{1-z^2}\right) & \text{if } |z| > 1 \end{cases}$$

For complex numbers we can write

$$\begin{pmatrix} c & s \\ -\bar{s} & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$$

where $c^2 + (\bar{s} \times s) = 1$ and $r = \frac{a}{|a|} \times \|(a, b)^T\|_2$ with $\|(a, b)^T\|_2 = \sqrt{|a|^2 + |b|^2}$ for $a \neq 0$ and $r = b$ for $a = 0$. Finally, the parameter $a$ is overwritten with $r$ on exit.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| a | host or device | in/out | <type> scalar that is overwritten with $r$. |
| b | host or device | in/out | <type> scalar that is overwritten with $z$. |
| c | host or device | output | cosine element of the rotation matrix. |
| s | host or device | output | sine element of the rotation matrix. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

srotg, drotg, crotg, zrotg

## 2.5.10. cublas<t>rotm()

```
cublasStatus_t cublasSrotm(cublasHandle_t handle, int n, float  *x, int incx,
                           float  *y, int incy, const float*  param)
cublasStatus_t cublasDrotm(cublasHandle_t handle, int n, double *x, int incx,
                           double *y, int incy, const double* param)
```

This function applies the modified Givens transformation

$$H = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}$$

to vectors **x** and **y**.

Hence, the result is $\mathbf{x}[k] = h_{11} \times \mathbf{x}[k] + h_{12} \times \mathbf{y}[j]$ and $\mathbf{y}[j] = h_{21} \times \mathbf{x}[k] + h_{22} \times \mathbf{y}[j]$ where $k = 1 + (i-1)*$ incx and $j = 1 + (i-1)*$ incy . Notice that the last two equations reflect 1-based indexing used for compatibility with Fortran.

The elements , , and of matrix $H$ are stored in **param[1]**, **param[2]**, **param[3]** and **param[4]**, respectively. The **flag=param[0]** defines the following predefined values for the matrix $H$ entries

| flag=-1.0 | flag= 0.0 | flag= 1.0 | flag=-2.0 |
|---|---|---|---|
| $\begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}$ | $\begin{pmatrix} 1.0 & h_{12} \\ h_{21} & 1.0 \end{pmatrix}$ | $\begin{pmatrix} h_{11} & 1.0 \\ -1.0 & h_{22} \end{pmatrix}$ | $\begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$ |

Notice that the values -1.0, 0.0 and 1.0 implied by the flag are not stored in param.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vectors **x** and **y**. |
| x | device | in/out | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | in/out | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |
| param | host or device | input | <type> vector of 5 elements, where **param[0]** and **param[1-4]** contain the flag and matrix $H$ . |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

srotm, drotm

## 2.5.11. cublas<t>rotmg()

```
cublasStatus_t cublasSrotmg(cublasHandle_t handle, float  *d1, float  *d2,
                            float  *x1, const float  *y1, float  *param)
cublasStatus_t cublasDrotmg(cublasHandle_t handle, double *d1, double *d2,
                            double *x1, const double *y1, double *param)
```

This function constructs the modified Givens transformation

$$H = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}$$

that zeros out the second entry of a $2 \times 1$ vector $\left(\sqrt{d1} * x1, \sqrt{d2} * y1\right)^T$.

The **flag=param[0]** defines the following predefined values for the matrix $H$ entries

| flag=-1.0 | flag= 0.0 | flag= 1.0 | flag=-2.0 |
|---|---|---|---|
| $\begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}$ | $\begin{pmatrix} 1.0 & h_{12} \\ h_{21} & 1.0 \end{pmatrix}$ | $\begin{pmatrix} h_{11} & 1.0 \\ -1.0 & h_{22} \end{pmatrix}$ | $\begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$ |

Notice that the values -1.0, 0.0 and 1.0 implied by the flag are not stored in param.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| d1 | host or device | in/out | <type> scalar that is overwritten on exit. |
| d2 | host or device | in/out | <type> scalar that is overwritten on exit. |
| x1 | host or device | in/out | <type> scalar that is overwritten on exit. |
| y1 | host or device | input | <type> scalar. |
| param | host or device | output | <type> vector of 5 elements, where **param[0]** and **param[1-4]** contain the flag and matrix $H$. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

srotmg, drotmg

## 2.5.12. cublas<t>scal()

```
cublasStatus_t  cublasSscal(cublasHandle_t handle, int n,
                            const float           *alpha,
                            float           *x, int incx)
cublasStatus_t  cublasDscal(cublasHandle_t handle, int n,
                            const double          *alpha,
                            double          *x, int incx)
cublasStatus_t  cublasCscal(cublasHandle_t handle, int n,
                            const cuComplex       *alpha,
                            cuComplex       *x, int incx)
cublasStatus_t cublasCsscal(cublasHandle_t handle, int n,
                            const float           *alpha,
                            cuComplex       *x, int incx)
cublasStatus_t  cublasZscal(cublasHandle_t handle, int n,
                            const cuDoubleComplex *alpha,
                            cuDoubleComplex *x, int incx)
cublasStatus_t cublasZdscal(cublasHandle_t handle, int n,
                            const double          *alpha,
                            cuDoubleComplex *x, int incx)
```

This function scales the vector **x** by the scalar $\alpha$ and overwrites it with the result. Hence, the performed operation is $\mathbf{x}[j] = \alpha \times \mathbf{x}[j]$ for $i = 1, \ldots, n$ and $j = 1 + (i - 1) * \text{incx}$. Notice that the last two equations reflect 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| n | | input | number of elements in the vector **x**. |
| x | device | in/out | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sscal, dscal, csscal, cscal, zdscal, zscal

## 2.5.13. cublas<t>swap()

```
cublasStatus_t cublasSswap(cublasHandle_t handle, int n, float           *x,
                           int incx, float           *y, int incy)
cublasStatus_t cublasDswap(cublasHandle_t handle, int n, double          *x,
                           int incx, double          *y, int incy)
cublasStatus_t cublasCswap(cublasHandle_t handle, int n, cuComplex       *x,
                           int incx, cuComplex       *y, int incy)
cublasStatus_t cublasZswap(cublasHandle_t handle, int n, cuDoubleComplex *x,
                           int incx, cuDoubleComplex *y, int incy)
```

This function interchanges the elements of vector **x** and **y**. Hence, the performed operation is $\mathbf{y}[j] \Leftrightarrow \mathbf{x}[k]$ for $i = 1, \ldots, n$, $k = 1+(i-1)*$ incx and $j = 1+(i-1)*$ incy. Notice that the last two equations reflect 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vector **x** and **y**. |
| x | device | in/out | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | in/out | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sswap, dswap, cswap, zswap

# 2.6. cuBLAS Level-2 Function Reference

In this chapter we describe the Level-2 Basic Linear Algebra Subprograms (BLAS2) functions that perform matrix-vector operations.

## 2.6.1. cublas<t>gbmv()

```
cublasStatus_t cublasSgbmv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n, int kl, int ku,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *x, int incx,
                           const float           *beta,
                           float           *y, int incy)
cublasStatus_t cublasDgbmv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n, int kl, int ku,
                           const double          *alpha,
                           const double          *A, int lda,
                           const double          *x, int incx,
                           const double          *beta,
                           double          *y, int incy)
cublasStatus_t cublasCgbmv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n, int kl, int ku,
                           const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *x, int incx,
                           const cuComplex       *beta,
                           cuComplex       *y, int incy)
cublasStatus_t cublasZgbmv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n, int kl, int ku,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *y, int incy)
```

This function performs the banded matrix-vector multiplication

$$\mathbf{y} = \alpha\ \mathrm{op}(A)\mathbf{x} + \beta\mathbf{y}$$

where $A$ is a banded matrix with $kl$ subdiagonals and $ku$ superdiagonals, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ and $\beta$ are scalars. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa} == \text{CUBLAS\_OP\_N} \\ A^T & \text{if transa} == \text{CUBLAS\_OP\_T} \\ A^H & \text{if transa} == \text{CUBLAS\_OP\_H} \end{cases}$$

The banded matrix $A$ is stored column by column, with the main diagonal stored in row $ku+1$ (starting in first position), the first superdiagonal stored in row $ku$ (starting in second position), the first subdiagonal stored in row $ku+2$ (starting in first position), etc. So that in general, the element $A(i, j)$ is stored in the memory location **`A(ku+1+i-j,j)`** for $j = 1, \ldots, n$ and $i \in [\max(1, j-ku), \min(m, j+kl)]$. Also, the elements in the array $A$ that do not conceptually correspond to the elements in the banded matrix (the top left $ku \times ku$ and bottom right $kl \times kl$ triangles) are not referenced.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix **A**. |
| n | | input | number of columns of matrix **A**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| kl | | input | number of subdiagonals of matrix **A**. |
| ku | | input | number of superdiagonals of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x n** with **lda>=kl+ku+1**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| x | device | input | <type> vector with **n** elements if **transa == CUBLAS_OP_N** and **m** elements otherwise. |
| incx | | input | stride between consecutive elements of **x**. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta == 0** then **y** does not have to be a valid input. |
| y | device | in/out | <type> vector with **m** elements if **transa == CUBLAS_OP_N** and **n** elements otherwise. |
| incy | | input | stride between consecutive elements of **y**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters or |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sgbmv, dgbmv, cgbmv, zgbmv

## 2.6.2. cublas<t>gemv()

```
cublasStatus_t cublasSgemv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *x, int incx,
                           const float           *beta,
                           float           *y, int incy)
cublasStatus_t cublasDgemv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n,
                           const double          *alpha,
                           const double          *A, int lda,
                           const double          *x, int incx,
                           const double          *beta,
                           double          *y, int incy)
cublasStatus_t cublasCgemv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n,
                           const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *x, int incx,
                           const cuComplex       *beta,
                           cuComplex       *y, int incy)
cublasStatus_t cublasZgemv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *y, int incy)
```

This function performs the matrix-vector multiplication

$$\mathbf{y} = \alpha \mathrm{op}(A)\mathbf{x} + \beta\mathbf{y}$$

where $A$ is a $m \times n$ matrix stored in column-major format, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ and $\beta$ are scalars. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa} == \text{CUBLAS\_OP\_N} \\ A^T & \text{if transa} == \text{CUBLAS\_OP\_T} \\ A^H & \text{if transa} == \text{CUBLAS\_OP\_H} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix **A**. |
| n | | input | number of columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x n** with **lda >= max(1,m)**. Before entry, the leading **m** by **n** part of the array **A** must contain the matrix of coefficients. Unchanged on exit. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. **lda** must be at least **max(1,m)**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| x | device | input | <type> vector at least `(1+(n-1)*abs(incx))` elements if `transa==CUBLAS_OP_N` and at least `(1+(m-1)*abs(incx))` elements otherwise. |
| incx | | input | stride between consecutive elements of `x`. |
| beta | host or device | input | <type> scalar used for multiplication, if `beta==0` then `y` does not have to be a valid input. |
| y | device | in/out | <type> vector at least `(1+(m-1)*abs(incy))` elements if `transa==CUBLAS_OP_N` and at least `(1+(n-1)*abs(incy))` elements otherwise. |
| incy | | input | stride between consecutive elements of `y` |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `m,n<0` or `incx,incy=0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

sgemv, dgemv, cgemv, zgemv

## 2.6.3. cublas<t>ger()

```
cublasStatus_t  cublasSger(cublasHandle_t handle, int m, int n,
                           const float           *alpha,
                           const float           *x, int incx,
                           const float           *y, int incy,
                           float           *A, int lda)
cublasStatus_t  cublasDger(cublasHandle_t handle, int m, int n,
                           const double          *alpha,
                           const double          *x, int incx,
                           const double          *y, int incy,
                           double          *A, int lda)
cublasStatus_t cublasCgeru(cublasHandle_t handle, int m, int n,
                           const cuComplex       *alpha,
                           const cuComplex       *x, int incx,
                           const cuComplex       *y, int incy,
                           cuComplex       *A, int lda)
cublasStatus_t cublasCgerc(cublasHandle_t handle, int m, int n,
                           const cuComplex       *alpha,
                           const cuComplex       *x, int incx,
                           const cuComplex       *y, int incy,
                           cuComplex       *A, int lda)
cublasStatus_t cublasZgeru(cublasHandle_t handle, int m, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *y, int incy,
                           cuDoubleComplex *A, int lda)
cublasStatus_t cublasZgerc(cublasHandle_t handle, int m, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *y, int incy,
                           cuDoubleComplex *A, int lda)
```

This function performs the rank-1 update

$$A = \begin{cases} \alpha \mathbf{x}\mathbf{y}^T + A & \text{if ger(),geru() is called} \\ \alpha \mathbf{x}\mathbf{y}^H + A & \text{if gerc() is called} \end{cases}$$

where $A$ is a $m \times n$ matrix stored in column-major format, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ is a scalar.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| m | | input | number of rows of matrix **A**. |
| n | | input | number of columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| x | device | input | <type> vector with **m** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | input | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |
| A | device | in/out | <type> array of dimension **lda x n** with **lda >= max(1,m)**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sger, dger, cgeru, cgerc, zgeru, zgerc

## 2.6.4. cublas<t>sbmv()

```
cublasStatus_t cublasSsbmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, int k, const float  *alpha,
                           const float  *A, int lda,
                           const float  *x, int incx,
                           const float  *beta, float *y, int incy)
cublasStatus_t cublasDsbmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, int k, const double *alpha,
                           const double *A, int lda,
                           const double *x, int incx,
                           const double *beta, double *y, int incy)
```

This function performs the symmetric banded matrix-vector multiplication

$$\mathbf{y} = \alpha A\mathbf{x} + \beta\mathbf{y}$$

where $A$ is a $n \times n$ symmetric banded matrix with $k$ subdiagonals and superdiagonals, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ and $\beta$ are scalars.

If **uplo == CUBLAS_FILL_MODE_LOWER** then the symmetric banded matrix $A$ is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. So that in general, the element $A(i, j)$ is stored in the memory location **A(1+i-j,j)** for $j = 1, \ldots, n$ and $i \in [j, \min(m, j+k)]$. Also, the elements in the array **A** that do not conceptually correspond to the elements in the banded matrix (the bottom right $k \times k$ triangle) are not referenced.

If **uplo == CUBLAS_FILL_MODE_UPPER** then the symmetric banded matrix $A$ is stored column by column, with the main diagonal of the matrix stored in row **k+1**, the first superdiagonal in row **k** (starting at second position), the second superdiagonal in row **k-1** (starting at third position), etc. So that in general, the element $A(i, j)$ is stored in the memory location **A(1+k+i-j,j)** for $j = 1, \ldots, n$ and $i \in [\max(1, j-k), j]$. Also,

the elements in the array **A** that do not conceptually correspond to the elements in the banded matrix (the top left $k \times k$ triangle) are not referenced.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| k | | input | number of sub- and super-diagonals of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x n** with **\lda >= k+1**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta==0** then **y** does not have to be a valid input. |
| y | device | in/out | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

ssbmv, dsbmv

## 2.6.5. cublas<t>spmv()

```
cublasStatus_t cublasSspmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const float  *alpha, const float  *AP,
                           const float  *x, int incx, const float  *beta,
                           float  *y, int incy)
cublasStatus_t cublasDspmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const double *alpha, const double *AP,
                           const double *x, int incx, const double *beta,
                           double *y, int incy)
```

This function performs the symmetric packed matrix-vector multiplication

$\mathbf{y} = \alpha A\mathbf{x} + \beta\mathbf{y}$

where $A$ is a $n \times n$ symmetric matrix stored in packed format, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ and $\beta$ are scalars.

If **`uplo == CUBLAS_FILL_MODE_LOWER`** then the elements in the lower triangular part of the symmetric matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **`AP[i+((2*n-j+1)*j)/2]`** for $j = 1, \ldots, n$ and $i \geq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

If **`uplo == CUBLAS_FILL_MODE_UPPER`** then the elements in the upper triangular part of the symmetric matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **`AP[i+(j*(j+1))/2]`** for $j = 1, \ldots, n$ and $i \leq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix $A$ lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix $A$. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| AP | device | input | <type> array with $A$ stored in packed format. |
| x | device | input | <type> vector with **`n`** elements. |
| incx | | input | stride between consecutive elements of **`x`**. |
| beta | host or device | input | <type> scalar used for multiplication, if **`beta==0`** then **`y`** does not have to be a valid input. |
| y | device | input | <type> vector with **`n`** elements. |
| incy | | input | stride between consecutive elements of **`y`**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **`CUBLAS_STATUS_SUCCESS`** | the operation completed successfully |
| **`CUBLAS_STATUS_NOT_INITIALIZED`** | the library was not initialized |
| **`CUBLAS_STATUS_INVALID_VALUE`** | the parameters **`n<0`** or **`incx,incy=0`** |
| **`CUBLAS_STATUS_ARCH_MISMATCH`** | the device does not support double-precision |
| **`CUBLAS_STATUS_EXECUTION_FAILED`** | the function failed to launch on the GPU |

For references please refer to:

sspmv, dspmv

## 2.6.6. cublas<t>spr()

```
cublasStatus_t cublasSspr(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const float  *alpha,
                          const float  *x, int incx, float  *AP)
cublasStatus_t cublasDspr(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const double *alpha,
                          const double *x, int incx, double *AP)
```

This function performs the packed symmetric rank-1 update

$A = \alpha \mathbf{x}\mathbf{x}^T + A$

where $A$ is a $n \times n$ symmetric matrix stored in packed format, $\mathbf{x}$ is a vector, and $\alpha$ is a scalar.

If **uplo == CUBLAS_FILL_MODE_LOWER** then the elements in the lower triangular part of the symmetric matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+((2*n-j+1)*j)/2]** for $j = 1, \ldots, n$ and $i \geq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

If **uplo == CUBLAS_FILL_MODE_UPPER** then the elements in the upper triangular part of the symmetric matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+(j*(j+1))/2]** for $j = 1, \ldots, n$ and $i \leq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix $A$ lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix $A$. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| AP | device | in/out | <type> array with $A$ stored in packed format. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sspr, dspr

## 2.6.7. cublas<t>spr2()

```
cublasStatus_t cublasSspr2(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const float  *alpha,
                           const float  *x, int incx,
                           const float  *y, int incy, float  *AP)
cublasStatus_t cublasDspr2(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const double *alpha,
                           const double *x, int incx,
                           const double *y, int incy, double *AP)
```

This function performs the packed symmetric rank-2 update

$A = \alpha(\mathbf{x}\mathbf{y}^T + \mathbf{y}\mathbf{x}^T) + A$

where $A$ is a $n \times n$ symmetric matrix stored in packed format, $\mathbf{x}$ is a vector, and $\alpha$ is a scalar.

If **uplo == CUBLAS_FILL_MODE_LOWER** then the elements in the lower triangular part of the symmetric matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+((2*n-j+1)*j)/2]** for $j = 1, \ldots, n$ and $i \geq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

If **uplo == CUBLAS_FILL_MODE_UPPER** then the elements in the upper triangular part of the symmetric matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+(j*(j+1))/2]** for $j = 1, \ldots, n$ and $i \leq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix $A$ lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix $A$. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| y | device | input | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |
| AP | device | in/out | <type> array with $A$ stored in packed format. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sspr2, dspr2

## 2.6.8. cublas<t>symv()

```
cublasStatus_t cublasSsymv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const float           *alpha,
                           const float           *A, int lda,
                           const float           *x, int incx, const float
      *beta,
                           float           *y, int incy)
cublasStatus_t cublasDsymv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const double          *alpha,
                           const double          *A, int lda,
                           const double          *x, int incx, const double
      *beta,
                           double          *y, int incy)
cublasStatus_t cublasCsymv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuComplex       *alpha, /* host or
 device pointer */
                           const cuComplex       *A, int lda,
                           const cuComplex       *x, int incx, const cuComplex
      *beta,
                           cuComplex       *y, int incy)
cublasStatus_t cublasZsymv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *x, int incx, const
 cuDoubleComplex *beta,
                           cuDoubleComplex *y, int incy)
```

This function performs the symmetric matrix-vector multiplication.

$$\mathbf{y} = \alpha A\mathbf{x} + \beta\mathbf{y}$$

where $A$ is a $n \times n$ symmetric matrix stored in lower or upper mode, $\mathbf{x}$ and $\mathbf{x}$ are vectors, and $\alpha$ and $\beta$ are scalars.

This function has an alternate faster implementation using atomics that can be enabled with **cublasSetAtomicsMode()**.

Please see the section on the function **cublasSetAtomicsMode()** for more details about the usage of atomics.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x n** with **lda>=max(1,n)**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta==0** then **y** does not have to be a valid input. |
| y | device | in/out | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

ssymv, dsymv

## 2.6.9. cublas<t>syr()

```
cublasStatus_t cublasSsyr(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const float           *alpha,
                          const float           *x, int incx, float
 *A, int lda)
cublasStatus_t cublasDsyr(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const double          *alpha,
                          const double          *x, int incx, double
 *A, int lda)
cublasStatus_t cublasCsyr(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const cuComplex       *alpha,
                          const cuComplex       *x, int incx, cuComplex
 *A, int lda)
cublasStatus_t cublasZsyr(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const cuDoubleComplex *alpha,
                          const cuDoubleComplex *x, int incx, cuDoubleComplex
 *A, int lda)
```

This function performs the symmetric rank-1 update

$A = \alpha \mathbf{x}\mathbf{x}^T + A$

where $A$ is a $n \times n$ symmetric matrix stored in column-major format, $\mathbf{x}$ is a vector, and $\alpha$ is a scalar.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| A | device | in/out | <type> array of dimensions **lda x n**, with **lda>=max(1,n)**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

ssyr, dsyr

## 2.6.10. cublas<t>syr2()

```
cublasStatus_t cublasSsyr2(cublasHandle_t handle, cublasFillMode_t uplo, int n,
                           const float           *alpha, const float
 *x, int incx,
                           const float           *y, int incy, float
 *A, int lda
cublasStatus_t cublasDsyr2(cublasHandle_t handle, cublasFillMode_t uplo, int n,
                           const double          *alpha, const double
 *x, int incx,
                           const double          *y, int incy, double
 *A, int lda
cublasStatus_t cublasCsyr2(cublasHandle_t handle, cublasFillMode_t uplo, int n,
                           const cuComplex       *alpha, const cuComplex
 *x, int incx,
                           const cuComplex       *y, int incy, cuComplex
 *A, int lda
cublasStatus_t cublasZsyr2(cublasHandle_t handle, cublasFillMode_t uplo, int n,
                           const cuDoubleComplex *alpha, const cuDoubleComplex
 *x, int incx,
                           const cuDoubleComplex *y, int incy, cuDoubleComplex
 *A, int lda
```

This function performs the symmetric rank-2 update

$$A = \alpha(\mathbf{x}\mathbf{y}^T + \mathbf{y}\mathbf{x}^T) + A$$

where $A$ is a $n \times n$ symmetric matrix stored in column-major format, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ is a scalar.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | input | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |
| A | device | in/out | <type> array of dimensions **lda x n**, with **lda>=max(1,n)**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

ssyr2, dsyr2

## 2.6.11. cublas<t>tbmv()

```
cublasStatus_t cublasStbmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, int k, const float           *A, int lda,
                           float           *x, int incx)
cublasStatus_t cublasDtbmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, int k, const double          *A, int lda,
                           double          *x, int incx)
cublasStatus_t cublasCtbmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, int k, const cuComplex       *A, int lda,
                           cuComplex       *x, int incx)
cublasStatus_t cublasZtbmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, int k, const cuDoubleComplex *A, int lda,
                           cuDoubleComplex *x, int incx)
```

This function performs the triangular banded matrix-vector multiplication

$\mathbf{x} = \mathrm{op}(A)\mathbf{x}$

where $A$ is a triangular banded matrix, and $\mathbf{x}$ is a vector. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

If **uplo == CUBLAS_FILL_MODE_LOWER** then the triangular banded matrix $A$ is stored column by column, with the main diagonal of the matrix stored in row **1**, the first subdiagonal in row **2** (starting at first position), the second subdiagonal in row **3** (starting at first position), etc. So that in general, the element $A(i, j)$ is stored in the memory location **A(1+i-j,j)** for $j = 1, \ldots, n$ and $i \in [j, \min(m, j+k)]$. Also, the elements in the array **A** that do not conceptually correspond to the elements in the banded matrix (the bottom right $k \times k$ triangle) are not referenced.

If **uplo == CUBLAS_FILL_MODE_UPPER** then the triangular banded matrix $A$ is stored column by column, with the main diagonal of the matrix stored in row **k+1**, the first superdiagonal in row **k** (starting at second position), the second superdiagonal in row **k-1** (starting at third position), etc. So that in general, the element $A(i, j)$ is stored in the memory location **A(1+k+i-j,j)** for $j = 1, \ldots, n$ and $i \in [\max(1, j-k, j)]$. Also, the

elements in the array **A** that do not conceptually correspond to the elements in the banded matrix (the top left $k \times k$ triangle) are not referenced.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix **A** are unity and should not be accessed. |
| n | | input | number of rows and columns of matrix **A**. |
| k | | input | number of sub- and super-diagonals of matrix . |
| A | device | input | <type> array of dimension **lda x n**, with **lda>=k+1**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| x | device | in/out | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** or **incx=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_ALLOC_FAILED** | the allocation of internal scratch memory failed |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

stbmv, dtbmv, ctbmv, ztbmv

## 2.6.12. cublas<t>tbsv()

```
cublasStatus_t cublasStbsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, int k, const float           *A, int lda,
                           float           *x, int incx)
cublasStatus_t cublasDtbsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, int k, const double          *A, int lda,
                           double          *x, int incx)
cublasStatus_t cublasCtbsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, int k, const cuComplex       *A, int lda,
                           cuComplex       *x, int incx)
cublasStatus_t cublasZtbsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, int k, const cuDoubleComplex *A, int lda,
                           cuDoubleComplex *x, int incx)
```

This function solves the triangular banded linear system with a single right-hand-side

$$\text{op}(A)\mathbf{x} = \mathbf{b}$$

where $A$ is a triangular banded matrix, and $\mathbf{x}$ and $\mathbf{b}$ are vectors. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

The solution $\mathbf{x}$ overwrites the right-hand-sides $\mathbf{b}$ on exit.

No test for singularity or near-singularity is included in this function.

If **`uplo == CUBLAS_FILL_MODE_LOWER`** then the triangular banded matrix $A$ is stored column by column, with the main diagonal of the matrix stored in row **`1`**, the first subdiagonal in row **`2`** (starting at first position), the second subdiagonal in row **`3`** (starting at first position), etc. So that in general, the element $A(i, j)$ is stored in the memory location **`A(1+i-j,j)`** for $j = 1, \ldots, n$ and $i \in [j, \min(m, j+k)]$. Also, the elements in the array **`A`** that do not conceptually correspond to the elements in the banded matrix (the bottom right $k \times k$ triangle) are not referenced.

If **`uplo == CUBLAS_FILL_MODE_UPPER`** then the triangular banded matrix $A$ is stored column by column, with the main diagonal of the matrix stored in row **`k+1`**, the first superdiagonal in row **`k`** (starting at second position), the second superdiagonal in row **`k-1`** (starting at third position), etc. So that in general, the element $A(i, j)$ is stored in the memory location **`A(1+k+i-j,j)`** for $j = 1, \ldots, n$ and $i \in [\max(1, j-k, j)]$. Also, the elements in the array **`A`** that do not conceptually correspond to the elements in the banded matrix (the top left $k \times k$ triangle) are not referenced.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **`A`** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**`A`**) that is non- or (conj.) transpose. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| diag | | input | indicates if the elements on the main diagonal of matrix **A** are unity and should not be accessed. |
| n | | input | number of rows and columns of matrix **A**. |
| k | | input | number of sub- and super-diagonals of matrix **A**. |
| A | device | input | <type> array of dimension **lda x n**, with **lda >= k+1**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| x | device | in/out | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** or **incx=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

stbsv, dtbsv, ctbsv, ztbsv

## 2.6.13. cublas<t>tpmv()

```
cublasStatus_t cublasStpmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const float           *AP,
                           float           *x, int incx)
cublasStatus_t cublasDtpmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const double          *AP,
                           double          *x, int incx)
cublasStatus_t cublasCtpmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const cuComplex       *AP,
                           cuComplex       *x, int incx)
cublasStatus_t cublasZtpmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const cuDoubleComplex *AP,
                           cuDoubleComplex *x, int incx)
```

This function performs the triangular packed matrix-vector multiplication

$$\mathbf{x} = \mathrm{op}(A)\mathbf{x}$$

where $A$ is a triangular matrix stored in packed format, and $\mathbf{x}$ is a vector. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

If **uplo == CUBLAS_FILL_MODE_LOWER** then the elements in the lower triangular part of the triangular matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+((2*n-j+1)*j)/2]** for $j = 1, \ldots, n$ and $i \geq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

If **uplo == CUBLAS_FILL_MODE_UPPER** then the elements in the upper triangular part of the triangular matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+(j*(j+1))/2]** for $A(i, j)$ and $i \leq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix **A** are unity and should not be accessed. |
| n | | input | number of rows and columns of matrix **A**. |
| AP | device | input | <type> array with $A$ stored in packed format. |
| x | device | in/out | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters $n<0 or **incx=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_ALLOC_FAILED** | the allocation of internal scratch memory failed |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

stpmv, dtpmv, ctpmv, ztpmv

## 2.6.14. cublas<t>tpsv()

```
cublasStatus_t cublasStpsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const float           *AP,
                           float           *x, int incx)
cublasStatus_t cublasDtpsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const double          *AP,
                           double          *x, int incx)
cublasStatus_t cublasCtpsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const cuComplex       *AP,
                           cuComplex       *x, int incx)
cublasStatus_t cublasZtpsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const cuDoubleComplex *AP,
                           cuDoubleComplex *x, int incx)
```

This function solves the packed triangular linear system with a single right-hand-side

$$\text{op}(A)\mathbf{x} = \mathbf{b}$$

where $A$ is a triangular matrix stored in packed format, and $\mathbf{x}$ and $\mathbf{b}$ are vectors. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

The solution $\mathbf{x}$ overwrites the right-hand-sides $\mathbf{b}$ on exit.

No test for singularity or near-singularity is included in this function.

If **`uplo == CUBLAS_FILL_MODE_LOWER`** then the elements in the lower triangular part of the triangular matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **`AP[i+((2*n-j+1)*j)/2]`** for $j = 1, \ldots, n$ and $i \geq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

If **`uplo == CUBLAS_FILL_MODE_UPPER`** then the elements in the upper triangular part of the triangular matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **`AP[i+(j*(j+1))/2]`** for $j = 1, \ldots, n$ and $i \leq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix are unity and should not be accessed. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| n | | input | number of rows and columns of matrix **A**. |
| AP | device | input | <type> array with **A** stored in packed format. |
| x | device | in/out | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

stpsv, dtpsv, ctpsv, ztpsv

## 2.6.15. cublas<t>trmv()

```
cublasStatus_t cublasStrmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const float           *A, int lda,
                           float           *x, int incx)
cublasStatus_t cublasDtrmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const double          *A, int lda,
                           double          *x, int incx)
cublasStatus_t cublasCtrmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const cuComplex       *A, int lda,
                           cuComplex       *x, int incx)
cublasStatus_t cublasZtrmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const cuDoubleComplex *A, int lda,
                           cuDoubleComplex *x, int incx)
```

This function performs the triangular matrix-vector multiplication

$$\mathbf{x} = \mathrm{op}(A)\mathbf{x}$$

where $A$ is a triangular matrix stored in lower or upper mode with or without the main diagonal, and $\mathbf{x}$ is a vector. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix **A** are unity and should not be accessed. |
| n | | input | number of rows and columns of matrix **A**. |
| A | device | input | <type> array of dimensions **lda x n** , with **lda>=max(1,n)**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| x | device | in/out | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_ALLOC_FAILED** | the allocation of internal scratch memory failed |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

strmv, dtrmv, ctrmv, ztrmv

## 2.6.16. cublas<t>trsv()

```
cublasStatus_t cublasStrsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const float           *A, int lda,
                           float           *x, int incx)
cublasStatus_t cublasDtrsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const double          *A, int lda,
                           double          *x, int incx)
cublasStatus_t cublasCtrsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const cuComplex       *A, int lda,
                           cuComplex       *x, int incx)
cublasStatus_t cublasZtrsv(cublasHandle_t handle, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int n, const cuDoubleComplex *A, int lda,
                           cuDoubleComplex *x, int incx)
```

This function solves the triangular linear system with a single right-hand-side

$$\text{op}(A)\mathbf{x} = \mathbf{b}$$

where $A$ is a triangular matrix stored in lower or upper mode with or without the main diagonal, and $\mathbf{x}$ and $\mathbf{b}$ are vectors. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

The solution $\mathbf{x}$ overwrites the right-hand-sides $\mathbf{b}$ on exit.

No test for singularity or near-singularity is included in this function.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix **A** are unity and should not be accessed. |
| n | | input | number of rows and columns of matrix **A**. |
| A | device | input | <type> array of dimension **lda x n**, with **lda>=max(1,n)**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| x | device | in/out | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

strsv, dtrsv, ctrsv, ztrsv

## 2.6.17. cublas<t>hemv()

```
cublasStatus_t cublasChemv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *x, int incx,
                           const cuComplex       *beta,
                           cuComplex       *y, int incy)
cublasStatus_t cublasZhemv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *y, int incy)
```

This function performs the Hermitian matrix-vector multiplication

$$\mathbf{y} = \alpha A\mathbf{x} + \beta\mathbf{y}$$

where $A$ is a $n \times n$ Hermitian matrix stored in lower or upper mode, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ and $\beta$ are scalars.

This function has an alternate faster implementation using atomics that can be enabled with

Please see the section on the for more details about the usage of atomics

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x n**, with **lda>=max(1,n)**. The imaginary parts of the diagonal elements are assumed to be zero. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta==0** then **y** does not have to be a valid input. |
| y | device | in/out | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

chemv, zhemv

## 2.6.18. cublas<t>hbmv()

```
cublasStatus_t cublasChbmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, int k, const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *x, int incx,
                           const cuComplex       *beta,
                           cuComplex       *y, int incy)
cublasStatus_t cublasZhbmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, int k, const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *y, int incy)
```

This function performs the Hermitian banded matrix-vector multiplication

$$\mathbf{y} = \alpha A\mathbf{x} + \beta\mathbf{y}$$

where $A$ is a $n \times n$ Hermitian banded matrix with $k$ subdiagonals and superdiagonals, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ and $\beta$ are scalars.

If **uplo == CUBLAS_FILL_MODE_LOWER** then the Hermitian banded matrix $A$ is stored column by column, with the main diagonal of the matrix stored in row **1**, the first subdiagonal in row **2** (starting at first position), the second subdiagonal in row **3** (starting at first position), etc. So that in general, the element $A(i, j)$ is stored in the memory location **A(1+i-j,j)** for $j = 1, \ldots, n$ and $i \in [j, \min(m, j+k)]$. Also, the elements in the array **A** that do not conceptually correspond to the elements in the banded matrix (the bottom right $k \times k$ triangle) are not referenced.

If **uplo == CUBLAS_FILL_MODE_UPPER** then the Hermitian banded matrix $A$ is stored column by column, with the main diagonal of the matrix stored in row **k+1**, the first superdiagonal in row **k** (starting at second position), the second superdiagonal in row **k-1** (starting at third position), etc. So that in general, the element $A(i, j)$ is stored in the memory location **A(1+k+i-j,j)** for $j = 1, \ldots, n$ and $i \in [\max(1, j-k), j]$. Also, the elements in the array **A** that do not conceptually correspond to the elements in the banded matrix (the top left $k \times k$ triangle) are not referenced.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| k | | input | number of sub- and super-diagonals of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimensions **lda x n**, with **lda>=k+1**. The imaginary parts of the diagonal elements are assumed to be zero. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta==0** then does not have to be a valid input. |
| y | device | in/out | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

chbmv, zhbmv

## 2.6.19. cublas<t>hpmv()

```
cublasStatus_t cublasChpmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuComplex       *alpha,
                           const cuComplex       *AP,
                           const cuComplex       *x, int incx,
                           const cuComplex       *beta,
                           cuComplex       *y, int incy)
cublasStatus_t cublasZhpmv(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuDoubleComplex *alpha,
                           const cuDoubleComplex *AP,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *y, int incy)
```

This function performs the Hermitian packed matrix-vector multiplication

$\mathbf{y} = \alpha A\mathbf{x} + \beta\mathbf{y}$

where $A$ is a $n \times n$ Hermitian matrix stored in packed format, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ and $\beta$ are scalars.

If **`uplo == CUBLAS_FILL_MODE_LOWER`** then the elements in the lower triangular part of the Hermitian matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **`AP[i+((2*n-j+1)*j)/2]`** for $j = 1, \ldots, n$ and $i \geq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

If **`uplo == CUBLAS_FILL_MODE_UPPER`** then the elements in the upper triangular part of the Hermitian matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **`AP[i+(j*(j+1))/2]`** for $j = 1, \ldots, n$ and $i \leq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| AP | device | input | <type> array with **A** stored in packed format. The imaginary parts of the diagonal elements are assumed to be zero. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta==0** then **y** does not have to be a valid input. |
| y | device | in/out | <type> vector with **n** elements. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| incy | | input | stride between consecutive elements of **y**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

chpmv, zhpmv

## 2.6.20. cublas<t>her()

```
cublasStatus_t cublasCher(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const float  *alpha,
                          const cuComplex       *x, int incx,
                          cuComplex       *A, int lda)
cublasStatus_t cublasZher(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const double *alpha,
                          const cuDoubleComplex *x, int incx,
                          cuDoubleComplex *A, int lda)
```

This function performs the Hermitian rank-1 update

$A = \alpha \mathbf{x}\mathbf{x}^H + A$

where $A$ is a $n \times n$ Hermitian matrix stored in column-major format, $\mathbf{x}$ is a vector, and $\alpha$ is a scalar.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| A | device | in/out | <type> array of dimensions **lda x n**, with **lda>=max(1,n)**. The imaginary parts of the diagonal elements are assumed and set to zero. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

cher, zher

## 2.6.21. cublas<t>her2()

```
cublasStatus_t cublasCher2(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuComplex       *alpha,
                           const cuComplex       *x, int incx,
                           const cuComplex       *y, int incy,
                           cuComplex       *A, int lda)
cublasStatus_t cublasZher2(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuDoubleComplex *alpha,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *y, int incy,
                           cuDoubleComplex *A, int lda)
```

This function performs the Hermitian rank-2 update

$A = \alpha \mathbf{x}\mathbf{y}^H + \bar{\alpha}\mathbf{y}\mathbf{x}^H + A$

where $A$ is a $n \times n$ Hermitian matrix stored in column-major format, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ is a scalar.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | input | <type> vector with **n** elements. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| incy | | input | stride between consecutive elements of **y**. |
| A | device | in/out | <type> array of dimension **lda x n** with **lda>=max(1,n)**. The imaginary parts of the diagonal elements are assumed and set to zero. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

cher2, zher2

## 2.6.22. cublas<t>hpr()

```
cublasStatus_t cublasChpr(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const float *alpha,
                          const cuComplex       *x, int incx,
                          cuComplex       *AP)
cublasStatus_t cublasZhpr(cublasHandle_t handle, cublasFillMode_t uplo,
                          int n, const double *alpha,
                          const cuDoubleComplex *x, int incx,
                          cuDoubleComplex *AP)
```

This function performs the packed Hermitian rank-1 update

$A = \alpha \mathbf{x}\mathbf{x}^H + A$

where $A$ is a $n \times n$ Hermitian matrix stored in packed format, $\mathbf{x}$ is a vector, and $\alpha$ is a scalar.

If **uplo == CULBAS_FILL_MODE_LOWER** then the elements in the lower triangular part of the Hermitian matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+((2*n-j+1)*j)/2]** for $j = 1, \ldots, n$ and $i \geq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

If **uplo == CULBAS_FILL_MODE_UPPER** then the elements in the upper triangular part of the Hermitian matrix $A$ are packed together column by column without gaps, so that

the element $A(i, j)$ is stored in the memory location **AP[i+(j*(j+1))/2]** for $j = 1, \ldots, n$ and $i \le j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| AP | device | in/out | <type> array with **A** stored in packed format. The imaginary parts of the diagonal elements are assumed and set to zero. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

chpr, zhpr

## 2.6.23. cublas<t>hpr2()

```
cublasStatus_t cublasChpr2(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuComplex       *alpha,
                           const cuComplex       *x, int incx,
                           const cuComplex       *y, int incy,
                           cuComplex       *AP)
cublasStatus_t cublasZhpr2(cublasHandle_t handle, cublasFillMode_t uplo,
                           int n, const cuDoubleComplex *alpha,
                           const cuDoubleComplex *x, int incx,
                           const cuDoubleComplex *y, int incy,
                           cuDoubleComplex *AP)
```

This function performs the packed Hermitian rank-2 update

$$A = \alpha \mathbf{x}\mathbf{y}^H + \bar{\alpha}\mathbf{y}\mathbf{x}^H + A$$

where $A$ is a $n \times n$ Hermitian matrix stored in packed format, $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\alpha$ is a scalar.

If **uplo == CULBAS_FILL_MODE_LOWER** then the elements in the lower triangular part of the Hermitian matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+((2*n-j+1)*j)/2]** for $j = 1, \ldots, n$ and $i \geq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

If **uplo == CULBAS_FILL_MODE_UPPER** then the elements in the upper triangular part of the Hermitian matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+(j*(j+1))/2]** for $j = 1, \ldots, n$ and $i \leq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| n | | input | number of rows and columns of matrix **A**. |
| alpha | host or device | input | <type> scalar used for multiplication. |
| x | device | input | <type> vector with **n** elements. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | input | <type> vector with **n** elements. |
| incy | | input | stride between consecutive elements of **y**. |
| AP | device | in/out | <type> array with **A** stored in packed format. The imaginary parts of the diagonal elements are assumed and set to zero. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** or **incx,incy=0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

chpr2, zhpr2

# 2.7. cuBLAS Level-3 Function Reference

In this chapter we describe the Level-3 Basic Linear Algebra Subprograms (BLAS3) functions that perform matrix-matrix operations.

## 2.7.1. cublas<t>gemm()

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *B, int ldb,
                           const float           *beta,
                           float           *C, int ldc)
cublasStatus_t cublasDgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const double          *alpha,
                           const double          *A, int lda,
                           const double          *B, int ldb,
                           const double          *beta,
                           double          *C, int ldc)
cublasStatus_t cublasCgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *B, int ldb,
                           const cuComplex       *beta,
                           cuComplex       *C, int ldc)
cublasStatus_t cublasZgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *B, int ldb,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *C, int ldc)
cublasStatus_t cublasHgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const __half *alpha,
                           const __half *A, int lda,
                           const __half *B, int ldb,
                           const __half *beta,
                           __half *C, int ldc)
```

This function performs the matrix-matrix multiplication

$$C = \alpha \text{op}(A)\text{op}(B) + \beta C$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are matrices stored in column-major format with dimensions $\text{op}(A)$ $m \times k$, $\text{op}(B)$ $k \times n$ and $C$ $m \times n$, respectively. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\text{op}(B)$ is defined similarly for matrix $B$.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| transa | | input | operation op(**A**) that is non- or (conj.) transpose. |
| transb | | input | operation op(**B**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(**A**) and **C**. |
| n | | input | number of columns of matrix op(**B**) and **C**. |
| k | | input | number of columns of op(**A**) and rows of op(**B**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimensions **lda x k** with **lda>=max(1,m)** if **transa == CUBLAS_OP_N** and **lda x m** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store the matrix **A**. |
| B | device | input | <type> array of dimension **ldb x n** with **ldb>=max(1,k)** if **transa == CUBLAS_OP_N** and **ldb x k** with **ldb>=max(1,n)** otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host or device | input | <type> scalar used for multiplication. If **beta==0**, **C** does not have to be a valid input. |
| C | device | in/out | <type> array of dimensions **ldc x n** with **ldc>=max(1,m)**. |
| ldc | | input | leading dimension of a two-dimensional array used to store the matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n,k<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision or in the case of **cublasHgemm** the device does not support math in half precision. |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sgemm, dgemm, cgemm, zgemm

## 2.7.2. cublas<t>gemm3m()

```
cublasStatus_t cublasCgemm3m(cublasHandle_t handle,
                             cublasOperation_t transa, cublasOperation_t transb,
                             int m, int n, int k,
                             const cuComplex       *alpha,
                             const cuComplex       *A, int lda,
                             const cuComplex       *B, int ldb,
                             const cuComplex       *beta,
                             cuComplex       *C, int ldc)
cublasStatus_t cublasZgemm3m(cublasHandle_t handle,
                             cublasOperation_t transa, cublasOperation_t transb,
                             int m, int n, int k,
                             const cuDoubleComplex *alpha,
                             const cuDoubleComplex *A, int lda,
                             const cuDoubleComplex *B, int ldb,
                             const cuDoubleComplex *beta,
                             cuDoubleComplex *C, int ldc)
```

This function performs the complex matrix-matrix multiplication, using Gauss complexity reduction algorithm. This can lead to an increase in performance up to 25%

$$C = \alpha \text{op}(A)\text{op}(B) + \beta C$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are matrices stored in column-major format with dimensions $\text{op}(A)$ $m \times k$, $\text{op}(B)$ $k \times n$ and $C$ $m \times n$, respectively. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\text{op}(B)$ is defined similarly for matrix $B$.

These 2 routines are only supported on GPUs with architecture capabilities equal or greater than 5.0

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| transa | | input | operation op(**A**) that is non- or (conj.) transpose. |
| transb | | input | operation op(**B**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(**A**) and **C**. |
| n | | input | number of columns of matrix op(**B**) and **C**. |
| k | | input | number of columns of op(**A**) and rows of op(**B**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimensions **lda x k** with **lda>=max(1,m)** if **transa == CUBLAS_OP_N** and **lda x m** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store the matrix **A**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| B | device | input | <type> array of dimension `ldb x n` with `ldb>=max(1,k)` if `transa == CUBLAS_OP_N` and `ldb x k` with `ldb>=max(1,n)` otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix `B`. |
| beta | host or device | input | <type> scalar used for multiplication. If `beta==0`, `C` does not have to be a valid input. |
| C | device | in/out | <type> array of dimensions `ldc x n` with `ldc>=max(1,m)`. |
| ldc | | input | leading dimension of a two-dimensional array used to store the matrix `C`. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `m,n,k<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device has a compute capabilites lower than 5.0 |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

cgemm, zgemm

## 2.7.3. cublas<t>gemmBatched()

```
cublasStatus_t cublasHgemmBatched(cublasHandle_t handle,
                                  cublasOperation_t transa,
                                  cublasOperation_t transb,
                                  int m, int n, int k,
                                  const __half           *alpha,
                                  const __half           *Aarray[], int lda,
                                  const __half           *Barray[], int ldb,
                                  const __half           *beta,
                                  __half           *Carray[], int ldc,
                                  int batchCount)
cublasStatus_t cublasSgemmBatched(cublasHandle_t handle,
                                  cublasOperation_t transa,
                                  cublasOperation_t transb,
                                  int m, int n, int k,
                                  const float           *alpha,
                                  const float           *Aarray[], int lda,
                                  const float           *Barray[], int ldb,
                                  const float           *beta,
                                  float           *Carray[], int ldc,
                                  int batchCount)
cublasStatus_t cublasDgemmBatched(cublasHandle_t handle,
                                  cublasOperation_t transa,
                                  cublasOperation_t transb,
                                  int m, int n, int k,
                                  const double          *alpha,
                                  const double          *Aarray[], int lda,
                                  const double          *Barray[], int ldb,
                                  const double          *beta,
                                  double          *Carray[], int ldc,
                                  int batchCount)
cublasStatus_t cublasCgemmBatched(cublasHandle_t handle,
                                  cublasOperation_t transa,
                                  cublasOperation_t transb,
                                  int m, int n, int k,
                                  const cuComplex       *alpha,
                                  const cuComplex       *Aarray[], int lda,
                                  const cuComplex       *Barray[], int ldb,
                                  const cuComplex       *beta,
                                  cuComplex       *Carray[], int ldc,
                                  int batchCount)
cublasStatus_t cublasZgemmBatched(cublasHandle_t handle,
                                  cublasOperation_t transa,
                                  cublasOperation_t transb,
                                  int m, int n, int k,
                                  const cuDoubleComplex *alpha,
                                  const cuDoubleComplex *Aarray[], int lda,
                                  const cuDoubleComplex *Barray[], int ldb,
                                  const cuDoubleComplex *beta,
                                  cuDoubleComplex *Carray[], int ldc,
                                  int batchCount)
```

This function performs the matrix-matrix multiplication of a batch of matrices. The batch is considered to be "uniform", i.e. all instances have the same dimensions (m, n, k), leading dimensions (lda, ldb, ldc) and transpositions (transa, transb) for their respective A, B and C matrices. The address of the input matrices and the output matrix of each instance of the batch are read from arrays of pointers passed to the function by the caller.

$$C[i] = \alpha \mathrm{op}(A[i]) \mathrm{op}(B[i]) + \beta C[i], \text{ for i } \in [0, batchCount - 1]$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are arrays of pointers to matrices stored in column-major format with dimensions $\text{op}(A[i])\ m \times k$, $\text{op}(B[i])\ k \times n$ and $C[i]\ m \times n$, respectively. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\text{op}(B[i])$ is defined similarly for matrix $B[i]$.

On certain problem sizes, it might be advantageous to make multiple calls to **cublas<t>gemm** in different CUDA streams, rather than use this API.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| transa | | input | operation op(**A[i]**) that is non- or (conj.) transpose. |
| transb | | input | operation op(**B[i]**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(**A[i]**) and **C[i]**. |
| n | | input | number of columns of op(**B[i]**) and **C[i]**. |
| k | | input | number of columns of op(**A[i]**) and rows of op(**B[i]**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| Aarray | device | input | array of pointers to <type> array, with each array of dim. **lda x k** with **lda>=max(1,m)** if **transa==CUBLAS_OP_N** and **lda x m** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store each matrix **A[i]**. |
| Barray | device | input | array of pointers to <type> array, with each array of dim. **ldb x n** with **ldb>=max(1,k)** if **transa==CUBLAS_OP_N** and **ldb x k** with **ldb>=max(1,n)** max(1,) otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store each matrix **B[i]**. |
| beta | host or device | input | <type> scalar used for multiplication. If **beta == 0**, **C** does not have to be a valid input. |
| Carray | device | in/out | array of pointers to <type> array. It has dimensions **ldc x n** with **ldc>=max(1,m)**. |
| ldc | | input | leading dimension of two-dimensional array used to store each matrix **C[i]**. |
| batchCount | | input | number of pointers contained in Aarray, Barray and Carray. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `m,n,k,batchCount<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

## 2.7.4. cublas<t>gemmStridedBatched()

```
cublasStatus_t cublasHgemmStridedBatched(cublasHandle_t handle,
                                         cublasOperation_t transa,
                                         cublasOperation_t transb,
                                         int m, int n, int k,
                                         const __half           *alpha,
                                         const __half           *A, int lda,
                                         long long int          strideA,
                                         const __half           *B, int ldb,
                                         long long int          strideB,
                                         const __half           *beta,
                                         __half                 *C, int ldc,
                                         long long int          strideC,
                                         int batchCount)
cublasStatus_t cublasSgemmStridedBatched(cublasHandle_t handle,
                                         cublasOperation_t transa,
                                         cublasOperation_t transb,
                                         int m, int n, int k,
                                         const float            *alpha,
                                         const float            *A, int lda,
                                         long long int          strideA,
                                         const float            *B, int ldb,
                                         long long int          strideB,
                                         const float            *beta,
                                         float                  *C, int ldc,
                                         long long int          strideC,
                                         int batchCount)
cublasStatus_t cublasDgemmStridedBatched(cublasHandle_t handle,
                                         cublasOperation_t transa,
                                         cublasOperation_t transb,
                                         int m, int n, int k,
                                         const double           *alpha,
                                         const double           *A, int lda,
                                         long long int          strideA,
                                         const double           *B, int ldb,
                                         long long int          strideB,
                                         const double           *beta,
                                         double                 *C, int ldc,
                                         long long int          strideC,
                                         int batchCount)
cublasStatus_t cublasCgemmStridedBatched(cublasHandle_t handle,
                                         cublasOperation_t transa,
                                         cublasOperation_t transb,
                                         int m, int n, int k,
                                         const cuComplex        *alpha,
                                         const cuComplex        *A, int lda,
                                         long long int          strideA,
                                         const cuComplex        *B, int ldb,
                                         long long int          strideB,
                                         const cuComplex        *beta,
                                         cuComplex              *C, int ldc,
                                         long long int          strideC,
                                         int batchCount)
cublasStatus_t cublasCgemm3mStridedBatched(cublasHandle_t handle,
                                         cublasOperation_t transa,
                                         cublasOperation_t transb,
                                         int m, int n, int k,
                                         const cuComplex        *alpha,
                                         const cuComplex        *A, int lda,
                                         long long int          strideA,
                                         const cuComplex        *B, int ldb,
                                         long long int          strideB,
                                         const cuComplex        *beta,
                                         cuComplex              *C, int ldc,
                                         long long int          strideC,
                                         int batchCount)
cublasStatus_t cublasZgemmStridedBatched(cublasHandle_t handle,
                                         cublasOperation_t transa,
                                         cublasOperation_t transb,
                                         int m, int n, int k,
                                         const cuDoubleComplex *alpha,
                                         const cuDoubleComplex *A, int lda,
                                         long long int          strideA,
```

This function performs the matrix-matrix multiplication of a batch of matrices. The batch is considered to be "uniform", i.e. all instances have the same dimensions (m, n, k), leading dimensions (lda, ldb, ldc) and transpositions (transa, transb) for their respective A, B and C matrices. Input matrices A, B and output matrix C for each instance of the batch are located at fixed address offsets from their locations in the previous instance. Pointers to A, B and C matrices for the first instance are passed to the function by the user along with the address offsets - strideA, strideB and strideC that determine the locations of input and output matrices in future instances.

$$C + i*\text{strideC} = \alpha\text{op}(A + i*\text{strideA})\text{op}(B + i*\text{strideB}) + \beta(C + i*\text{strideC}), \text{ for i } \in [0, batchCount - 1]$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are arrays of pointers to matrices stored in column-major format with dimensions $\text{op}(A[i])$ $m \times k$, $\text{op}(B[i])$ $k \times n$ and $C[i]$ $m \times n$, respectively. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\text{op}(B[i])$ is defined similarly for matrix $B[i]$.

On certain problem sizes, it might be advantageous to make multiple calls to **cublas<t>gemm** in different CUDA streams, rather than use this API.

Note: In the table below, we use **A[i], B[i], C[i]** as notation for A, B and C matrices in the ith instance of the batch, implicitly assuming they are respectively address offsets **strideA, strideB, strideC** away from **A[i-1], B[i-1], C[i-1]**.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| transa | | input | operation op(**A[i]**) that is non- or (conj.) transpose. |
| transb | | input | operation op(**B[i]**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(**A[i]**) and **C[i]**. |
| n | | input | number of columns of op(**B[i]**) and **C[i]**. |
| k | | input | number of columns of op(**A[i]**) and rows of op(**B[i]**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type>* pointer to the A matrix corresponding to the first instance of the batch, with dimensions **lda x k** with **lda>=max(1,m)** if **transa==CUBLAS_OP_N** and **lda x m** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store each matrix **A[i]**. |
| strideA | | input | Value of type long long int that gives the address offset between **A[i]** and **A[i+1]** |
| B | device | input | <type>* pointer to the B matrix corresponding to the first instance of the batch, with dimensions **ldb x n** with |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| | | | `ldb>=max(1,k)` if `transa==CUBLAS_OP_N` and `ldb x k` with `ldb>=max(1,n)` max(1,) otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store each matrix `B[i]`. |
| strideB | | input | Value of type long long int that gives the address offset between `B[i]` and `B[i+1]` |
| beta | host or device | input | <type> scalar used for multiplication. If `beta == 0`, `C` does not have to be a valid input. |
| C | device | in/out | <type>* pointer to the C matrix corresponding to the first instance of the batch, with dimensions `ldc x n` with `ldc>=max(1,m)`. |
| ldc | | input | leading dimension of two-dimensional array used to store each matrix `C[i]`. |
| strideC | | input | Value of type long long int that gives the address offset between `C[i]` and `C[i+1]` |
| batchCount | | input | number of GEMMs to perform in the batch. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `m,n,k,batchCount<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

## 2.7.5. cublas<t>symm()

```
cublasStatus_t cublasSsymm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           int m, int n,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *B, int ldb,
                           const float           *beta,
                           float           *C, int ldc)
cublasStatus_t cublasDsymm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           int m, int n,
                           const double          *alpha,
                           const double          *A, int lda,
                           const double          *B, int ldb,
                           const double          *beta,
                           double          *C, int ldc)
cublasStatus_t cublasCsymm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           int m, int n,
                           const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *B, int ldb,
                           const cuComplex       *beta,
                           cuComplex       *C, int ldc)
cublasStatus_t cublasZsymm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           int m, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *B, int ldb,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *C, int ldc)
```

This function performs the symmetric matrix-matrix multiplication

$$C = \begin{cases} \alpha AB + \beta C & \text{if side == CUBLAS\_SIDE\_LEFT} \\ \alpha BA + \beta C & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A$ is a symmetric matrix stored in lower or upper mode, $B$ and $C$ are $m \times n$ matrices, and $\alpha$ and $\beta$ are scalars.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| side | | input | indicates if matrix **A** is on the left or right of **B**. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| m | | input | number of rows of matrix **C** and **B**, with matrix **A** sized accordingly. |
| n | | input | number of columns of matrix **C** and **B**, with matrix **A** sized accordingly. |
| alpha | host or device | input | <type> scalar used for multiplication. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| A | device | input | <type> array of dimension `lda x m` with `lda>=max(1,m)` if `side == CUBLAS_SIDE_LEFT` and `lda x n` with `lda>=max(1,n)` otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix `A`. |
| B | device | input | <type> array of dimension `ldb x n` with `ldb>=max(1,m)`. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix `B`. |
| beta | host or device | input | <type> scalar used for multiplication, if `beta == 0` then `C` does not have to be a valid input. |
| C | device | in/out | <type> array of dimension `ldc x n` with `ldc>=max(1,m)`. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix `C`. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `m,n<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

ssymm, dsymm, csymm, zsymm

## 2.7.6. cublas<t>syrk()

```
cublasStatus_t cublasSsyrk(cublasHandle_t handle,
                           cublasFillMode_t uplo, cublasOperation_t trans,
                           int n, int k,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *beta,
                           float           *C, int ldc)
cublasStatus_t cublasDsyrk(cublasHandle_t handle,
                           cublasFillMode_t uplo, cublasOperation_t trans,
                           int n, int k,
                           const double          *alpha,
                           const double          *A, int lda,
                           const double          *beta,
                           double          *C, int ldc)
cublasStatus_t cublasCsyrk(cublasHandle_t handle,
                           cublasFillMode_t uplo, cublasOperation_t trans,
                           int n, int k,
                           const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *beta,
                           cuComplex       *C, int ldc)
cublasStatus_t cublasZsyrk(cublasHandle_t handle,
                           cublasFillMode_t uplo, cublasOperation_t trans,
                           int n, int k,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *C, int ldc)
```

This function performs the symmetric rank- $k$ update

$$C = \alpha \mathrm{op}(A) \mathrm{op}(A)^T + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a symmetric matrix stored in lower or upper mode, and $A$ is a matrix with dimensions $\mathrm{op}(A)$ $n \times k$. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix C lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or transpose. |
| n | | input | number of rows of matrix op(**A**) and C. |
| k | | input | number of columns of matrix op(**A**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **trans == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| lda | | input | leading dimension of two-dimensional array used to store matrix A. |
| beta | host or device | input | <type> scalar used for multiplication, if `beta==0` then C does not have to be a valid input. |
| C | device | in/out | <type> array of dimension `ldc x n`, with `ldc>=max(1,n)`. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix C. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `n,k<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

ssyrk, dsyrk, csyrk, zsyrk

## 2.7.7. cublas<t>syr2k()

```
cublasStatus_t cublasSsyr2k(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const float           *alpha,
                            const float           *A, int lda,
                            const float           *B, int ldb,
                            const float           *beta,
                            float           *C, int ldc)
cublasStatus_t cublasDsyr2k(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const double          *alpha,
                            const double          *A, int lda,
                            const double          *B, int ldb,
                            const double          *beta,
                            double          *C, int ldc)
cublasStatus_t cublasCsyr2k(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const cuComplex       *alpha,
                            const cuComplex       *A, int lda,
                            const cuComplex       *B, int ldb,
                            const cuComplex       *beta,
                            cuComplex       *C, int ldc)
cublasStatus_t cublasZsyr2k(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const cuDoubleComplex *alpha,
                            const cuDoubleComplex *A, int lda,
                            const cuDoubleComplex *B, int ldb,
                            const cuDoubleComplex *beta,
                            cuDoubleComplex *C, int ldc)
```

This function performs the symmetric rank- $2k$ update

$$C = \alpha(\mathrm{op}(A)\mathrm{op}(B)^T + \mathrm{op}(B)\mathrm{op}(A)^T) + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a symmetric matrix stored in lower or upper mode, and $A$ and $B$ are matrices with dimensions $\mathrm{op}(A)$ $n \times k$ and $\mathrm{op}(B)$ $n \times k$, respectively. Also, for matrix $A$ and $B$

$$\mathrm{op}(A) \text{ and } \mathrm{op}(B) = \begin{cases} A \text{ and } B & \text{if trans == CUBLAS\_OP\_N} \\ A^T \text{ and } B^T & \text{if trans == CUBLAS\_OP\_T} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix c lower or upper part, is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or transpose. |
| n | | input | number of rows of matrix op(**A**), op(**B**) and **C**. |
| k | | input | number of columns of matrix op(**A**) and op(**B**). |
| alpha | host or device | input | <type> scalar used for multiplication. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **transa == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | device | input | <type> array of dimensions **ldb x k** with **ldb>=max(1,n)** if **transa == CUBLAS_OP_N** and **ldb x n** with **ldb>=max(1,k)** otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta==0**, then **C** does not have to be a valid input. |
| C | device | in/out | <type> array of dimensions **ldc x n** with **ldc>=max(1,n)**. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

ssyr2k, dsyr2k, csyr2k, zsyr2k

## 2.7.8. cublas<t>syrkx()

```
cublasStatus_t cublasSsyrkx(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const float           *alpha,
                            const float           *A, int lda,
                            const float           *B, int ldb,
                            const float           *beta,
                            float           *C, int ldc)
cublasStatus_t cublasDsyrkx(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const double          *alpha,
                            const double          *A, int lda,
                            const double          *B, int ldb,
                            const double          *beta,
                            double          *C, int ldc)
cublasStatus_t cublasCsyrkx(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const cuComplex       *alpha,
                            const cuComplex       *A, int lda,
                            const cuComplex       *B, int ldb,
                            const cuComplex       *beta,
                            cuComplex       *C, int ldc)
cublasStatus_t cublasZsyrkx(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const cuDoubleComplex *alpha,
                            const cuDoubleComplex *A, int lda,
                            const cuDoubleComplex *B, int ldb,
                            const cuDoubleComplex *beta,
                            cuDoubleComplex *C, int ldc)
```

This function performs a variation of the symmetric rank- $k$ update

$$C = \alpha(\mathrm{op}(A)\mathrm{op}(B)^T + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a symmetric matrix stored in lower or upper mode, and $A$ and $B$ are matrices with dimensions $\mathrm{op}(A)$ $n \times k$ and $\mathrm{op}(B)$ $n \times k$, respectively. Also, for matrix $A$ and $B$

$$\mathrm{op}(A) \text{ and } \mathrm{op}(B) = \begin{cases} A \text{ and } B & \text{if trans == CUBLAS\_OP\_N} \\ A^T \text{ and } B^T & \text{if trans == CUBLAS\_OP\_T} \end{cases}$$

This routine can be used when B is in such way that the result is garanteed to be symmetric. An usual example is when the matrix B is a scaled form of the matrix A : this is equivalent to B being the product of the matrix A and a diagonal matrix. For an efficient computation of the product of a regular matrix with a diagonal matrix, refer to the routine cublas<t>dgmm.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix c lower or upper part, is stored, the other symmetric part is not referenced and is inferred from the stored elements. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| trans | | input | operation op(**A**) that is non- or transpose. |
| n | | input | number of rows of matrix op(**A**), op(**B**) and **C**. |
| k | | input | number of columns of matrix op(**A**) and op(**B**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **transa == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | device | input | <type> array of dimensions **ldb x k** with **ldb>=max(1,n)** if **transa == CUBLAS_OP_N** and **ldb x n** with **ldb>=max(1,k)** otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta==0**, then **C** does not have to be a valid input. |
| C | device | in/out | <type> array of dimensions **ldc x n** with **ldc>=max(1,n)**. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

ssyrk, dsyrk, csyrk, zsyrk and

ssyr2k, dsyr2k, csyr2k, zsyr2k

## 2.7.9. cublas<t>trmm()

```
cublasStatus_t cublasStrmm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int m, int n,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *B, int ldb,
                           float                 *C, int ldc)
cublasStatus_t cublasDtrmm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int m, int n,
                           const double          *alpha,
                           const double          *A, int lda,
                           const double          *B, int ldb,
                           double                *C, int ldc)
cublasStatus_t cublasCtrmm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int m, int n,
                           const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *B, int ldb,
                           cuComplex             *C, int ldc)
cublasStatus_t cublasZtrmm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int m, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *B, int ldb,
                           cuDoubleComplex       *C, int ldc)
```

This function performs the triangular matrix-matrix multiplication

$$C = \begin{cases} \alpha \mathrm{op}(A) B & \text{if side == CUBLAS\_SIDE\_LEFT} \\ \alpha B \mathrm{op}(A) & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A$ is a triangular matrix stored in lower or upper mode with or without the main diagonal, $B$ and $C$ are $m \times n$ matrix, and $\alpha$ is a scalar. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

Notice that in order to achieve better parallelism cuBLAS differs from the BLAS API only for this routine. The BLAS API assumes an in-place implementation (with results written back to B), while the cuBLAS API assumes an out-of-place implementation (with results written into C). The application can obtain the in-place functionality of BLAS in the cuBLAS API by passing the address of the matrix B in place of the matrix C. No other overlapping in the input parameters is supported.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| side | | input | indicates if matrix **A** is on the left or right of **B**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix **A** are unity and should not be accessed. |
| m | | input | number of rows of matrix **B**, with matrix **A** sized accordingly. |
| n | | input | number of columns of matrix **B**, with matrix **A** sized accordingly. |
| alpha | host or device | input | <type> scalar used for multiplication, if **alpha==0** then **A** is not referenced and **B** does not have to be a valid input. |
| A | device | input | <type> array of dimension **lda x m** with **lda>=max(1,m)** if **side == CUBLAS_SIDE_LEFT** and **lda x n** with **lda>=max(1,n)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | device | input | <type> array of dimension **ldb x n** with **ldb>=max(1,m).** |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| C | device | in/out | <type> array of dimension **ldc x n** with **ldc>=max(1,m).** |
| ldc | | input | leading dimension of two-dimensional array used to store matrix C. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

strmm, dtrmm, ctrmm, ztrmm

## 2.7.10. cublas<t>trsm()

```
cublasStatus_t cublasStrsm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int m, int n,
                           const float           *alpha,
                           const float           *A, int lda,
                           float           *B, int ldb)
cublasStatus_t cublasDtrsm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int m, int n,
                           const double          *alpha,
                           const double          *A, int lda,
                           double          *B, int ldb)
cublasStatus_t cublasCtrsm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int m, int n,
                           const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           cuComplex       *B, int ldb)
cublasStatus_t cublasZtrsm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           cublasOperation_t trans, cublasDiagType_t diag,
                           int m, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           cuDoubleComplex *B, int ldb)
```

This function solves the triangular linear system with multiple right-hand-sides

$$\begin{cases} \mathrm{op}(A)X = \alpha B & \text{if side == CUBLAS\_SIDE\_LEFT} \\ X\mathrm{op}(A) = \alpha B & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A$ is a triangular matrix stored in lower or upper mode with or without the main diagonal, $X$ and $B$ are $m \times n$ matrices, and $\alpha$ is a scalar. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

The solution $X$ overwrites the right-hand-sides $B$ on exit.

No test for singularity or near-singularity is included in this function.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| side | | input | indicates if matrix **A** is on the left or right of **X**. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix **A** are unity and should not be accessed. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| m | | input | number of rows of matrix **B**, with matrix **A** sized accordingly. |
| n | | input | number of columns of matrix **B**, with matrix **A** is sized accordingly. |
| alpha | host or device | input | <type> scalar used for multiplication, if **alpha==0** then **A** is not referenced and **B** does not have to be a valid input. |
| A | device | input | <type> array of dimension **lda x m** with **lda>=max(1,m)** if **side == CUBLAS_SIDE_LEFT** and **lda x n** with **lda>=max(1,n)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | device | in/out | <type> array. It has dimensions **ldb x n** with **ldb>=max(1,m)**. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

strsm, dtrsm, ctrsm, ztrsm

## 2.7.11. cublas<t>trsmBatched()

```
cublasStatus_t cublasStrsmBatched( cublasHandle_t    handle,
                                   cublasSideMode_t  side,
                                   cublasFillMode_t  uplo,
                                   cublasOperation_t trans,
                                   cublasDiagType_t  diag,
                                   int m,
                                   int n,
                                   const float *alpha,
                                   float *A[],
                                   int lda,
                                   float *B[],
                                   int ldb,
                                   int batchCount);
cublasStatus_t cublasDtrsmBatched( cublasHandle_t    handle,
                                   cublasSideMode_t  side,
                                   cublasFillMode_t  uplo,
                                   cublasOperation_t trans,
                                   cublasDiagType_t  diag,
                                   int m,
                                   int n,
                                   const double *alpha,
                                   double *A[],
                                   int lda,
                                   double *B[],
                                   int ldb,
                                   int batchCount);
cublasStatus_t cublasCtrsmBatched( cublasHandle_t    handle,
                                   cublasSideMode_t  side,
                                   cublasFillMode_t  uplo,
                                   cublasOperation_t trans,
                                   cublasDiagType_t  diag,
                                   int m,
                                   int n,
                                   const cuComplex *alpha,
                                   cuComplex *A[],
                                   int lda,
                                   cuComplex *B[],
                                   int ldb,
                                   int batchCount);
cublasStatus_t cublasZtrsmBatched( cublasHandle_t    handle,
                                   cublasSideMode_t  side,
                                   cublasFillMode_t  uplo,
                                   cublasOperation_t trans,
                                   cublasDiagType_t  diag,
                                   int m,
                                   int n,
                                   const cuDoubleComplex *alpha,
                                   cuDoubleComplex *A[],
                                   int lda,
                                   cuDoubleComplex *B[],
                                   int ldb,
                                   int batchCount);
```

This function solves an array of triangular linear systems with multiple right-hand-sides

$$\begin{cases} \mathrm{op}(A[i])X[i] = \alpha B[i] & \text{if side == CUBLAS\_SIDE\_LEFT} \\ X[i]\mathrm{op}(A[i]) = \alpha B[i] & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A[i]$ is a triangular matrix stored in lower or upper mode with or without the main diagonal, $X[i]$ and $B[i]$ are $m \times n$ matrices, and $\alpha$ is a scalar. Also, for matrix $A$

$$\text{op}(A[i]) = \begin{cases} A[i] & \text{if transa == CUBLAS\_OP\_N} \\ A^T[i] & \text{if transa == CUBLAS\_OP\_T} \\ A^H[i] & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

The solution $X[i]$ overwrites the right-hand-sides $B[i]$ on exit.

No test for singularity or near-singularity is included in this function.

This function works for any sizes but is intended to be used for matrices of small sizes where the launch overhead is a significant factor. For bigger sizes, it might be advantageous to call **batchCount** times the regular **cublas<t>trsm** within a set of CUDA streams.

The current implementation is limited to devices with compute capability above or equal 2.0.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| side | | input | indicates if matrix **A[i]** is on the left or right of **X[i]**. |
| uplo | | input | indicates if matrix **A[i]** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A[i]**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix **A[i]** are unity and should not be accessed. |
| m | | input | number of rows of matrix **B[i]**, with matrix **A[i]** sized accordingly. |
| n | | input | number of columns of matrix **B[i]**, with matrix **A[i]** is sized accordingly. |
| alpha | host or device | input | <type> scalar used for multiplication, if **alpha==0** then **A[i]** is not referenced and **B[i]** does not have to be a valid input. |
| A | device | input | array of pointers to <type> array, with each array of dim. **lda x m** with **lda>=max(1,m)** if **transa==CUBLAS_OP_N** and **lda x n** with **lda>=max(1,n)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A[i]**. |
| B | device | in/out | array of pointers to <type> array, with each array of dim. **ldb x n** with **ldb>=max(1,m)** |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B[i]**. |
| batchCount | | input | number of pointers contained in A and B. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n<0**. |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device is below compute capability 2.0. |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

strsm, dtrsm, ctrsm, ztrsm

## 2.7.12. cublas<t>hemm()

```
cublasStatus_t cublasChemm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           int m, int n,
                           const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *B, int ldb,
                           const cuComplex       *beta,
                           cuComplex       *C, int ldc)
cublasStatus_t cublasZhemm(cublasHandle_t handle,
                           cublasSideMode_t side, cublasFillMode_t uplo,
                           int m, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *B, int ldb,
                           const cuDoubleComplex *beta,
                           cuDoubleComplex *C, int ldc)
```

This function performs the Hermitian matrix-matrix multiplication

$$C = \begin{cases} \alpha AB + \beta C & \text{if side == CUBLAS\_SIDE\_LEFT} \\ \alpha BA + \beta C & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A$ is a Hermitian matrix stored in lower or upper mode, $B$ and $C$ are $m \times n$ matrices, and $\alpha$ and $\beta$ are scalars.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| side | | input | indicates if matrix **A** is on the left or right of **B**. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| m | | input | number of rows of matrix **C** and **B**, with matrix **A** sized accordingly. |
| n | | input | number of columns of matrix **C** and **B**, with matrix **A** sized accordingly. |
| alpha | host or device | input | <type> scalar used for multiplication. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| A | device | input | <type> array of dimension `lda x m` with `lda>=max(1,m)` if `side==CUBLAS_SIDE_LEFT` and `lda x n` with `lda>=max(1,n)` otherwise. The imaginary parts of the diagonal elements are assumed to be zero. |
| lda | | input | leading dimension of two-dimensional array used to store matrix `A`. |
| B | device | input | <type> array of dimension `ldb x n` with `ldb>=max(1,m)`. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix `B`. |
| beta | | input | <type> scalar used for multiplication, if `beta==0` then `C` does not have to be a valid input. |
| C | device | in/out | <type> array of dimensions `ldc x n` with `ldc>=max(1,m)`. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix `C`. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `m,n<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

chemm, zhemm

## 2.7.13. cublas<t>herk()

```
cublasStatus_t cublasCherk(cublasHandle_t handle,
                           cublasFillMode_t uplo, cublasOperation_t trans,
                           int n, int k,
                           const float  *alpha,
                           const cuComplex       *A, int lda,
                           const float  *beta,
                           cuComplex       *C, int ldc)
cublasStatus_t cublasZherk(cublasHandle_t handle,
                           cublasFillMode_t uplo, cublasOperation_t trans,
                           int n, int k,
                           const double *alpha,
                           const cuDoubleComplex *A, int lda,
                           const double *beta,
                           cuDoubleComplex *C, int ldc)
```

This function performs the Hermitian rank- $k$ update

$$C = \alpha \mathrm{op}(A) \mathrm{op}(A)^H + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a Hermitian matrix stored in lower or upper mode, and $A$ is a matrix with dimensions $\text{op}(A)$ $n \times k$. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| n | | input | number of rows of matrix op(**A**) and **C**. |
| k | | input | number of columns of matrix op(**A**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **transa == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| beta | | input | <type> scalar used for multiplication, if **beta==0** then **C** does not have to be a valid input. |
| C | device | in/out | <type> array of dimension **ldc x n**, with **ldc>=max(1,n)**. The imaginary parts of the diagonal elements are assumed and set to zero. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

cherk, zherk

## 2.7.14. cublas<t>her2k()

```
cublasStatus_t cublasCher2k(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const cuComplex       *alpha,
                            const cuComplex       *A, int lda,
                            const cuComplex       *B, int ldb,
                            const float  *beta,
                            cuComplex       *C, int ldc)
cublasStatus_t cublasZher2k(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const cuDoubleComplex *alpha,
                            const cuDoubleComplex *A, int lda,
                            const cuDoubleComplex *B, int ldb,
                            const double *beta,
                            cuDoubleComplex *C, int ldc)
```

This function performs the Hermitian rank- $2k$ update

$$C = \alpha \text{op}(A)\text{op}(B)^H + \bar{\alpha}\text{op}(B)\text{op}(A)^H + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a Hermitian matrix stored in lower or upper mode, and $A$ and $B$ are matrices with dimensions $\text{op}(A)$ $n \times k$ and $\text{op}(B)$ $n \times k$, respectively. Also, for matrix $A$ and $B$

$$\text{op}(A) \text{ and } \text{op}(B) = \begin{cases} A \text{ and } B & \text{if trans == CUBLAS\_OP\_N} \\ A^H \text{ and } B^H & \text{if trans == CUBLAS\_OP\_C} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| n | | input | number of rows of matrix op(**A**), op(**B**) and **C**. |
| k | | input | number of columns of matrix op(**A**) and op(**B**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **transa == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | device | input | <type> array of dimension **ldb x k** with **ldb>=max(1,n)** if **transa == CUBLAS_OP_N** and **ldb x n** with **ldb>=max(1,k)** otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| beta | host or device | input | <type> scalar used for multiplication, if `beta==0` then C does not have to be a valid input. |
| C | device | in/out | <type> array of dimension `ldc x n`, with `ldc>=max(1,n)`. The imaginary parts of the diagonal elements are assumed and set to zero. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix C. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `n,k<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

cher2k, zher2k

## 2.7.15. cublas<t>herkx()

```
cublasStatus_t cublasCherkx(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const cuComplex       *alpha,
                            const cuComplex       *A, int lda,
                            const cuComplex       *B, int ldb,
                            const float  *beta,
                            cuComplex       *C, int ldc)
cublasStatus_t cublasZherkx(cublasHandle_t handle,
                            cublasFillMode_t uplo, cublasOperation_t trans,
                            int n, int k,
                            const cuDoubleComplex *alpha,
                            const cuDoubleComplex *A, int lda,
                            const cuDoubleComplex *B, int ldb,
                            const double *beta,
                            cuDoubleComplex *C, int ldc)
```

This function performs a variation of the Hermitian rank- $k$ update

$$C = \alpha \text{op}(A) \text{op}(B)^H + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a Hermitian matrix stored in lower or upper mode, and $A$ and $B$ are matrices with dimensions $\text{op}(A)$ $n \times k$ and $\text{op}(B)$ $n \times k$, respectively. Also, for matrix $A$ and $B$

$$\text{op}(A) \text{ and } \text{op}(B) = \begin{cases} A \text{ and } B & \text{if trans == CUBLAS\_OP\_N} \\ A^H \text{ and } B^H & \text{if trans == CUBLAS\_OP\_C} \end{cases}$$

This routine can be used when the matrix B is in such way that the result is garanteed to be hermitian. An usual example is when the matrix B is a scaled form of the matrix A : this is equivalent to B being the product of the matrix A and a diagonal matrix. For an efficient computation of the product of a regular matrix with a diagonal matrix, refer to the routine cublas<t>dgmm.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| n | | input | number of rows of matrix op(**A**), op(**B**) and **C**. |
| k | | input | number of columns of matrix op(**A**) and op(**B**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **transa == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | device | input | <type> array of dimension **ldb x k** with **ldb>=max(1,n)** if **transa == CUBLAS_OP_N** and **ldb x n** with **ldb>=max(1,k)** otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host or device | input | real scalar used for multiplication, if **beta==0** then **C** does not have to be a valid input. |
| C | device | in/out | <type> array of dimension **ldc x n**, with **ldc>=max(1,n)**. The imaginary parts of the diagonal elements are assumed and set to zero. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

cherk, zherk and

cher2k, zher2k

# 2.8. BLAS-like Extension

In this chapter we describe the BLAS-extension functions that perform matrix-matrix operations.

## 2.8.1. cublas<t>geam()

```
cublasStatus_t cublasSgeam(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *beta,
                           const float           *B, int ldb,
                           float           *C, int ldc)
cublasStatus_t cublasDgeam(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n,
                           const double          *alpha,
                           const double          *A, int lda,
                           const double          *beta,
                           const double          *B, int ldb,
                           double          *C, int ldc)
cublasStatus_t cublasCgeam(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n,
                           const cuComplex       *alpha,
                           const cuComplex       *A, int lda,
                           const cuComplex       *beta ,
                           const cuComplex       *B, int ldb,
                           cuComplex       *C, int ldc)
cublasStatus_t cublasZgeam(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n,
                           const cuDoubleComplex *alpha,
                           const cuDoubleComplex *A, int lda,
                           const cuDoubleComplex *beta,
                           const cuDoubleComplex *B, int ldb,
                           cuDoubleComplex *C, int ldc)
```

This function performs the matrix-matrix addition/transposition

$$C = \alpha \mathrm{op}(A) + \beta \mathrm{op}(B)$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are matrices stored in column-major format with dimensions $\mathrm{op}(A)$ $m \times n$, $\mathrm{op}(B)$ $m \times n$ and $C$ $m \times n$, respectively. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\mathrm{op}(B)$ is defined similarly for matrix $B$.

The operation is out-of-place if C does not overlap A or B.

The in-place mode supports the following two operations,

$C = \alpha^{*}C + \beta \mathrm{op}(B)$

$C = \alpha \mathrm{op}(A) + \beta^{*}C$

For in-place mode, if **C = A**, **ldc = lda** and **transa = CUBLAS_OP_N**. If **C = B**, **ldc = ldb** and **transb = CUBLAS_OP_N**. If the user does not meet above requirements, **CUBLAS_STATUS_INVALID_VALUE** is returned.

The operation includes the following special cases:

the user can reset matrix C to zero by setting ***alpha=*beta=0**.

the user can transpose matrix A by setting ***alpha=1 and *beta=0**.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| transa | | input | operation op(**A**) that is non- or (conj.) transpose. |
| transb | | input | operation op(**B**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(**A**) and **C**. |
| n | | input | number of columns of matrix op(**B**) and **C**. |
| alpha | host or device | input | <type> scalar used for multiplication. If ***alpha == 0**, **A** does not have to be a valid input. |
| A | device | input | <type> array of dimensions **lda x n** with **lda>=max(1,m)** if **transa == CUBLAS_OP_N** and **lda x m** with **lda>=max(1,n)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store the matrix **A**. |
| B | device | input | <type> array of dimension **ldb x n** with **ldb>=max(1,m)** if **transa == CUBLAS_OP_N** and **ldb x m** with **ldb>=max(1,n)** otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host or device | input | <type> scalar used for multiplication. If ***beta == 0**, **B** does not have to be a valid input. |
| C | device | output | <type> array of dimensions **ldc x n** with **ldc>=max(1,m)**. |
| ldc | | input | leading dimension of a two-dimensional array used to store the matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n<0**, **alpha,beta=NULL** or improper settings of in-place mode |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

## 2.8.2. cublas<t>dgmm()

```
cublasStatust cublasSdgmm(cublasHandle_t handle, cublasSideMode_t mode,
                          int m, int n,
                          const float           *A, int lda,
                          const float           *x, int incx,
                          float           *C, int ldc)
cublasStatus_t cublasDdgmm(cublasHandle_t handle, cublasSideMode_t mode,
                          int m, int n,
                          const double          *A, int lda,
                          const double          *x, int incx,
                          double          *C, int ldc)
cublasStatus_t cublasCdgmm(cublasHandle_t handle, cublasSideMode_t mode,
                          int m, int n,
                          const cuComplex       *A, int lda,
                          const cuComplex       *x, int incx,
                          cuComplex       *C, int ldc)
cublasStatus_t cublasZdgmm(cublasHandle_t handle, cublasSideMode_t mode,
                          int m, int n,
                          const cuDoubleComplex *A, int lda,
                          const cuDoubleComplex *x, int incx,
                          cuDoubleComplex *C, int ldc)
```

This function performs the matrix-matrix multiplication

$$C = \begin{cases} A \times diag(X) & \text{if mode == CUBLAS\_SIDE\_RIGHT} \\ diag(X) \times A & \text{if mode == CUBLAS\_SIDE\_LEFT} \end{cases}$$

where $A$ and $C$ are matrices stored in column-major format with dimensions $m \times n$ . $X$ is a vector of size $n$ if **mode == CUBLAS_SIDE_RIGHT** and of size $m$ if **mode == CUBLAS_SIDE_LEFT**. $X$ is gathered from one-dimensional array x with stride **incx**. The absolute value of **incx** is the stride and the sign of **incx** is direction of the stride. If **incx** is positive, then we forward x from the first element. Otherwise, we backward x from the last element. The formula of X is

$$X[j] = \begin{cases} x[j \times incx] & \text{if } incx \geq 0 \\ x[(\chi - 1) \times |incx| - j \times |incx|] & \text{if } incx < 0 \end{cases}$$

where $\chi = m$ if **mode == CUBLAS_SIDE_LEFT** and $\chi = n$ if **mode == CUBLAS_SIDE_RIGHT**.

Example 1: if the user wants to perform $diag(diag(B)) \times A$ , then $incx = ldb + 1$ where $ldb$ is leading dimension of matrix **B**, either row-major or column-major.

Example 2: if the user wants to perform $\alpha \times A$ , then there are two choices, either cublasgeam with ***beta=0** and **transa == CUBLAS_OP_N** or cublasdgmm with **incx=0** and **x[0]=alpha**.

The operation is out-of-place. The in-place only works if **lda = ldc**.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| mode | | input | left multiply if **mode == CUBLAS_SIDE_LEFT** or right multiply if **mode == CUBLAS_SIDE_RIGHT** |
| m | | input | number of rows of matrix **A** and **C**. |
| n | | input | number of columns of matrix **A** and **C**. |
| A | device | input | <type> array of dimensions **lda x n** with **lda>=max(1,m)** |
| lda | | input | leading dimension of two-dimensional array used to store the matrix **A**. |
| x | device | input | one-dimensional <type> array of size $\|inc\| \times m$ if **mode == CUBLAS_SIDE_LEFT** and $\|inc\| \times n$ if **mode == CUBLAS_SIDE_RIGHT** |
| incx | | input | stride of one-dimensional array **x**. |
| C | device | in/out | <type> array of dimensions **ldc x n** with **ldc>=max(1,m)**. |
| ldc | | input | leading dimension of a two-dimensional array used to store the matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n<0** or **mode != CUBLAS_SIDE_LEFT, CUBLAS_SIDE_RIGHT** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

## 2.8.3. cublas<t>getrfBatched()

```
cublasStatus_t cublasSgetrfBatched(cublasHandle_t handle,
                                   int n,
                                   float *Aarray[],
                                   int lda,
                                   int *PivotArray,
                                   int *infoArray,
                                   int batchSize);

cublasStatus_t cublasDgetrfBatched(cublasHandle_t handle,
                                   int n,
                                   double *Aarray[],
                                   int lda,
                                   int *PivotArray,
                                   int *infoArray,
                                   int batchSize);

cublasStatus_t cublasCgetrfBatched(cublasHandle_t handle,
                                   int n,
                                   cuComplex *Aarray[],
                                   int lda,
                                   int *PivotArray,
                                   int *infoArray,
                                   int batchSize);

cublasStatus_t cublasZgetrfBatched(cublasHandle_t handle,
                                   int n,
                                   cuDoubleComplex *Aarray[],
                                   int lda,
                                   int *PivotArray,
                                   int *infoArray,
                                   int batchSize);
```

**Aarray** is an array of pointers to matrices stored in column-major format with dimensions **nxn** and leading dimension **lda**.

This function performs the LU factorization of each **Aarray[i]** for i = 0, ..., **batchSize-1** by the following equation

$$P*\text{Aarray}[i] = L*U$$

where **P** is a permutation matrix which represents partial pivoting with row interchanges. **L** is a lower triangular matrix with unit diagonal and **U** is an upper triangular matrix.

Formally **P** is written by a product of permutation matrices **Pj**, for **j = 1,2,...,n**, say **P = P1 * P2 * P3 * .... * Pn**. **Pj** is a permutation matrix which interchanges two rows of vector x when performing **Pj*x**. **Pj** can be constructed by **j** element of **PivotArray[i]** by the following matlab code

```
// In Matlab PivotArray[i] is an array of base-1.
// In C, PivotArray[i] is base-0.
Pj = eye(n);
swap Pj(j,:) and Pj(PivotArray[i][j]  ,:)
```

**L** and **U** are written back to original matrix **A**, and diagonal elements of **L** are discarded. The **L** and **U** can be constructed by the following matlab code

```
// A is a matrix of nxn after getrf.
L = eye(n);
for j = 1:n
    L(:,j+1:n) = A(:,j+1:n)
end
U = zeros(n);
for i = 1:n
    U(i,i:n) = A(i,i:n)
end
```

If matrix **A(=Aarray[i])** is singular, getrf still works and the value of **info(=infoArray[i])** reports first row index that LU factorization cannot proceed. If info is **k**, **U(k,k)** is zero. The equation **P*A=L*U** still holds, however **L** and **U** are from the following matlab code

```
// A is a matrix of nxn after getrf.
// info is k, which means U(k,k) is zero.
L = eye(n);
for j = 1:k-1
    L(:,j+1:n) = A(:,j+1:n)
end
U = zeros(n);
for i = 1:k-1
    U(i,i:n) = A(i,i:n)
end
for i = k:n
    U(i,k:n) = A(i,k:n)
end
```

This function is intended to be used for matrices of small sizes where the launch overhead is a significant factor.

cublas<t>getrfBatched supports non-pivot LU factorization if **PivotArray** is nil.

cublas<t>getrfBatched supports arbitrary dimension.

cublas<t>getrfBatched only supports compute capability 2.0 or above.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of rows and columns of **Aarray[i]**. |
| Aarray | device | input | array of pointers to <type> array, with each array of dim. **n x n** with **lda>=max(1,n)**. |
| lda | | input | leading dimension of two-dimensional array used to store each matrix **Aarray[i]**. |
| PivotArray | device | output | array of size **n x batchSize** that contains the pivoting sequence of each factorization of **Aarray[i]** stored in a linear fashion. If **PivotArray** is nil, pivoting is disabled. |
| infoArray | device | output | array of size **batchSize** that info(=infoArray[i]) contains the information of factorization of **Aarray[i]**.<br>If info=0, the execution is successful.<br>If info = -j, the j-th parameter had an illegal value. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| | | | If info = k, U(k,k) is 0. The factorization has been completed, but U is exactly singular. |
| batchSize | | input | number of pointers contained in A |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,batchSize,lda <0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device has a compute capability < 200 |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sgeqrf, dgeqrf, cgeqrf, zgeqrf

## 2.8.4. cublas<t>getrsBatched()

```
cublasStatus_t cublasSgetrsBatched(cublasHandle_t handle,
                                   cublasOperation_t trans,
                                   int n,
                                   int nrhs,
                                   const float *Aarray[],
                                   int lda,
                                   const int *devIpiv,
                                   float *Barray[],
                                   int ldb,
                                   int *info,
                                   int batchSize);

cublasStatus_t cublasDgetrsBatched(cublasHandle_t handle,
                                   cublasOperation_t trans,
                                   int n,
                                   int nrhs,
                                   const double *Aarray[],
                                   int lda,
                                   const int *devIpiv,
                                   double *Barray[],
                                   int ldb,
                                   int *info,
                                   int batchSize);

cublasStatus_t cublasCgetrsBatched(cublasHandle_t handle,
                                   cublasOperation_t trans,
                                   int n,
                                   int nrhs,
                                   const cuComplex *Aarray[],
                                   int lda,
                                   const int *devIpiv,
                                   cuComplex *Barray[],
                                   int ldb,
                                   int *info,
                                   int batchSize);

cublasStatus_t cublasZgetrsBatched(cublasHandle_t handle,
                                   cublasOperation_t trans,
                                   int n,
                                   int nrhs,
                                   const cuDoubleComplex *Aarray[],
                                   int lda,
                                   const int *devIpiv,
                                   cuDoubleComplex *Barray[],
                                   int ldb,
                                   int *info,
                                   int batchSize);
```

This function solves an array of systems of linear equations of the form :

$\text{op}(A[i])X[i] = \alpha B[i]$

where $A[i]$ is a matrix which has been LU factorized with pivoting , $X[i]$ and $B[i]$ are $n \times \text{nrhs}$ matrices. Also, for matrix $A$

$$\text{op}(A[i]) = \begin{cases} A[i] & \text{if trans == CUBLAS\_OP\_N} \\ A^T[i] & \text{if trans == CUBLAS\_OP\_T} \\ A^H[i] & \text{if trans == CUBLAS\_OP\_C} \end{cases}$$

This function is intended to be used for matrices of small sizes where the launch overhead is a significant factor.

cublas<t>getrsBatched supports non-pivot LU factorization if **devIpiv** is nil.

cublas<t>getrsBatched supports arbitrary dimension.

cublas<t>getrsBatched only supports compute capability 2.0 or above.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| n | | input | number of rows and columns of **Aarray[i]**. |
| nrhs | | input | number of columns of **Barray[i]**. |
| Aarray | device | input | array of pointers to <type> array, with each array of dim. **n x n** with **lda>=max(1,n)**. |
| lda | | input | leading dimension of two-dimensional array used to store each matrix **Aarray[i]**. |
| devIpiv | device | input | array of size **n x batchSize** that contains the pivoting sequence of each factorization of **Aarray[i]** stored in a linear fashion. If **devIpiv** is nil, pivoting for all **Aarray[i]** is ignored. |
| Barray | device | input/ output | array of pointers to <type> array, with each array of dim. **n x nrhs** with **ldb>=max(1,n)**. |
| ldb | | input | leading dimension of two-dimensional array used to store each solution matrix **Barray[i]**. |
| info | host | output | If info=0, the execution is successful.<br>If info = -j, the j-th parameter had an illegal value. |
| batchSize | | input | number of pointers contained in A |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,batchSize,lda <0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device has a compute capability < 200 |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sgeqrs, dgeqrs, cgeqrs, zgeqrs

## 2.8.5. cublas<t>getriBatched()

```
cublasStatus_t cublasSgetriBatched(cublasHandle_t handle,
                                   int n,
                                   float *Aarray[],
                                   int lda,
                                   int *PivotArray,
                                   float *Carray[],
                                   int ldc,
                                   int *infoArray,
                                   int batchSize);

cublasStatus_t cublasDgetriBatched(cublasHandle_t handle,
                                   int n,
                                   double *Aarray[],
                                   int lda,
                                   int *PivotArray,
                                   double *Carray[],
                                   int ldc,
                                   int *infoArray,
                                   int batchSize);

cublasStatus_t cublasCgetriBatched(cublasHandle_t handle,
                                   int n,
                                   cuComplex *Aarray[],
                                   int lda,
                                   int *PivotArray,
                                   cuComplex *Carray[],
                                   int ldc,
                                   int *infoArray,
                                   int batchSize);

cublasStatus_t cublasZgetriBatched(cublasHandle_t handle,
                                   int n,
                                   cuDoubleComplex *Aarray[],
                                   int lda,
                                   int *PivotArray,
                                   cuDoubleComplex *Carray[],
                                   int ldc,
                                   int *infoArray,
                                   int batchSize);
```

**Aarray** and **Carray** are arrays of pointers to matrices stored in column-major format with dimensions **n*n** and leading dimension **lda** and **ldc** respectively.

This function performs the inversion of matrices **A[i]** for i = 0, ..., **batchSize-1**.

Prior to calling cublas<t>getriBatched, the matrix **A[i]** must be factorized first using the routine cublas<t>getrfBatched. After the call of cublas<t>getrfBatched, the matrix pointing by **Aarray[i]** will contain the LU factors of the matrix **A[i]** and the vector pointing by **(PivotArray+i)** will contain the pivoting sequence.

Following the LU factorization, cublas<t>getriBatched uses forward and backward triangular solvers to complete inversion of matrices **A[i]** for i = 0, ..., **batchSize-1**. The inversion is out-of-place, so memory space of Carray[i] cannot overlap memory space of Array[i].

Typically all parameters in cublas<t>getrfBatched would be passed into cublas<t>getriBatched. For example,

```
// step 1: perform in-place LU decomposition, P*A = L*U.
//      Aarray[i] is n*n matrix A[i]
    cublasDgetrfBatched(handle, n, Aarray, lda, PivotArray, infoArray,
 batchSize);
//      check infoArray[i] to see if factorization of A[i] is successful or not.
//      Array[i] contains LU factorization of A[i]

// step 2: perform out-of-place inversion, Carray[i] = inv(A[i])
    cublasDgetriBatched(handle, n, Aarray, lda, PivotArray, Carray, ldc,
 infoArray, batchSize);
//      check infoArray[i] to see if inversion of A[i] is successful or not.
```

The user can check singularity from either cublas<t>getrfBatched or cublas<t>getriBatched.

This function is intended to be used for matrices of small sizes where the launch overhead is a significant factor.

If cublas<t>getrfBatched is performed by non-pivoting, **`PivotArray`** of cublas<t>getriBatched should be nil.

cublas<t>getriBatched supports arbitrary dimension.

cublas<t>getriBatched only supports compute capability 2.0 or above.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of rows and columns of **`Aarray[i]`**. |
| Aarray | device | input | array of pointers to <type> array, with each array of dimension **`n*n`** with **`lda>=max(1,n)`**. |
| lda | | input | leading dimension of two-dimensional array used to store each matrix **`Aarray[i]`**. |
| PivotArray | device | output | array of size **`n*batchSize`** that contains the pivoting sequence of each factorization of **`Aarray[i]`** stored in a linear fashion. If **`PivotArray`** is nil, pivoting is disabled. |
| Carray | device | output | array of pointers to <type> array, with each array of dimension **`n*n`** with **`ldc>=max(1,n)`**. |
| ldc | | input | leading dimension of two-dimensional array used to store each matrix **`Carray[i]`**. |
| infoArray | device | output | array of size **`batchSize`** that info(=infoArray[i]) contains the information of inversion of **`A[i]`**.<br>If info=0, the execution is successful.<br>If info = k, U(k,k) is 0. The U is exactly singular and the inversion failed. |
| batchSize | | input | number of pointers contained in A |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,batchSize,lda,ldc <0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device has a compute capability < 200 |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

## 2.8.6. cublas<t>matinvBatched()

```
cublasStatus_t cublasSmatinvBatched(cublasHandle_t handle,
                                    int n,
                                    const float *A[],
                                    int lda,
                                    float *Ainv[],
                                    int lda_inv,
                                    int *info,
                                    int batchSize);

cublasStatus_t cublasDmatinvBatched(cublasHandle_t handle,
                                    int n,
                                    const double *A[],
                                    int lda,
                                    double *Ainv[],
                                    int lda_inv,
                                    int *info,
                                    int batchSize);

cublasStatus_t cublasCmatinvBatched(cublasHandle_t handle,
                                    int n,
                                    const cuComplex *A[],
                                    int lda,
                                    cuComplex *Ainv[],
                                    int lda_inv,
                                    int *info,
                                    int batchSize);

cublasStatus_t cublasZmatinvBatched(cublasHandle_t handle,
                                    int n,
                                    const cuDoubleComplex *A[],
                                    int lda,
                                    cuDoubleComplex *Ainv[],
                                    int lda_inv,
                                    int *info,
                                    int batchSize);
```

**A** and **Ainv** are arrays of pointers to matrices stored in column-major format with dimensions **n*n** and leading dimension **lda** and **lda_inv** respectively.

This function performs the inversion of matrices **A[i]** for i = 0, ..., **batchSize-1**.

This function is a short cut of **cublas<t>getrfBatched** plus **cublas<t>getriBatched**. However it only works if **n** is less than 32. If not, the user has to go through **cublas<t>getrfBatched** and **cublas<t>getriBatched**.

If the matrix **A[i]** is singular, then **info[i]** reports singularity, the same as **cublas<t>getrfBatched**.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of rows and columns of **A[i]**. |
| A | device | input | array of pointers to <type> array, with each array of dimension **n*n** with **lda>=max(1,n)**. |
| lda | | input | leading dimension of two-dimensional array used to store each matrix **A[i]**. |
| Ainv | device | output | array of pointers to <type> array, with each array of dimension **n*n** with **lda_inv>=max(1,n)**. |
| lda_inv | | input | leading dimension of two-dimensional array used to store each matrix **Ainv[i]**. |
| info | device | output | array of size **batchSize** that info[i] contains the information of inversion of **A[i]**.<br>If info[i]=0, the execution is successful.<br>If info[i]=k, U(k,k) is 0. The U is exactly singular and the inversion failed. |
| batchSize | | input | number of pointers contained in A. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,batchSize,lda,lda_inv <0; or n >32** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device has a compute capability < 200 |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

## 2.8.7. cublas<t>geqrfBatched()

```
cublasStatus_t cublasSgeqrfBatched( cublasHandle_t handle,
                                    int m,
                                    int n,
                                    float *Aarray[],
                                    int lda,
                                    float *TauArray[],

                                    int *info,
                                    int batchSize);

cublasStatus_t cublasDgeqrfBatched( cublasHandle_t handle,
                                    int m,
                                    int n,
                                    double *Aarray[],
                                    int lda,
                                    double *TauArray[],

                                    int *info,
                                    int batchSize);

cublasStatus_t cublasCgeqrfBatched( cublasHandle_t handle,
                                    int m,
                                    int n,
                                    cuComplex *Aarray[],
                                    int lda,
                                    cuComplex *TauArray[],

                                    int *info,
                                    int batchSize);

cublasStatus_t cublasZgeqrfBatched( cublasHandle_t handle,
                                    int m,
                                    int n,
                                    cuDoubleComplex *Aarray[],
                                    int lda,
                                    cuDoubleComplex *TauArray[],

                                    int *info,
                                    int batchSize);
```

**Aarray** is an array of pointers to matrices stored in column-major format with dimensions **m x n** and leading dimension **lda**. **TauArray** is an array of pointers to vectors of dimension of at least **max (1, min(m, n)**.

This function performs the QR factorization of each **Aarray[i]** for **i = 0, ...,batchSize-1** using Householder reflections. Each matrix **Q[i]** is represented as a product of elementary reflectors and is stored in the lower part of each **Aarray[i]** as follows :

```
     Q[j] = H[j][1] H[j][2] . . . H[j](k), where k = min(m,n).

     Each H[j][i] has the form

          H[j][i] = I - tau[j] * v * v'
          where tau[j] is a real scalar, and v is a real vector with
          v(1:i-1) = 0 and v(i) = 1; v(i+1:m) is stored on exit in Aarray[j][i
+1:m,i],
          and tau in TauArray[j][i]
```

This function is intended to be used for matrices of small sizes where the launch overhead is a significant factor.

cublas<t>geqrfBatched supports arbitrary dimension.

cublas<t>geqrfBatched only supports compute capability 2.0 or above.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| m | | input | number of rows **`Aarray[i]`**. |
| n | | input | number of columns of **`Aarray[i]`**. |
| Aarray | device | input | array of pointers to <type> array, with each array of dim. **`m x n`** with **`lda>=max(1,m)`**. |
| lda | | input | leading dimension of two-dimensional array used to store each matrix **`Aarray[i]`**. |
| TauArray | device | output | array of pointers to <type> vector, with each vector of dim. **`max(1,min(m,n))`**. |
| info | host | output | If info=0, the parameters passed to the function are valid<br>If info<0, the parameter in postion -info is invalid |
| batchSize | | input | number of pointers contained in A |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **`CUBLAS_STATUS_SUCCESS`** | the operation completed successfully |
| **`CUBLAS_STATUS_NOT_INITIALIZED`** | the library was not initialized |
| **`CUBLAS_STATUS_INVALID_VALUE`** | the parameters **`m,n,batchSize <0 or lda < imax(1,m)`** |
| **`CUBLAS_STATUS_ARCH_MISMATCH`** | the device has a compute capability < 200 |
| **`CUBLAS_STATUS_EXECUTION_FAILED`** | the function failed to launch on the GPU |

For references please refer to:

sgeqrf, dgeqrf, cgeqrf, zgeqrf

## 2.8.8. cublas<t>gelsBatched()

```
cublasStatus_t cublasSgelsBatched( cublasHandle_t handle,
                                   cublasOperation_t trans,
                                   int m,
                                   int n,
                                   int nrhs,
                                   float *Aarray[],
                                   int lda,
                                   float *Carray[],
                                   int ldc,

                                   int *info,
                                   int *devInfoArray,
                                   int batchSize );

cublasStatus_t cublasDgelsBatched( cublasHandle_t handle,
                                   cublasOperation_t trans,
                                   int m,
                                   int n,
                                   int nrhs,
                                   double *Aarray[],
                                   int lda,
                                   double *Carray[],
                                   int ldc,

                                   int *info,
                                   int *devInfoArray,
                                   int batchSize );

cublasStatus_t cublasCgelsBatched( cublasHandle_t handle,
                                   cublasOperation_t trans,
                                   int m,
                                   int n,
                                   int nrhs,
                                   cuComplex *Aarray[],
                                   int lda,
                                   cuComplex *Carray[],
                                   int ldc,

                                   int *info,
                                   int *devInfoArray,
                                   int batchSize );

cublasStatus_t cublasZgelsBatched( cublasHandle_t handle,
                                   cublasOperation_t trans,
                                   int m,
                                   int n,
                                   int nrhs,
                                   cuDoubleComplex *Aarray[],
                                   int lda,
                                   cuDoubleComplex *Carray[],
                                   int ldc,

                                   int *info,
                                   int *devInfoArray,
                                   int batchSize );
```

**`Aarray`** is an array of pointers to matrices stored in column-major format with dimensions **`m x n`** and leading dimension **`lda`**. **`Carray`** is an array of pointers to matrices stored in column-major format with dimensions **`n x nrhs`** and leading dimension **`ldc`**.

This function find the least squares solution of a batch of overdetermined systems : it solves the least squares problem described as follows :

```
          minimize  || Carray[i] - Aarray[i]*Xarray[i] || , with i =
0, ...,batchSize-1
```

On exit, each **`Aarray[i]`** is overwritten with their QR factorization and each **`Carray[i]`** is overwritten with the least square solution

cublas<t>gelsBatched supports only the non-transpose operation and only solves over-determined systems (m >= n).

cublas<t>gelsBatched only supports compute capability 2.0 or above.

This function is intended to be used for matrices of small sizes where the launch overhead is a significant factor.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| trans | | input | operation op(**`Aarray[i]`**) that is non- or (conj.) transpose. Only non-transpose operation is currently supported. |
| m | | input | number of rows **`Aarray[i]`**. |
| n | | input | number of columns of each **`Aarray[i]`** and rows of each **`Carray[i]`**. |
| nrhs | | input | number of columns of each **`Carray[i]`**. |
| Aarray | device | input/ output | array of pointers to <type> array, with each array of dim. **`m x n`** with **`lda>=max(1,m)`**. |
| lda | | input | leading dimension of two-dimensional array used to store each matrix **`Aarray[i]`**. |
| Carray | device | input/ output | array of pointers to <type> array, with each array of dim. **`n x nrhs`** with **`ldc>=max(1,m)`**. |
| ldc | | input | leading dimension of two-dimensional array used to store each matrix **`Carray[i]`**. |
| info | host | output | If info=0, the parameters passed to the function are valid<br>If info<0, the parameter in position -info is invalid |
| devInfoArray | device | output | optional array of integers of dimension batchsize.<br>If non-null, every element devInfoArray[i] contain a value V with the following meaning:<br>V = 0 : the i-th problem was sucessfully solved<br>V > 0 : the V-th diagonal element of the Aarray[i] is zero. Aarray[i] does not have full rank. |
| batchSize | | input | number of pointers contained in Aarray and Carray |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n,batchSize <0 , lda < imax(1,m)** or **ldc < imax(1,m)** |
| **CUBLAS_STATUS_NOT_SUPPORTED** | the parameters **m <n** or **trans** is different from non-transpose. |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device has a compute capability < 200 |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sgels, dgels, cgels, zgels

## 2.8.9. cublas<t>tpttr()

```
cublasStatus_t cublasStpttr ( cublasHandle_t handle,
                              cublasFillMode_t uplo,
                              int n,
                              const float *AP,
                              float *A,
                              int lda );

cublasStatus_t cublasDtpttr ( cublasHandle_t handle,
                              cublasFillMode_t uplo,
                              int n,
                              const double *AP,
                              double *A,
                              int lda );

cublasStatus_t cublasCtpttr ( cublasHandle_t handle,
                              cublasFillMode_t uplo,
                              int n,
                              const cuComplex *AP,
                              cuComplex *A,
                              int lda );

cublasStatus_t cublasZtpttr ( cublasHandle_t handle,
                              cublasFillMode_t uplo
                              int n,
                              const cuDoubleComplex *AP,
                              cuDoubleComplex *A,
                              int lda );
```

This function performs the conversion from the triangular packed format to the triangular format

If **uplo == CUBLAS_FILL_MODE_LOWER** then the elements of **AP** are copied into the lower triangular part of the triangular matrix **A** and the upper part of **A** is left untouched. If **uplo == CUBLAS_FILL_MODE_UPPER** then the elements of **AP** are copied into the upper triangular part of the triangular matrix **A** and the lower part of **A** is left untouched.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| uplo | | input | indicates if matrix **AP** contains lower or upper part of matrix **A**. |
| n | | input | number of rows and columns of matrix **A**. |
| AP | device | input | <type> array with $A$ stored in packed format. |
| A | device | output | <type> array of dimensions **lda x n** , with **lda>=max(1,n)**. The opposite side of A is left untouched. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

stpttr, dtpttr, ctpttr, ztpttr

## 2.8.10. cublas<t>trttp()

```
cublasStatus_t cublasStrttp ( cublasHandle_t handle,
                              cublasFillMode_t uplo,
                              int n,
                              const float *A,
                              int lda,
                              float *AP );

cublasStatus_t cublasDtrttp ( cublasHandle_t handle,
                              cublasFillMode_t uplo,
                              int n,
                              const double *A,
                              int lda,
                              double *AP );

cublasStatus_t cublasCtrttp ( cublasHandle_t handle,
                              cublasFillMode_t uplo,
                              int n,
                              const cuComplex *A,
                              int lda,
                              cuComplex *AP );

cublasStatus_t cublasZtrttp ( cublasHandle_t handle,
                              cublasFillMode_t uplo,
                              int n,
                              const cuDoubleComplex *A,
                              int lda,
                              cuDoubleComplex *AP );
```

This function performs the conversion from the triangular format to the triangular packed format

If **uplo == CUBLAS_FILL_MODE_LOWER** then the lower triangular part of the triangular matrix **A** is copied into the array **AP**. If **uplo == CUBLAS_FILL_MODE_UPPER** then then the upper triangular part of the triangular matrix **A** is copied into the array **AP**.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates which matrix **A** lower or upper part is referenced. |
| n | | input | number of rows and columns of matrix **A**. |
| A | device | input | <type> array of dimensions **lda x n** , with **lda>=max(1,n)**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| AP | device | output | <type> array with $A$ stored in packed format. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

strttp, dtrttp, ctrttp, ztrttp

## 2.8.11. cublas<t>gemmEx()

```
cublasStatus_t cublasSgemmEx(cublasHandle_t handle,
                             cublasOperation_t transa,
                             cublasOperation_t transb,
                             int m,
                             int n,
                             int k,
                             const float    *alpha,
                             const void     *A,
                             cudaDataType_t Atype,
                             int lda,
                             const void     *B,
                             cudaDataType_t Btype,
                             int ldb,
                             const float    *beta,
                             void           *C,
                             cudaDataType_t Ctype,
                             int ldc)
cublasStatus_t cublasCgemmEx(cublasHandle_t handle,
                             cublasOperation_t transa,
                             cublasOperation_t transb,
                             int m,
                             int n,
                             int k,
                             const cuComplex *alpha,
                             const void      *A,
                             cudaDataType_t  Atype,
                             int lda,
                             const void      *B,
                             cudaDataType_t  Btype,
                             int ldb,
                             const cuComplex *beta,
                             void            *C,
                             cudaDataType_t  Ctype,
                             int ldc)
```

This function is an extension of **cublas<t>gemm** where the input matrices and output matrices can have a lower precision but the computation is still done in the type **<t>**, e.g in **float** for **cublasSgemmEx** and **cuComplex** for **cublasSgemmEx**

$$C = \alpha \mathrm{op}(A)\mathrm{op}(B) + \beta C$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are matrices stored in column-major format with dimensions $\mathrm{op}(A)$ $m \times k$, $\mathrm{op}(B)$ $k \times n$ and $C$ $m \times n$, respectively. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\mathrm{op}(B)$ is defined similarly for matrix $B$.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| transa | | input | operation op(**A**) that is non- or (conj.) transpose. |
| transb | | input | operation op(**B**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(**A**) and C. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| n | | input | number of columns of matrix op(B) and C. |
| k | | input | number of columns of op(A) and rows of op(B). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimensions `lda x k` with `lda>=max(1,m)` if `transa == CUBLAS_OP_N` and `lda x m` with `lda>=max(1,k)` otherwise. |
| Atype | | input | enumerant specifying the datatype of matrix A. |
| lda | | input | leading dimension of two-dimensional array used to store the matrix A. |
| B | device | input | <type> array of dimension `ldb x n` with `ldb>=max(1,k)` if `transa == CUBLAS_OP_N` and `ldb x k` with `ldb>=max(1,n)` otherwise. |
| Btype | | input | enumerant specifying the datatype of matrix B. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix B. |
| beta | host or device | input | <type> scalar used for multiplication. If `beta==0`, C does not have to be a valid input. |
| C | device | in/out | <type> array of dimensions `ldc x n` with `ldc>=max(1,m)`. |
| Ctype | | input | enumerant specifying the datatype of matrix C. |
| ldc | | input | leading dimension of a two-dimensional array used to store the matrix C. |

The matrix types combinations supported for **`cublasSgemmEx`** are listed below :

| A | B | C |
|---|---|---|
| `CUDA_R_16F` | `CUDA_R_16F` | `CUDA_R_16F` |
| `CUDA_R_16F` | `CUDA_R_16F` | `CUDA_R_32F` |
| `CUDA_R_8I` | `CUDA_R_8I` | `CUDA_R_32F` |
| `CUDA_R_32F` | `CUDA_R_32F` | `CUDA_R_32F` |

The matrix types combinations supported for **`cublasCgemmEx`** are listed below :

| A | B | C |
|---|---|---|
| `CUDA_C_8I` | `CUDA_C_8I` | `CUDA_C_32F` |
| `CUDA_C_32F` | `CUDA_C_32F` | `CUDA_C_32F` |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_ARCH_MISMATCH** | **cublasCgemmEx** is only supported for GPU with architecture capabilities equal or greater than 5.0 |
| **CUBLAS_STATUS_NOT_SUPPORTED** | the combination of the parameters **Atype,Btype** and **Ctype** is not supported |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n,k<0** |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

sgemm

## 2.8.12. cublasGemmEx()

```
cublasStatus_t cublasGemmEx(cublasHandle_t handle,
                            cublasOperation_t transa,
                            cublasOperation_t transb,
                            int m,
                            int n,
                            int k,
                            const void    *alpha,
                            const void     *A,
                            cudaDataType_t Atype,
                            int lda,
                            const void     *B,
                            cudaDataType_t Btype,
                            int ldb,
                            const void    *beta,
                            void           *C,
                            cudaDataType_t Ctype,
                            int ldc,
                            cudaDataType_t computeType,
                            cublasGemmAlgo_t algo)
```

This function is an extension of **cublas<t>gemm** that allows the user to individually specify the data types for each of the A, B and C matrices, the precision of computation and the GEMM algorithm to be run. Supported combinations of arguments are listed further down in this section.

$$C = \alpha \text{op}(A)\text{op}(B) + \beta C$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are matrices stored in column-major format with dimensions $\text{op}(A)$ $m \times k$, $\text{op}(B)$ $k \times n$ and $C$ $m \times n$, respectively. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\text{op}(B)$ is defined similarly for matrix $B$.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| transa | | input | operation op(**A**) that is non- or (conj.) transpose. |
| transb | | input | operation op(**B**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(**A**) and **C**. |
| n | | input | number of columns of matrix op(**B**) and **C**. |
| k | | input | number of columns of op(**A**) and rows of op(**B**). |
| alpha | host or device | input | scaling factor for A*B; of same type as computeType. |
| A | device | input | <type> array of dimensions **lda x k** with **lda>=max(1,m)** if **transa == CUBLAS_OP_N** and **lda x m** with **lda>=max(1,k)** otherwise. |
| Atype | | input | enumerant specifying the datatype of matrix **A**. |
| lda | | input | leading dimension of two-dimensional array used to store the matrix **A**. |
| B | device | input | <type> array of dimension **ldb x n** with **ldb>=max(1,k)** if **transa == CUBLAS_OP_N** and **ldb x k** with **ldb>=max(1,n)** otherwise. |
| Btype | | input | enumerant specifying the datatype of matrix **B**. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host or device | input | scaling factor for C; of same type as computeType. If **beta==0**, **C** does not have to be a valid input. |
| C | device | in/out | <type> array of dimensions **ldc x n** with **ldc>=max(1,m)**. |
| Ctype | | input | enumerant specifying the datatype of matrix **C**. |
| ldc | | input | leading dimension of a two-dimensional array used to store the matrix **C**. |
| computeType | | input | enumerant specifying the computation type. |
| algo | | input | enumerant specifying the algorithm. |

**cublasGemmEx** supports the following computeType, Atype/Btype, and Ctype:

| computeType | Atype/Btype | Ctype |
|---|---|---|
| CUDA_R_16F | CUDA_R_16F | CUDA_R_16F |
| CUDA_R_8I | CUDA_R_8I | CUDA_R_32I |
| CUDA_R_32F | CUDA_R_16F | CUDA_R_16F |
| | CUDA_R_8I | CUDA_R_32F |
| | CUDA_R_16F | CUDA_R_32F |

| computeType | Atype/Btype | Ctype |
|---|---|---|
| | CUDA_R_32F | CUDA_R_32F |
| CUDA_R_64F | CUDA_R_64F | CUDA_R_64F |
| CUDA_C_32F | CUDA_C_8I | CUDA_C_32F |
| | CUDA_C_32F | CUDA_C_32F |
| CUDA_C_64F | CUDA_C_64F | CUDA_C_64F |

**cublasGemmEx** routine runs for the following algorithm.

| CublasGemmAlgo_t | Meaning |
|---|---|
| CUBLAS_GEMM_DEFAULT | Apply Heuristics to select the GEMM algorithm |
| CUBLAS_GEMM_ALGO0 to CUBLAS_GEMM_ALGO17 | Explicitly choose an algorithm |
| CUBLAS_GEMM_DEFAULT_TENSOR_OP | Apply Heuristics to select the GEMM algorithm while allowing the use of Tensor Core operations if possible |
| CUBLAS_GEMM_ALGO0_TENSOR_OP to CUBLAS_GEMM_ALGO4_TENSOR_OP | Explicitly choose a GEMM algorithm allowing it to use Tensor Core operations if possible, otherwise falls back to **cublas<t>gemmBatched** based on computeType |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_ARCH_MISMATCH | **cublasGemmEx** is only supported for GPU with architecture capabilities equal or greater than 5.0 |
| CUBLAS_STATUS_NOT_SUPPORTED | the combination of the parameters **Atype**, **Btype** and **Ctype** or the algorithm, **algo** is not supported |
| CUBLAS_STATUS_INVALID_VALUE | the parameters **m,n,k<0** |
| CUBLAS_STATUS_EXECUTION_FAILED | the function failed to launch on the GPU |

NOTE: This function is not supported in Device API.

For references please refer to:

sgemm

## 2.8.13. cublasGemmBatchedEx()

```
cublasStatus_t cublasGemmBatchedEx(cublasHandle_t handle,
                                   cublasOperation_t transa,
                                   cublasOperation_t transb,
                                   int m,
                                   int n,
                                   int k,
                                   const void    *alpha,
                                   const void     *Aarray[],
                                   cudaDataType_t Atype,
                                   int lda,
                                   const void     *Barray,
                                   cudaDataType_t Btype,
                                   int ldb,
                                   const void    *beta,
                                   void           *Carray,
                                   cudaDataType_t Ctype,
                                   int ldc,
                                   int batchCount,
                                   cudaDataType_t computeType,
                                   cublasGemmAlgo_t algo)
```

This function is an extension of **cublas<t>gemmBatched** that performs the matrix-matrix multiplication of a batch of matrices and allows the user to individually specify the data types for each of the A, B and C matrix arrays, the precision of computation and the GEMM algorithm to be run. Like **cublas<t>gemmBatched**, the batch is considered to be "uniform", i.e. all instances have the same dimensions (m, n, k), leading dimensions (lda, ldb, ldc) and transpositions (transa, transb) for their respective A, B and C matrices. The address of the input matrices and the output matrix of each instance of the batch are read from arrays of pointers passed to the function by the caller. Supported combinations of arguments are listed further down in this section.

$$C[i] = \alpha \mathrm{op}(A[i]) \mathrm{op}(B[i]) + \beta C[i], \text{ for i } \in [0, batchCount - 1]$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are arrays of pointers to matrices stored in column-major format with dimensions $\mathrm{op}(A[i])\ m \times k$, $\mathrm{op}(B[i])\ k \times n$ and $C[i]\ m \times n$, respectively. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\mathrm{op}(B[i])$ is defined similarly for matrix $B[i]$.

On certain problem sizes, it might be advantageous to make multiple calls to **cublas<t>gemm** in different CUDA streams, rather than use this API.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| transa | | input | operation op(**A[i]**) that is non- or (conj.) transpose. |
| transb | | input | operation op(**B[i]**) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(**A[i]**) and **C[i]**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| n | | input | number of columns of matrix op(`B[i]`) and `C[i]`. |
| k | | input | number of columns of op(`A[i]`) and rows of op(`B[i]`). |
| alpha | host or device | input | scalar used for multiplication; of same type as computeType. |
| Aarray | device | input | array of pointers to <Atype> array, with each array of dim. `lda x k` with `lda>=max(1,m)` if `transa == CUBLAS_OP_N` and `lda x m` with `lda>=max(1,k)` otherwise. |
| Atype | | input | enumerant specifying the datatype of `Aarray`. |
| lda | | input | leading dimension of two-dimensional array used to store the matrix `A[i]`. |
| Barray | device | input | array of pointers to <Btype> array, with each array of dim. `ldb x n` with `ldb>=max(1,k)` if `transa == CUBLAS_OP_N` and `ldb x k` with `ldb>=max(1,n)` otherwise. |
| Btype | | input | enumerant specifying the datatype of `Barray`. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix `B[i]`. |
| beta | host or device | input | scalar used for multiplication; of same type as computeType. If `beta==0`, `C[i]` does not have to be a valid input. |
| Carray | device | in/out | array of pointers to <Ctype> array. It has dimensions `ldc x n` with `ldc>=max(1,m)`. |
| Ctype | | input | enumerant specifying the datatype of `Carray`. |
| ldc | | input | leading dimension of a two-dimensional array used to store each matrix `C[i]`. |
| batchCount | | input | number of pointers contained in Aarray, Barray and Carray. |
| computeType | | input | enumerant specifying the computation type. |
| algo | | input | enumerant specifying the algorithm. |

**`cublasGemmBatchedEx`** supports the following computeType, Atype/Btype, and Ctype:

| Compute type | A/B | C |
|---|---|---|
| `CUDA_R_16F` | `CUDA_R_16F` | `CUDA_R_16F` |
| `CUDA_R_32F` | `CUDA_R_16F` | `CUDA_R_16F` |
| | `CUDA_R_16F` | `CUDA_R_32F` |
| | `CUDA_R_8I` | `CUDA_R_32F` |
| | `CUDA_R_32F` | `CUDA_R_32F` |

| Compute type | A/B | C |
|---|---|---|
| CUDA_R_64F | CUDA_R_64F | CUDA_R_64F |
| CUDA_C_32F | CUDA_C_8I | CUDA_C_32F |
| | CUDA_C_32F | CUDA_C_32F |
| CUDA_C_64F | CUDA_C_64F | CUDA_C_64F |

**cublasGemmBatchedEx** routine is run for the following algorithm.

| CublasGemmAlgo_t | Meaning |
|---|---|
| CUBLAS_GEMM_DEFAULT | Apply Heuristics to select the GEMM algorithm |
| CUBLAS_GEMM_ALGO0 to CUBLAS_GEMM_ALGO17 | Explicitly choose an algorithm |
| CUBLAS_GEMM_DEFAULT_TENSOR_OP | Apply Heuristics to select the GEMM algorithm while allowing the use of Tensor Core operations if possible |
| CUBLAS_GEMM_ALGO0_TENSOR_OP to CUBLAS_GEMM_ALGO4_TENSOR_OP | Explicitly choose a GEMM algorithm allowing it to use Tensor Core operations if possible, otherwise falls back to **cublas<t>gemmBatched** based on computeType |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_ARCH_MISMATCH | **cublasGemmBatchedEx** is only supported for GPU with architecture capabilities equal or greater than 5.0 |
| CUBLAS_STATUS_NOT_SUPPORTED | the combination of the parameters **Atype**, **Btype** and **Ctype** or the algorithm, **algo** is not supported |
| CUBLAS_STATUS_INVALID_VALUE | the parameters **m,n,k<0** |
| CUBLAS_STATUS_EXECUTION_FAILED | the function failed to launch on the GPU |

NOTE: This function is not supported in Device API.

For references please refer to:

sgemm

## 2.8.14. cublasGemmStridedBatchedEx()

```
cublasStatus_t cublasGemmStridedBatchedEx(cublasHandle_t handle,
                          cublasOperation_t transa,
                          cublasOperation_t transb,
                          int m,
                          int n,
                          int k,
                          const void    *alpha,
                          const void     *A,
                          cudaDataType_t Atype,
                          int lda,
                          long long int strideA,
                          const void     *B,
                          cudaDataType_t Btype,
                          int ldb,
                          long long int strideB,
                          const void    *beta,
                          void           *C,
                          cudaDataType_t Ctype,
                          int ldc,
                          long long int strideC,
                          int batchCount,
                          cudaDataType_t computeType,
                          cublasGemmAlgo_t algo)
```

This function is an extension of **cublas<t>gemmStridedBatched** that performs the matrix-matrix multiplication of a batch of matrices and allows the user to individually specify the data types for each of the A, B and C matrices, the precision of computation and the GEMM algorithm to be run. Like **cublas<t>gemmStridedBatched**, the batch is considered to be "uniform", i.e. all instances have the same dimensions (m, n, k), leading dimensions (lda, ldb, ldc) and transpositions (transa, transb) for their respective A, B and C matrices. Input matrices A, B and output matrix C for each instance of the batch are located at fixed address offsets from their locations in the previous instance. Pointers to A, B and C matrices for the first instance are passed to the function by the user along with the address offsets - strideA, strideB and strideC that determine the locations of input and output matrices in future instances.

$$C + i*\text{strideC} = \alpha \text{op}(A + i*\text{strideA}) \text{op}(B + i*\text{strideB}) + \beta(C + i*\text{strideC}), \text{ for i } \in [0, batchCount - 1]$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are arrays of pointers to matrices stored in column-major format with dimensions $\text{op}(A[i])$ $m \times k$, $\text{op}(B[i])$ $k \times n$ and $C[i]$ $m \times n$, respectively. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\text{op}(B[i])$ is defined similarly for matrix $B[i]$.

On certain problem sizes, it might be advantageous to make multiple calls to **cublas<t>gemm** in different CUDA streams, rather than use this API.

Note: In the table below, we use **A[i], B[i], C[i]** as notation for A, B and C matrices in the ith instance of the batch, implicitly assuming they are respectively address offsets **strideA, strideB, strideC** away from **A[i-1], B[i-1], C[i-1]** .

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| transa | | input | operation op(`A[i]`) that is non- or (conj.) transpose. |
| transb | | input | operation op(`B[i]`) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(`A[i]`) and `C[i]`. |
| n | | input | number of columns of matrix op(`B[i]`) and `C[i]`. |
| k | | input | number of columns of op(`A[i]`) and rows of op(`B[i]`). |
| alpha | host or device | input | scalar used for multiplication; of same type as computeType. |
| A | device | input | pointer to <Atype> matrix, A, corresponds to the first instance of the batch, with dimensions `lda x k` with `lda>=max(1,m)` if `transa == CUBLAS_OP_N` and `lda x m` with `lda>=max(1,k)` otherwise. |
| Atype | | input | enumerant specifying the datatype of `A`. |
| lda | | input | leading dimension of two-dimensional array used to store the matrix `A[i]`. |
| strideA | | input | value of type long long int that gives the address offset between `A[i]` and `A[i+1]`. |
| B | device | input | pointer to <Btype> matrix, B, corresponds to the first instance of the batch, with dimensions `ldb x n` with `ldb>=max(1,k)` if `transa == CUBLAS_OP_N` and `ldb x k` with `ldb>=max(1,n)` otherwise. |
| Btype | | input | enumerant specifying the datatype of `B`. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix `B[i]`. |
| strideB | | input | value of type long long int that gives the address offset between `B[i]` and `B[i+1]`. |
| beta | host or device | input | scalar used for multiplication; of same type as computeType. If `beta==0`, `C[i]` does not have to be a valid input. |
| Carray | device | in/out | pointer to <Ctype> matrix, C, corresponds to the first instance of the batch, with dimensions `ldc x n` with `ldc>=max(1,m)`. |
| Ctype | | input | enumerant specifying the datatype of `C`. |
| ldc | | input | leading dimension of a two-dimensional array used to store each matrix `C[i]`. |
| strideC | | input | value of type long long int that gives the address offset between `C[i]` and `C[i+1]`. |
| batchCount | | input | number of GEMMs to perform in the batch. |
| computeType | | input | enumerant specifying the computation type. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| algo | | input | enumerant specifying the algorithm. |

**cublasGemmStridedBatchedEx** supports the following computeType, Atype/Btype, and Ctype:

| Compute type | A/B | C |
|---|---|---|
| CUDA_R_16F | CUDA_R_16F | CUDA_R_16F |
| CUDA_R_32F | CUDA_R_16F | CUDA_R_16F |
| | CUDA_R_16F | CUDA_R_32F |
| | CUDA_R_8I | CUDA_R_32F |
| | CUDA_R_32F | CUDA_R_32F |
| CUDA_R_64F | CUDA_R_64F | CUDA_R_64F |
| CUDA_C_32F | CUDA_C_8I | CUDA_C_32F |
| | CUDA_C_32F | CUDA_C_32F |
| CUDA_C_64F | CUDA_C_64F | CUDA_C_64F |

**cublasGemmStridedBatchedEx** routine is run for the following algorithm.

| CublasGemmAlgo_t | Meaning |
|---|---|
| CUBLAS_GEMM_DEFAULT | Apply Heuristics to select the GEMM algorithm |
| CUBLAS_GEMM_ALGO0 to CUBLAS_GEMM_ALGO17 | Explicitly choose an algorithm |
| CUBLAS_GEMM_DEFAULT_TENSOR_OP | Apply Heuristics to select the GEMM algorithm while allowing the use of Tensor Core operations if possible |
| CUBLAS_GEMM_ALGO0_TENSOR_OP to CUBLAS_GEMM_ALGO4_TENSOR_OP | Explicitly choose a GEMM algorithm allowing it to use Tensor Core operations if possible, otherwise falls back to **cublas<t>gemmStridedBatched** |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_ARCH_MISMATCH | **cublasGemmBatchedEx** is only supported for GPU with architecture capabilities equal or greater than 5.0 |
| CUBLAS_STATUS_NOT_SUPPORTED | the combination of the parameters **Atype**, **Btype** and **Ctype** or the algorithm, **algo** is not supported |
| CUBLAS_STATUS_INVALID_VALUE | the parameters **m,n,k<0** |

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

NOTE: This function is not supported in Device API.

For references please refer to:

sgemm

## 2.8.15. cublasCsyrkEx()

```
cublasStatus_t cublasCsyrkEx(cublasHandle_t handle,
                             cublasFillMode_t uplo,
                             cublasOperation_t trans,
                             int n,
                             int k,
                             const float     *alpha,
                             const void      *A,
                             cudaDataType    Atype,
                             int lda,
                             const float    *beta,
                             cuComplex      *C,
                             cudaDataType   Ctype,
                             int ldc)
```

This function is an extension of **cublasCsyrk** where the input matrix and output matrix can have a lower precision but the computation is still done in the type **cuComplex**

This function performs the symmetric rank- $k$ update

$$C = \alpha \mathrm{op}(A) \mathrm{op}(A)^T + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a symmetric matrix stored in lower or upper mode, and $A$ is a matrix with dimensions $\mathrm{op}(A)$ $n \times k$ . Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \end{cases}$$

This routine is only supported on GPUs with architecture capabilities equal or greater than 5.0

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix C lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or transpose. |
| n | | input | number of rows of matrix op(**A**) and C. |
| k | | input | number of columns of matrix op(**A**). |
| alpha | host or device | input | <type> scalar used for multiplication. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **trans == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| Atype | | input | enumerant specifying the datatype of matrix **A**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix A. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta==0** then **C** does not have to be a valid input. |
| C | device | in/out | <type> array of dimension **ldc x n**, with **ldc>=max(1,n)**. |
| Ctype | | input | enumerant specifying the datatype of matrix **C**. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The matrix types combinations supported for **cublasCsyrkEx** are listed below :

| A | C |
|---|---|
| **CUDA_C_8I** | **CUDA_C_32F** |
| **CUDA_C_32F** | **CUDA_C_32F** |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** |
| **CUBLAS_STATUS_NOT_SUPPORTED** | the combination of the parameters **Atype** and **Ctype** is not supported |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device has a compute capabilites lower than 5.0 |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

ssyrk, dsyrk, csyrk, zsyrk

## 2.8.16. cublasCsyrk3mEx()

```
cublasStatus_t cublasCsyrk3mEx(cublasHandle_t handle,
                               cublasFillMode_t uplo,
                               cublasOperation_t trans,
                               int n,
                               int k,
                               const float     *alpha,
                               const void      *A,
                               cudaDataType    Atype,
                               int lda,
                               const float    *beta,
                               cuComplex      *C,
                               cudaDataType   Ctype,
                               int ldc)
```

This function is an extension of **cublasCsyrk** where the input matrix and output matrix can have a lower precision but the computation is still done in the type **cuComplex**. This routine is implemented using the Gauss complexity reduction algorithm which can lead to an increase in performance up to 25%

This function performs the symmetric rank- $k$ update

$$C = \alpha \mathrm{op}(A) \mathrm{op}(A)^T + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a symmetric matrix stored in lower or upper mode, and $A$ is a matrix with dimensions $\mathrm{op}(A)$ $n \times k$ . Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \end{cases}$$

This routine is only supported on GPUs with architecture capabilities equal or greater than 5.0

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix C lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or transpose. |
| n | | input | number of rows of matrix op(**A**) and **C**. |
| k | | input | number of columns of matrix op(**A**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **trans == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| Atype | | input | enumerant specifying the datatype of matrix **A**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| lda | | input | leading dimension of two-dimensional array used to store matrix A. |
| beta | host or device | input | <type> scalar used for multiplication, if **beta==0** then C does not have to be a valid input. |
| C | device | in/out | <type> array of dimension **ldc x n**, with **ldc>=max(1,n)**. |
| Ctype | | input | enumerant specifying the datatype of matrix C. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix C. |

The matrix types combinations supported for **cublasCsyrk3mEx** are listed below :

| A | C |
|---|---|
| CUDA_C_8I | CUDA_C_32F |
| CUDA_C_32F | CUDA_C_32F |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_INVALID_VALUE | the parameters n,k<0 |
| CUBLAS_STATUS_NOT_SUPPORTED | the combination of the parameters Atype and Ctype is not supported |
| CUBLAS_STATUS_ARCH_MISMATCH | the device has a compute capabilites lower than 5.0 |
| CUBLAS_STATUS_EXECUTION_FAILED | the function failed to launch on the GPU |

For references please refer to:

ssyrk, dsyrk, csyrk, zsyrk

## 2.8.17. cublasCherkEx()

```
cublasStatus_t cublasCherkEx(cublasHandle_t handle,
                             cublasFillMode_t uplo,
                             cublasOperation_t trans,
                             int n,
                             int k,
                             const float     *alpha,
                             const void      *A,
                             cudaDataType    Atype,
                             int lda,
                             const float    *beta,
                             cuComplex      *C,
                             cudaDataType   Ctype,
                             int ldc)
```

This function is an extension of **cublasCherk** where the input matrix and output matrix can have a lower precision but the computation is still done in the type **cuComplex**

This function performs the Hermitian rank- $k$ update

$$C = \alpha \mathrm{op}(A) \mathrm{op}(A)^H + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a Hermitian matrix stored in lower or upper mode, and $A$ is a matrix with dimensions $\mathrm{op}(A)$ $n \times k$ . Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

This routine is only supported on GPUs with architecture capabilities equal or greater than 5.0

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| n | | input | number of rows of matrix op(**A**) and **C**. |
| k | | input | number of columns of matrix op(**A**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **transa == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| Atype | | input | enumerant specifying the datatype of matrix **A**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| beta | | input | <type> scalar used for multiplication, if **beta==0** then **C** does not have to be a valid input. |
| C | device | in/out | <type> array of dimension **ldc x n**, with **ldc>=max(1,n)**. The imaginary parts of the diagonal elements are assumed and set to zero. |
| Ctype | | input | enumerant specifying the datatype of matrix **C**. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The matrix types combinations supported for **cublasCherkEx** are listed below :

| A | C |
|---|---|
| CUDA_C_8I | CUDA_C_32F |

| A | C |
|---|---|
| CUDA_C_32F | CUDA_C_32F |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_INVALID_VALUE | the parameters n,k<0 |
| CUBLAS_STATUS_NOT_SUPPORTED | the combination of the parameters Atype and Ctype is not supported |
| CUBLAS_STATUS_ARCH_MISMATCH | the device has a compute capabilites lower than 5.0 |
| CUBLAS_STATUS_EXECUTION_FAILED | the function failed to launch on the GPU |

For references please refer to:

cherk

## 2.8.18. cublasCherk3mEx()

```
cublasStatus_t cublasCherk3mEx(cublasHandle_t handle,
                               cublasFillMode_t uplo,
                               cublasOperation_t trans,
                               int n,
                               int k,
                               const float     *alpha,
                               const void      *A,
                               cudaDataType    Atype,
                               int lda,
                               const float    *beta,
                               cuComplex      *C,
                               cudaDataType   Ctype,
                               int ldc)
```

This function is an extension of **cublasCherk** where the input matrix and output matrix can have a lower precision but the computation is still done in the type **cuComplex**. This routine is implemented using the Gauss complexity reduction algorithm which can lead to an increase in performance up to 25%

This function performs the Hermitian rank- $k$ update

$$C = \alpha \mathrm{op}(A) \mathrm{op}(A)^H + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a Hermitian matrix stored in lower or upper mode, and $A$ is a matrix with dimensions $\mathrm{op}(A)$ $n \times k$ . Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

This routine is only supported on GPUs with architecture capabilities equal or greater than 5.0

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| n | | input | number of rows of matrix op(**A**) and **C**. |
| k | | input | number of columns of matrix op(**A**). |
| alpha | host or device | input | <type> scalar used for multiplication. |
| A | device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **transa == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| Atype | | input | enumerant specifying the datatype of matrix **A**. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| beta | | input | <type> scalar used for multiplication, if **beta==0** then **C** does not have to be a valid input. |
| C | device | in/out | <type> array of dimension **ldc x n**, with **ldc>=max(1,n)**. The imaginary parts of the diagonal elements are assumed and set to zero. |
| Ctype | | input | enumerant specifying the datatype of matrix **C**. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The matrix types combinations supported for **cublasCherk3mEx** are listed below :

| A | C |
|---|---|
| CUDA_C_8I | CUDA_C_32F |
| CUDA_C_32F | CUDA_C_32F |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** |
| **CUBLAS_STATUS_NOT_SUPPORTED** | the combination of the parameters **Atype** and **Ctype** is not supported |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device has a compute capabilites lower than 5.0 |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

cherk

## 2.8.19. cublasNrm2Ex()

```
cublasStatus_t  cublasNrm2Ex( cublasHandle_t handle,
                              int n,
                              const void *x,
                              cudaDataType xType,
                              int incx,
                              void *result,
                              cudaDataType resultType,
                              cudaDataType executionType)
```

This function is an API generalization of the routine **cublas<t>nrm2** where input data, output data and compute type can be specified independently.

This function computes the Euclidean norm of the vector **x**. The code uses a multiphase model of accumulation to avoid intermediate underflow and overflow, with the result being equivalent to $\sqrt{\sum_{i=1}^{n}(\mathbf{x}[j]\times\mathbf{x}[j])}$ where $j = 1+(i-1)*$ incx in exact arithmetic. Notice that the last equation reflects 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vector **x**. |
| x | device | input | <type> vector with **n** elements. |
| xType | | input | enumerant specifying the datatype of vector **x**. |
| incx | | input | stride between consecutive elements of **x**. |
| result | host or device | output | the resulting norm, which is **0.0** if **n,incx<=0**. |
| resultType | | input | enumerant specifying the datatype of the **result**. |
| executionType | | input | enumerant specifying the datatype in which the computation is executed. |

The datatypes combinations currrently supported for **cublasNrm2Ex** are listed below :

| x | result | execution |
|---|---|---|
| CUDA_R_16F | CUDA_R_16F | CUDA_R_32F |
| CUDA_R_32F | CUDA_R_32F | CUDA_R_32F |
| CUDA_R_64F | CUDA_R_64F | CUDA_R_64F |
| CUDA_C_32F | CUDA_C_32F | CUDA_C_32F |
| CUDA_C_64F | CUDA_C_64F | CUDA_C_64F |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_ALLOC_FAILED | the reduction buffer could not be allocated |
| CUBLAS_STATUS_NOT_SUPPORTED | the combination of the parameters **xType**, **resultType** and **executionType** is not supported |
| CUBLAS_STATUS_EXECUTION_FAILED | the function failed to launch on the GPU |

For references please refer to:

snrm2, snrm2, dnrm2, dnrm2, scnrm2, scnrm2, dznrm2

## 2.8.20. cublasAxpyEx()

```
cublasStatus_t cublasAxpyEx (cublasHandle_t handle,
                             int n,
                             const void *alpha,
                             cudaDataType alphaType,
                             const void *x,
                             cudaDataType xType,
                             int incx,
                             void *y,
                             cudaDataType yType,
                             int incy,
                             cudaDataType executiontype);
```

This function is an API generalization of the routine **cublas<t>axpy** where input data, output data and compute type can be specified independently.

This function multiplies the vector **x** by the scalar $\alpha$ and adds it to the vector **y** overwriting the latest vector with the result. Hence, the performed operation is $\mathbf{y}[j] = \alpha \times \mathbf{x}[k] + \mathbf{y}[j]$ for $i = 1, \ldots, n$, $k = 1 + (i-1) * \text{incx}$ and $j = 1 + (i-1) * \text{incy}$. Notice that the last two equations reflect 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| alpha | host or device | input | <type> scalar used for multiplication. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| n | | input | number of elements in the vector **x** and **y**. |
| x | device | input | <type> vector with **n** elements. |
| xType | | input | enumerant specifying the datatype of vector **x**. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | in/out | <type> vector with **n** elements. |
| yType | | input | enumerant specifying the datatype of vector **y**. |
| incy | | input | stride between consecutive elements of **y**. |
| executionTy | | input | enumerant specifying the datatype in which the computation is executed. |

The datatypes combinations currrently supported for **cublasAxpyEx** are listed below :

| x | y | execution |
|---|---|---|
| CUDA_R_16F | CUDA_R_16F | CUDA_R_32F |
| CUDA_R_32F | CUDA_R_32F | CUDA_R_32F |
| CUDA_R_64F | CUDA_R_64F | CUDA_R_64F |
| CUDA_C_32F | CUDA_C_32F | CUDA_C_32F |
| CUDA_C_64F | CUDA_C_64F | CUDA_C_64F |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_NOT_SUPPORTED | the combination of the parameters **xType**,**yType**, and **executionType** is not supported |
| CUBLAS_STATUS_EXECUTION_FAILED | the function failed to launch on the GPU |

For references please refer to:

saxpy, daxpy, caxpy, zaxpy

## 2.8.21. cublasDotEx()

```
cublasStatus_t cublasDotEx (cublasHandle_t handle,
                            int n,
                            const void *x,
                            cudaDataType xType,
                            int incx,
                            const void *y,
                            cudaDataType yType,
                            int incy,
                            void *result,
                            cudaDataType resultType,
                            cudaDataType executionType);

cublasStatus_t cublasDotcEx (cublasHandle_t handle,
                             int n,
                             const void *x,
                             cudaDataType xType,
                             int incx,
                             const void *y,
                             cudaDataType yType,
                             int incy,
                             void *result,
                             cudaDataType resultType,
                             cudaDataType executionType);
```

These functions are an API generalization of the routines **cublas<t>dot** and **cublas<t>dotc** where input data, output data and compute type can be specified independently.

This function computes the dot product of vectors **x** and **y**. Hence, the result is $\sum_{i=1}^{n}(\mathbf{x}[k]\times\mathbf{y}[j])$ where $k=1+(i-1)*$ incx and $j=1+(i-1)*$ incy . Notice that in the first equation the conjugate of the element of vector should be used if the function name ends in character 'c' and that the last two equations reflect 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |
| n | | input | number of elements in the vectors **x** and **y**. |
| x | device | input | <type> vector with **n** elements. |
| xType | | input | enumerant specifying the datatype of vector **x**. |
| incx | | input | stride between consecutive elements of **x**. |
| y | device | input | <type> vector with **n** elements. |
| yType | | input | enumerant specifying the datatype of vector **y**. |
| incy | | input | stride between consecutive elements of **y**. |
| result | host or device | output | the resulting dot product, which is `0.0` if **n<=0**. |
| resultType | | input | enumerant specifying the datatype of the **result**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| executionTy | | input | enumerant specifying the datatype in which the computation is executed. |

The datatypes combinations currrently supported for **cublasDotEx** and **cublasDotcEx** are listed below :

| x | y | result | execution |
|---|---|---|---|
| CUDA_R_16F | CUDA_R_16F | CUDA_R_16F | CUDA_R_32F |
| CUDA_R_32F | CUDA_R_32F | CUDA_R_32F | CUDA_R_32F |
| CUDA_R_64F | CUDA_R_64F | CUDA_R_64F | CUDA_R_64F |
| CUDA_C_32F | CUDA_C_32F | CUDA_C_32F | CUDA_C_32F |
| CUDA_C_64F | CUDA_C_64F | CUDA_C_64F | CUDA_C_64F |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_ALLOC_FAILED | the reduction buffer could not be allocated |
| CUBLAS_STATUS_NOT_SUPPORTED | the combination of the parameters **xType**,**yType**, **resultType** and **executionType** is not supported |
| CUBLAS_STATUS_EXECUTION_FAILED | the function failed to launch on the GPU |

For references please refer to:

sdot, ddot, cdotu, cdotc, zdotu, zdotc

## 2.8.22. cublasScalEx()

```
cublasStatus_t  cublasScalEx(cublasHandle_t handle,
                             int n,
                             const void *alpha,
                             cudaDataType alphaType,
                             void *x,
                             cudaDataType xType,
                             int incx,
                             cudaDataType executionType);
```

This function scales the vector **x** by the scalar $\alpha$ and overwrites it with the result. Hence, the performed operation is $\mathbf{x}[j] = \alpha \times \mathbf{x}[j]$ for $i = 1, \ldots, n$ and $j = 1 + (i-1) * \text{incx}$. Notice that the last two equations reflect 1-based indexing used for compatibility with Fortran.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cuBLAS library context. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| alpha | host or device | input | <type> scalar used for multiplication. |
| n | | input | number of elements in the vector **x**. |
| x | device | in/out | <type> vector with **n** elements. |
| xType | | input | enumerant specifying the datatype of vector **x**. |
| incx | | input | stride between consecutive elements of **x**. |
| executionTy | | input | enumerant specifying the datatype in which the computation is executed. |

The datatypes combinations currrently supported for **cublasScalEx** are listed below :

| x | execution |
|---|---|
| CUDA_R_16F | CUDA_R_32F |
| CUDA_R_32F | CUDA_R_32F |
| CUDA_R_64F | CUDA_R_64F |
| CUDA_C_32F | CUDA_C_32F |
| CUDA_C_64F | CUDA_C_64F |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_NOT_SUPPORTED | the combination of the parameters **xType** and **executionType** is not supported |
| CUBLAS_STATUS_EXECUTION_FAILED | the function failed to launch on the GPU |

For references please refer to:

sscal, dscal, csscal, cscal, zdscal, zscal

# Chapter 3. USING THE CUBLASXT API

## 3.1. General description

The cublasXt API of cuBLAS exposes a multi-GPU capable Host interface : when using this API the application only needs to allocate the required matrices on the Host memory space. There are no restriction on the sizes of the matrices as long as they can fit into the Host memory. The cublasXt API takes care of allocating the memory across the designated GPUs and dispatched the workload between them and finally retrieves the results back to the Host. The cublasXt API supports only the compute-intensive BLAS3 routines (e.g matrix-matrix operations) where the PCI transfers back and forth from the GPU can be amortized. The cublasXt API has its own header file **`cublasXt.h`**.

Starting with release 8.0, cublasXt API allows any of the matrices to be located on a GPU device.

**Note : The cublasXt API is only supported on 64-bit platforms.**

### 3.1.1. Tiling design approach

To be able to share the workload between multiples GPUs, the cublasXt API uses a tiling strategy : every matrix is divided in square tiles of user-controllable dimension BlockDim x BlockDim. The resulting matrix tiling defines the static scheduling policy : each resulting tile is affected to a GPU in a round robin fashion One CPU thread is created per GPU and is responsible to do the proper memory transfers and cuBLAS operations to compute all the tiles that it is responsible for. From a performance point of view, due to this static scheduling strategy, it is better that compute capabilites and PCI bandwidth are the same for every GPU. The figure below illustrates the tiles distribution between 3 GPUs. To compute the first tile G0 from C, the CPU thread 0 responsible of GPU0, have to load 3 tiles from the first row of A and tiles from the first columun of B in a pipeline fashion in order to overlap memory transfer and computations and sum the results into the first tile G0 of C before to move on to the next tile G0.

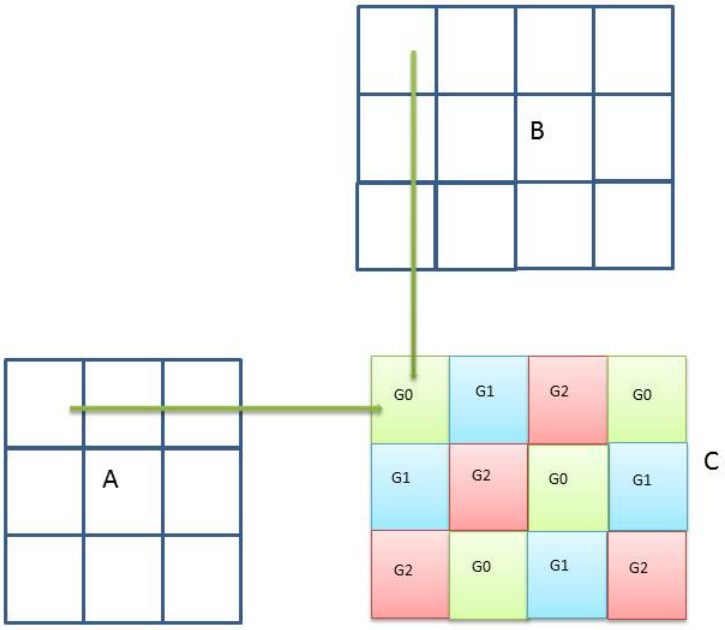


Figure 1 Example of `cublasXt<t>gemm()` tiling for 3 Gpus

When the tile dimension is not an exact multiple of the dimensions of C, some tiles are partially filled on the right border or/and the bottom border. The current implementation does not pad the incomplete tiles but simply keep track of those incomplete tiles by doing the right reduced cuBLAS opearations : this way, no extra computation is done. However it still can lead to some load unbalance when all GPUS do not have the same number of incomplete tiles to work on.

When one or more matrices are located on some GPU devices, the same tiling approach and workload sharing is applied. The memory transfers are in this case done between devices. However, when the computation of a tile and some data are located on the same GPU device, the memory transfer to/from the local data into tiles is bypassed and the GPU operates directly on the local data. This can lead to a significant performance increase, especially when only one GPU is used for the computation.

The matrices can be located on any GPU device, and do not have to be located on the same GPU device. Furthermore, the matrices can even be located on a GPU device that do not participate to the computation.

On the contrary of the cuBLAS API, even if all matrices are located on the same device, the cublasXt API is still a blocking API from the Host point of view : the data results wherever located will be valid on the call return and no device synchronization is required.

### 3.1.2. Hybrid CPU-GPU computation

In the case of very large problems, the cublasXt API offers the possibility to offload some of the computation to the Host CPU. This feature can be setup with the routines **cublasXtSetCpuRoutine()** and **cublasXtSetCpuRatio()** The workload affected to the CPU is put aside : it is simply a percentage of the resulting matrix taken from the bottom and the right side whichever dimension is bigger. The GPU tiling is done after that on the reduced resulting matrix.

If any of the matrices is located on a GPU device, the feature is ignored and all computation will be done only on the GPUs

This feature should be used with caution because it could interfere with the CPU threads responsible of feeding the GPUs.

Currenty, only the routine **cublasXt<t>gemm()** supports this feature.

### 3.1.3. Results reproducibility

Currently all CUBLAS XT API routines from a given toolkit version, generate the same bit-wise results when the following conditions are respected :

- all GPUs particating to the computation have the same compute-capabilities and the same number of SMs.
- the tiles size is kept the same between run.
- either the CPU hybrid computation is not used or the CPU Blas provided is also guaranteed to produce reproducible results.

## 3.2. cublasXt API Datatypes Reference

### 3.2.1. cublasXtHandle_t

The **cublasXtHandle_t** type is a pointer type to an opaque structure holding the cublasXt API context. The cublasXt API context must be initialized using **cublasXtCreate()** and the returned handle must be passed to all subsequent cublasXt API function calls. The context should be destroyed at the end using **cublasXtDestroy()**.

### 3.2.2. cublasXtOpType_t

The **cublasOpType_t** enumerates the four possible types supported by BLAS routines. This enum is used as parameters of the routines **cublasXtSetCpuRoutine** and **cublasXtSetCpuRatio** to setup the hybrid configuration.

| Value | Meaning |
|---|---|
| CUBLASXT_FLOAT | float or single precision type |

| Value | Meaning |
|---|---|
| CUBLASXT_DOUBLE | double precision type |
| CUBLASXT_COMPLEX | single precision complex |
| CUBLASXT_DOUBLECOMPLEX | double precision complex |

## 3.2.3. cublasXtBlasOp_t

The **cublasXtBlasOp_t** type enumerates the BLAS3 or BLAS-like routine supported by cublasXt API. This enum is used as parameters of the routines **cublasXtSetCpuRoutine** and **cublasXtSetCpuRatio** to setup the hybrid configuration.

| Value | Meaning |
|---|---|
| CUBLASXT_GEMM | GEMM routine |
| CUBLASXT_SYRK | SYRK routine |
| CUBLASXT_HERK | HERK routine |
| CUBLASXT_SYMM | SYMM routine |
| CUBLASXT_HEMM | HEMM routine |
| CUBLASXT_TRSM | TRSM routine |
| CUBLASXT_SYR2K | SYR2K routine |
| CUBLASXT_HER2K | HER2K routine |
| CUBLASXT_SPMM | SPMM routine |
| CUBLASXT_SYRKX | SYRKX routine |
| CUBLASXT_HERKX | HERKX routine |

## 3.2.4. cublasXtPinningMemMode_t

The type is used to enable or disable the Pinning Memory mode through the routine **cubasMgSetPinningMemMode**

| Value | Meaning |
|---|---|
| CUBLASXT_PINNING_DISABLED | the Pinning Memory mode is disabled |
| CUBLASXT_PINNING_ENABLED | the Pinning Memory mode is enabled |

# 3.3. cublasXt API Helper Function Reference

## 3.3.1. cublasXtCreate()

```
cublasStatus_t
cublasXtCreate(cublasXtHandle_t *handle)
```

This function initializes the cublasXt API and creates a handle to an opaque structure holding the cublasXt API context. It allocates hardware resources on the host and device and must be called prior to making any other cublasXt API calls.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the initialization succeeded |
| **CUBLAS_STATUS_ALLOC_FAILED** | the resources could not be allocated |
| **CUBLAS_STATUS_NOT_SUPPORTED** | cublasXt API is only supported on 64-bit platform |

## 3.3.2. cublasXtDestroy()

```
cublasStatus_t
cublasXtDestroy(cublasXtHandle_t handle)
```

This function releases hardware resources used by the cublasXt API context. The release of GPU resources may be deferred until the application exits. This function is usually the last call with a particular handle to the cublasXt API.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the shut down succeeded |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

## 3.3.3. cublasXtDeviceSelect()

```
cublasXtDeviceSelect(cublasXtHandle_t handle, int nbDevices, int deviceId[])
```

This function allows the user to provide the number of GPU devices and their respective Ids that will participate to the subsequent cublasXt API Math function calls. This function will create a cuBLAS context for every GPU provided in that list. Currently the device configuration is static and cannot be changed between Math function calls. In that regard, this function should be called only once after cublasXtCreate. To be able to run multiple configurations, multiple cublasXt API contexts should be created.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | User call was sucessful |
| **CUBLAS_STATUS_INVALID_VALUE** | Access to at least one of the device could not be done or a cuBLAS context could not be created on at least one of the device |
| **CUBLAS_STATUS_ALLOC_FAILED** | Some resources could not be allocated. |

## 3.3.4. cublasXtSetBlockDim()

```
cublasXtSetBlockDim(cublasXtHandle_t handle, int blockDim)
```

This function allows the user to set the block dimension used for the tiling of the matrices for the subsequent Math function calls. Matrices are split in square tiles of blockDim x blockDim dimension. This function can be called anytime and will take effect for the following Math function calls. The block dimension should be chosen in a way to optimize the math operation and to make sure that the PCI transfers are well overlapped with the computation.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the call has been successful |
| **CUBLAS_STATUS_INVALID_VALUE** | blockDim <= 0 |

## 3.3.5. cublasXtGetBlockDim()

```
cublasXtGetBlockDim(cublasXtHandle_t handle, int *blockDim)
```

This function allows the user to query the block dimension used for the tiling of the matrices.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the call has been successful |

## 3.3.6. cublasXtSetCpuRoutine()

```
cublasXtSetCpuRoutine(cublasXtHandle_t handle, cublasXtBlasOp_t blasOp,
 cublasXtOpType_t type, void *blasFunctor)
```

This function allows the user to provide a CPU implementation of the corresponding BLAS routine. This function can be used with the function cublasXtSetCpuRatio() to define an hybrid computation between the CPU and the GPUs. Currently the hybrid feature is only supported for the xGEMM routines.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the call has been successful |
| **CUBLAS_STATUS_INVALID_VALUE** | blasOp or type define an invalid combination |
| **CUBLAS_STATUS_NOT_SUPPORTED** | CPU-GPU Hybridization for that routine is not supported |

## 3.3.7. cublasXtSetCpuRatio()

```
cublasXtSetCpuRatio(cublasXtHandle_t handle, cublasXtBlasOp_t blasOp,
 cublasXtOpType_t type, float ratio )
```

This function allows the user to define the percentage of workload that should be done on a CPU in the context of an hybrid computation. This function can be used with the

function cublasXtSetCpuRoutine() to define an hybrid computation between the CPU and the GPUs. Currently the hybrid feature is only supported for the xGEMM routines.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the call has been successful |
| **CUBLAS_STATUS_INVALID_VALUE** | blasOp or type define an invalid combination |
| **CUBLAS_STATUS_NOT_SUPPORTED** | CPU-GPU Hybridization for that routine is not supported |

## 3.3.8. cublasXtSetPinningMemMode()

```
cublasXtSetPinningMemMode(cublasXtHandle_t handle, cublasXtPinningMemMode_t
 mode)
```

This function allows the user to enable or disable the Pinning Memory mode. When enabled, the matrices passed in subsequent cublasXt API calls will be pinned/ unpinned using the CUDART routine **cudaHostRegister** and **cudaHostUnregister** respectively if the matrices are not already pinned. If a matrix happened to be pinned partially, it will also not be pinned. Pinning the memory improve PCI transfer performace and allows to overlap PCI memory transfer with computation. However pinning/unpinning the memory take some time which might not be amortized. It is advised that the user pins the memory on its own using **cudaMallocHost** or **cudaHostRegister** and unpin it when the computation sequence is completed. By default, the Pinning Memory mode is disabled.

The Pinning Memory mode should not enabled when matrices used for different calls to cublasXt API overlap. CublasXt determines that a matrix is pinned or not if the first address of that matrix is pinned using **cudaHostGetFlags**, thus cannot know if the matrix is already partially pinned or not. This is especially true in multi-threaded application where memory could be partially or totally pinned or unpinned while another thread is accessing that memory.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the call has been successful |
| **CUBLAS_STATUS_INVALID_VALUE** | the mode value is different from **CUBLASXT_PINNING_DISABLED** and **CUBLASXT_PINNING_ENABLED** |

## 3.3.9. cublasXtGetPinningMemMode()

```
cublasXtGetPinningMemMode(cublasXtHandle_t handle, cublasXtPinningMemMode_t
 *mode)
```

This function allows the user to query the Pinning Memory mode. By default, the Pinning Memory mode is disabled.

| Return Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the call has been successful |

# 3.4. cublasXt API Math Functions Reference

In this chapter we describe the actual Linear Agebra routines that cublasXt API supports. We will use abbreviations <*type*> for type and <*t*> for the corresponding short type to make a more concise and clear presentation of the implemented functions. Unless otherwise specified <*type*> and <*t*> have the following meanings:

| <type> | <t> | Meaning |
|---|---|---|
| `float` | 's' or 'S' | real single-precision |
| `double` | 'd' or 'D' | real double-precision |
| `cuComplex` | 'c' or 'C' | complex single-precision |
| `cuDoubleComplex` | 'z' or 'Z' | complex double-precision |

The abbreviation **Re**(.) and **Im**(.) will stand for the real and imaginary part of a number, respectively. Since imaginary part of a real number does not exist, we will consider it to be zero and can usually simply discard it from the equation where it is being used. Also, the $\bar{\alpha}$ will denote the complex conjugate of $\alpha$ .

In general throughout the documentation, the lower case Greek symbols $\alpha$ and $\beta$ will denote scalars, lower case English letters in bold type $\mathbf{x}$ and $\mathbf{y}$ will denote vectors and capital English letters $A$ , $B$ and $C$ will denote matrices.

## 3.4.1. cublasXt<t>gemm()

```
cublasStatus_t cublasXtSgemm(cublasXtHandle_t handle,
                             cublasOperation_t transa, cublasOperation_t transb,
                             size_t m, size_t n, size_t k,
                             const float           *alpha,
                             const float           *A, int lda,
                             const float           *B, int ldb,
                             const float           *beta,
                             float           *C, int ldc)
cublasStatus_t cublasXtDgemm(cublasXtHandle_t handle,
                             cublasOperation_t transa, cublasOperation_t transb,
                             int m, int n, int k,
                             const double          *alpha,
                             const double          *A, int lda,
                             const double          *B, int ldb,
                             const double          *beta,
                             double          *C, int ldc)
cublasStatus_t cublasXtCgemm(cublasXtHandle_t handle,
                             cublasOperation_t transa, cublasOperation_t transb,
                             int m, int n, int k,
                             const cuComplex       *alpha,
                             const cuComplex       *A, int lda,
                             const cuComplex       *B, int ldb,
                             const cuComplex       *beta,
                             cuComplex       *C, int ldc)
cublasStatus_t cublasXtZgemm(cublasXtHandle_t handle,
                             cublasOperation_t transa, cublasOperation_t transb,
                             int m, int n, int k,
                             const cuDoubleComplex *alpha,
                             const cuDoubleComplex *A, int lda,
                             const cuDoubleComplex *B, int ldb,
                             const cuDoubleComplex *beta,
                             cuDoubleComplex *C, int ldc)
```

This function performs the matrix-matrix multiplication

$$C = \alpha \text{op}(A)\text{op}(B) + \beta C$$

where $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are matrices stored in column-major format with dimensions $\text{op}(A)$ $m \times k$, $\text{op}(B)$ $k \times n$ and $C$ $m \times n$, respectively. Also, for matrix $A$

$$\text{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

and $\text{op}(B)$ is defined similarly for matrix $B$.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| transa | | input | operation op(A) that is non- or (conj.) transpose. |
| transb | | input | operation op(B) that is non- or (conj.) transpose. |
| m | | input | number of rows of matrix op(A) and C. |
| n | | input | number of columns of matrix op(B) and C. |
| k | | input | number of columns of op(A) and rows of op(B). |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| alpha | host | input | <type> scalar used for multiplication. |
| A | host or device | input | <type> array of dimensions `lda x k` with `lda>=max(1,m)` if `transa == CUBLAS_OP_N` and `lda x m` with `lda>=max(1,k)` otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store the matrix `A`. |
| B | host or device | input | <type> array of dimension `ldb x n` with `ldb>=max(1,k)` if `transa == CUBLAS_OP_N` and `ldb x k` with `ldb>=max(1,n)` otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix `B`. |
| beta | host | input | <type> scalar used for multiplication. If `beta==0`, `C` does not have to be a valid input. |
| C | host or device | in/out | <type> array of dimensions `ldc x n` with `ldc>=max(1,m)`. |
| ldc | | input | leading dimension of a two-dimensional array used to store the matrix `C`. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `m,n,k<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

sgemm, dgemm, cgemm, zgemm

## 3.4.2. cublasXt<t>hemm()

```
cublasStatus_t cublasXtChemm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             size_t m, size_t n,
                             const cuComplex       *alpha,
                             const cuComplex       *A, size_t lda,
                             const cuComplex       *B, size_t ldb,
                             const cuComplex       *beta,
                             cuComplex       *C, size_t ldc)
cublasStatus_t cublasXtZhemm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             size_t m, size_t n,
                             const cuDoubleComplex *alpha,
                             const cuDoubleComplex *A, size_t lda,
                             const cuDoubleComplex *B, size_t ldb,
                             const cuDoubleComplex *beta,
                             cuDoubleComplex *C, size_t ldc)
```

This function performs the Hermitian matrix-matrix multiplication

$$C = \begin{cases} \alpha AB + \beta C & \text{if side == CUBLAS\_SIDE\_LEFT} \\ \alpha BA + \beta C & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A$ is a Hermitian matrix stored in lower or upper mode, $B$ and $C$ are $m \times n$ matrices, and $\alpha$ and $\beta$ are scalars.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| side | | input | indicates if matrix **A** is on the left or right of **B**. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| m | | input | number of rows of matrix **C** and **B**, with matrix **A** sized accordingly. |
| n | | input | number of columns of matrix **C** and **B**, with matrix **A** sized accordingly. |
| alpha | host | input | <type> scalar used for multiplication. |
| A | host or device | input | <type> array of dimension **lda x m** with **lda>=max(1,m)** if **side==CUBLAS_SIDE_LEFT** and **lda x n** with **lda>=max(1,n)** otherwise. The imaginary parts of the diagonal elements are assumed to be zero. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | host or device | input | <type> array of dimension **ldb x n** with **ldb>=max(1,m)**. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host | input | <type> scalar used for multiplication, if **beta==0** then **C** does not have to be a valid input. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| C | host or device | in/out | <type> array of dimensions `ldc x n` with `ldc>=max(1,m)`. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix `C`. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters `m,n<0` |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

chemm, zhemm

## 3.4.3. cublasXt<t>symm()

```
cublasStatus_t cublasXtSsymm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             size_t m, size_t n,
                             const float           *alpha,
                             const float           *A, size_t lda,
                             const float           *B, size_t ldb,
                             const float           *beta,
                             float           *C, size_t ldc)
cublasStatus_t cublasXtDsymm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             size_t m, size_t n,
                             const double          *alpha,
                             const double          *A, size_t lda,
                             const double          *B, size_t ldb,
                             const double          *beta,
                             double          *C, size_t ldc)
cublasStatus_t cublasXtCsymm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             size_t m, size_t n,
                             const cuComplex       *alpha,
                             const cuComplex       *A, size_t lda,
                             const cuComplex       *B, size_t ldb,
                             const cuComplex       *beta,
                             cuComplex       *C, size_t ldc)
cublasStatus_t cublasXtZsymm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             size_t m, size_t n,
                             const cuDoubleComplex *alpha,
                             const cuDoubleComplex *A, size_t lda,
                             const cuDoubleComplex *B, size_t ldb,
                             const cuDoubleComplex *beta,
                             cuDoubleComplex *C, size_t ldc)
```

This function performs the symmetric matrix-matrix multiplication

$$C = \begin{cases} \alpha AB + \beta C & \text{if side == CUBLAS\_SIDE\_LEFT} \\ \alpha BA + \beta C & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A$ is a symmetric matrix stored in lower or upper mode, $A$ and $A$ are $m \times n$ matrices, and $\alpha$ and $\beta$ are scalars.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| side | | input | indicates if matrix **A** is on the left or right of **B**. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| m | | input | number of rows of matrix **A** and **B**, with matrix **A** sized accordingly. |
| n | | input | number of columns of matrix **C** and **A**, with matrix **A** sized accordingly. |
| alpha | host | input | <type> scalar used for multiplication. |
| A | host or device | input | <type> array of dimension **lda x m** with **lda>=max(1,m)** if **side == CUBLAS_SIDE_LEFT** and **lda x n** with **lda>=max(1,n)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | host or device | input | <type> array of dimension **ldb x n** with **ldb>=max(1,m)**. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host | input | <type> scalar used for multiplication, if **beta == 0** then **C** does not have to be a valid input. |
| C | host or device | in/out | <type> array of dimension **ldc x n** with **ldc>=max(1,m)**. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

ssymm, dsymm, csymm, zsymm

## 3.4.4. cublasXt<t>syrk()

```
cublasStatus_t cublasXtSsyrk(cublasXtHandle_t handle,
                             cublasFillMode_t uplo, cublasOperation_t trans,
                             int n, int k,
                             const float           *alpha,
                             const float           *A, int lda,
                             const float           *beta,
                             float           *C, int ldc)
cublasStatus_t cublasXtDsyrk(cublasXtHandle_t handle,
                             cublasFillMode_t uplo, cublasOperation_t trans,
                             int n, int k,
                             const double          *alpha,
                             const double          *A, int lda,
                             const double          *beta,
                             double          *C, int ldc)
cublasStatus_t cublasXtCsyrk(cublasXtHandle_t handle,
                             cublasFillMode_t uplo, cublasOperation_t trans,
                             int n, int k,
                             const cuComplex       *alpha,
                             const cuComplex       *A, int lda,
                             const cuComplex       *beta,
                             cuComplex       *C, int ldc)
cublasStatus_t cublasXtZsyrk(cublasXtHandle_t handle,
                             cublasFillMode_t uplo, cublasOperation_t trans,
                             int n, int k,
                             const cuDoubleComplex *alpha,
                             const cuDoubleComplex *A, int lda,
                             const cuDoubleComplex *beta,
                             cuDoubleComplex *C, int ldc)
```

This function performs the symmetric rank- $k$ update

$$C = \alpha \mathrm{op}(A) \mathrm{op}(A)^T + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a symmetric matrix stored in lower or upper mode, and $A$ is a matrix with dimensions $\mathrm{op}(A)$ $n \times k$. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| uplo | | input | indicates if matrix C lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or transpose. |
| n | | input | number of rows of matrix op(**A**) and C. |
| k | | input | number of columns of matrix op(**A**). |
| alpha | host | input | <type> scalar used for multiplication. |
| A | host or device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **trans == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| lda | | input | leading dimension of two-dimensional array used to store matrix A. |
| beta | host | input | <type> scalar used for multiplication, if `beta==0` then `C` does not have to be a valid input. |
| C | host or device | in/out | <type> array of dimension `ldc x n`, with `ldc>=max(1,n)`. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix `C`. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `n,k<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

ssyrk, dsyrk, csyrk, zsyrk

## 3.4.5. cublasXt<t>syr2k()

```
cublasStatus_t cublasXtSsyr2k(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const float           *alpha,
                              const float           *A, size_t lda,
                              const float           *B, size_t ldb,
                              const float           *beta,
                              float           *C, size_t ldc)
cublasStatus_t cublasXtDsyr2k(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const double          *alpha,
                              const double          *A, size_t lda,
                              const double          *B, size_t ldb,
                              const double          *beta,
                              double          *C, size_t ldc)
cublasStatus_t cublasXtCsyr2k(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const cuComplex       *alpha,
                              const cuComplex       *A, size_t lda,
                              const cuComplex       *B, size_t ldb,
                              const cuComplex       *beta,
                              cuComplex       *C, size_t ldc)
cublasStatus_t cublasXtZsyr2k(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const cuDoubleComplex *alpha,
                              const cuDoubleComplex *A, size_t lda,
                              const cuDoubleComplex *B, size_t ldb,
                              const cuDoubleComplex *beta,
                              cuDoubleComplex *C, size_t ldc)
```

This function performs the symmetric rank- $2k$ update

$$C = \alpha(\mathrm{op}(A)\mathrm{op}(B)^T + \mathrm{op}(B)\mathrm{op}(A)^T) + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a symmetric matrix stored in lower or upper mode, and $A$ and $B$ are matrices with dimensions $\mathrm{op}(A)$ $n \times k$ and $\mathrm{op}(B)$ $n \times k$, respectively. Also, for matrix $A$ and $B$

$$\mathrm{op}(A) \text{ and } \mathrm{op}(B) = \begin{cases} A \text{ and } B & \text{if trans == CUBLAS\_OP\_N} \\ A^T \text{ and } B^T & \text{if trans == CUBLAS\_OP\_T} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| uplo | | input | indicates if matrix c lower or upper part, is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or transpose. |
| n | | input | number of rows of matrix op(**A**), op(**B**) and **C**. |
| k | | input | number of columns of matrix op(**A**) and op(**B**). |
| alpha | host | input | <type> scalar used for multiplication. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| A | host or device | input | <type> array of dimension `lda x k` with `lda>=max(1,n)` if `transa == CUBLAS_OP_N` and `lda x n` with `lda>=max(1,k)` otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix `A`. |
| B | host or device | input | <type> array of dimensions `ldb x k` with `ldb>=max(1,n)` if `transa == CUBLAS_OP_N` and `ldb x n` with `ldb>=max(1,k)` otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix `B`. |
| beta | host | input | <type> scalar used for multiplication, if `beta==0`, then `C` does not have to be a valid input. |
| C | host or device | in/out | <type> array of dimensions `ldc x n` with `ldc>=max(1,n)`. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix `C`. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `n,k<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

ssyr2k, dsyr2k, csyr2k, zsyr2k

## 3.4.6. cublasXt<t>syrkx()

```
cublasStatus_t cublasXtSsyrkx(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const float           *alpha,
                              const float           *A, size_t lda,
                              const float           *B, size_t ldb,
                              const float           *beta,
                              float           *C, size_t ldc)
cublasStatus_t cublasXtDsyrkx(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const double          *alpha,
                              const double          *A, size_t lda,
                              const double          *B, size_t ldb,
                              const double          *beta,
                              double          *C, size_t ldc)
cublasStatus_t cublasXtCsyrkx(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const cuComplex       *alpha,
                              const cuComplex       *A, size_t lda,
                              const cuComplex       *B, size_t ldb,
                              const cuComplex       *beta,
                              cuComplex       *C, size_t ldc)
cublasStatus_t cublasXtZsyrkx(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const cuDoubleComplex *alpha,
                              const cuDoubleComplex *A, size_t lda,
                              const cuDoubleComplex *B, size_t ldb,
                              const cuDoubleComplex *beta,
                              cuDoubleComplex *C, size_t ldc)
```

This function performs a variation of the symmetric rank- $k$ update

$$C = \alpha(\mathrm{op}(A)\mathrm{op}(B)^T + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a symmetric matrix stored in lower or upper mode, and $A$ and $B$ are matrices with dimensions $\mathrm{op}(A)$ $n \times k$ and $\mathrm{op}(B)$ $n \times k$, respectively. Also, for matrix $A$ and $B$

$$\mathrm{op}(A) \text{ and } \mathrm{op}(B) = \begin{cases} A \text{ and } B & \text{if trans == CUBLAS\_OP\_N} \\ A^T \text{ and } B^T & \text{if trans == CUBLAS\_OP\_T} \end{cases}$$

This routine can be used when B is in such way that the result is garanteed to be symmetric. An usual example is when the matrix B is a scaled form of the matrix A : this is equivalent to B being the product of the matrix A and a diagonal matrix.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| uplo | | input | indicates if matrix C lower or upper part, is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(A) that is non- or transpose. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| n | | input | number of rows of matrix op(A), op(B) and C. |
| k | | input | number of columns of matrix op(A) and op(B). |
| alpha | host | input | <type> scalar used for multiplication. |
| A | host or device | input | <type> array of dimension `lda x k` with `lda>=max(1,n)` if `transa == CUBLAS_OP_N` and `lda x n` with `lda>=max(1,k)` otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix A. |
| B | host or device | input | <type> array of dimensions `ldb x k` with `ldb>=max(1,n)` if `transa == CUBLAS_OP_N` and `ldb x n` with `ldb>=max(1,k)` otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix B. |
| beta | host | input | <type> scalar used for multiplication, if `beta==0`, then C does not have to be a valid input. |
| C | host or device | in/out | <type> array of dimensions `ldc x n` with `ldc>=max(1,n)`. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix C. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `n,k<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

ssyrk, dsyrk, csyrk, zsyrk and

ssyr2k, dsyr2k, csyr2k, zsyr2k

## 3.4.7. cublasXt<t>herk()

```
cublasStatus_t cublasXtCherk(cublasXtHandle_t handle,
                             cublasFillMode_t uplo, cublasOperation_t trans,
                             int n, int k,
                             const float  *alpha,
                             const cuComplex       *A, int lda,
                             const float  *beta,
                             cuComplex       *C, int ldc)
cublasStatus_t cublasXtZherk(cublasXtHandle_t handle,
                             cublasFillMode_t uplo, cublasOperation_t trans,
                             int n, int k,
                             const double *alpha,
                             const cuDoubleComplex *A, int lda,
                             const double *beta,
                             cuDoubleComplex *C, int ldc)
```

This function performs the Hermitian rank- $k$ update

$$C = \alpha \mathrm{op}(A) \mathrm{op}(A)^H + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a Hermitian matrix stored in lower or upper mode, and $A$ is a matrix with dimensions $\mathrm{op}(A)$ $n \times k$. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| n | | input | number of rows of matrix op(**A**) and **C**. |
| k | | input | number of columns of matrix op(**A**). |
| alpha | host | input | <type> scalar used for multiplication. |
| A | host or device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **transa == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| beta | host | input | <type> scalar used for multiplication, if **beta==0** then **C** does not have to be a valid input. |
| C | host or device | in/out | <type> array of dimension **ldc x n**, with **ldc>=max(1,n)**. The imaginary parts of the diagonal elements are assumed and set to zero. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

cherk, zherk

## 3.4.8. cublasXt<t>her2k()

```
cublasStatus_t cublasXtCher2k(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const cuComplex       *alpha,
                              const cuComplex       *A, size_t lda,
                              const cuComplex       *B, size_t ldb,
                              const float  *beta,
                              cuComplex       *C, size_t ldc)
cublasStatus_t cublasXtZher2k(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const cuDoubleComplex *alpha,
                              const cuDoubleComplex *A, size_t lda,
                              const cuDoubleComplex *B, size_t ldb,
                              const double *beta,
                              cuDoubleComplex *C, size_t ldc)
```

This function performs the Hermitian rank- $2k$ update

$$C = \alpha \mathrm{op}(A)\mathrm{op}(B)^H + \bar{\alpha}\mathrm{op}(B)\mathrm{op}(A)^H + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a Hermitian matrix stored in lower or upper mode, and $A$ and $B$ are matrices with dimensions $\mathrm{op}(A)$ $n \times k$ and $\mathrm{op}(B)$ $n \times k$, respectively. Also, for matrix $A$ and $B$

$$\mathrm{op}(A) \text{ and } \mathrm{op}(B) = \begin{cases} A \text{ and } B & \text{if trans == CUBLAS\_OP\_N} \\ A^H \text{ and } B^H & \text{if trans == CUBLAS\_OP\_C} \end{cases}$$

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| n | | input | number of rows of matrix op(**A**), op(**B**) and **C**. |
| k | | input | number of columns of matrix op(**A**) and op(**B**). |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| alpha | host | input | <type> scalar used for multiplication. |
| A | host or device | input | <type> array of dimension `lda x k` with `lda>=max(1,n)` if `transa == CUBLAS_OP_N` and `lda x n` with `lda>=max(1,k)` otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix `A`. |
| B | host or device | input | <type> array of dimension `ldb x k` with `ldb>=max(1,n)` if `transa == CUBLAS_OP_N` and `ldb x n` with `ldb>=max(1,k)` otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix `B`. |
| beta | host | input | <type> scalar used for multiplication, if `beta==0` then `C` does not have to be a valid input. |
| C | host or device | in/out | <type> array of dimension `ldc x n`, with `ldc>=max(1,n)`. The imaginary parts of the diagonal elements are assumed and set to zero. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix `C`. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| `CUBLAS_STATUS_SUCCESS` | the operation completed successfully |
| `CUBLAS_STATUS_NOT_INITIALIZED` | the library was not initialized |
| `CUBLAS_STATUS_INVALID_VALUE` | the parameters `n,k<0` |
| `CUBLAS_STATUS_ARCH_MISMATCH` | the device does not support double-precision |
| `CUBLAS_STATUS_EXECUTION_FAILED` | the function failed to launch on the GPU |

For references please refer to:

cher2k, zher2k

## 3.4.9. cublasXt<t>herkx()

```
cublasStatus_t cublasXtCherkx(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const cuComplex       *alpha,
                              const cuComplex       *A, size_t lda,
                              const cuComplex       *B, size_t ldb,
                              const float  *beta,
                              cuComplex       *C, size_t ldc)
cublasStatus_t cublasXtZherkx(cublasXtHandle_t handle,
                              cublasFillMode_t uplo, cublasOperation_t trans,
                              size_t n, size_t k,
                              const cuDoubleComplex *alpha,
                              const cuDoubleComplex *A, size_t lda,
                              const cuDoubleComplex *B, size_t ldb,
                              const double *beta,
                              cuDoubleComplex *C, size_t ldc)
```

This function performs a variation of the Hermitian rank- $k$ update

$$C = \alpha \mathrm{op}(A) \mathrm{op}(B)^H + \beta C$$

where $\alpha$ and $\beta$ are scalars, $C$ is a Hermitian matrix stored in lower or upper mode, and $A$ and $B$ are matrices with dimensions $\mathrm{op}(A)$ $n \times k$ and $\mathrm{op}(B)$ $n \times k$, respectively. Also, for matrix $A$ and $B$

$$\mathrm{op}(A) \text{ and } \mathrm{op}(B) = \begin{cases} A \text{ and } B & \text{if trans == CUBLAS\_OP\_N} \\ A^H \text{ and } B^H & \text{if trans == CUBLAS\_OP\_C} \end{cases}$$

This routine can be used when the matrix B is in such way that the result is garanteed to be hermitian. An usual example is when the matrix B is a scaled form of the matrix A : this is equivalent to B being the product of the matrix A and a diagonal matrix. For an efficient computation of the product of a regular matrix with a diagonal matrix, refer to the routine cublasXt<t>dgmm.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other Hermitian part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| n | | input | number of rows of matrix op(**A**), op(**B**) and **C**. |
| k | | input | number of columns of matrix op(**A**) and op(**B**). |
| alpha | host | input | <type> scalar used for multiplication. |
| A | host or device | input | <type> array of dimension **lda x k** with **lda>=max(1,n)** if **transa == CUBLAS_OP_N** and **lda x n** with **lda>=max(1,k)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| B | host or device | input | <type> array of dimension **ldb x k** with **ldb>=max(1,n)** if **transa == CUBLAS_OP_N** and **ldb x n** with **ldb>=max(1,k)** otherwise. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host | input | real scalar used for multiplication, if **beta==0** then **C** does not have to be a valid input. |
| C | host or device | in/out | <type> array of dimension **ldc x n**, with **ldc>=max(1,n)**. The imaginary parts of the diagonal elements are assumed and set to zero. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **n,k<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

cherk, zherk and

cher2k, zher2k

## 3.4.10. cublasXt<t>trsm()

```
cublasStatus_t cublasXtStrsm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             cublasOperation_t trans, cublasXtDiagType_t diag,
                             size_t m, size_t n,
                             const float           *alpha,
                             const float           *A, size_t lda,
                             float           *B, size_t ldb)
cublasStatus_t cublasXtDtrsm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             cublasOperation_t trans, cublasXtDiagType_t diag,
                             size_t m, size_t n,
                             const double          *alpha,
                             const double          *A, size_t lda,
                             double          *B, size_t ldb)
cublasStatus_t cublasXtCtrsm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             cublasOperation_t trans, cublasXtDiagType_t diag,
                             size_t m, size_t n,
                             const cuComplex       *alpha,
                             const cuComplex       *A, size_t lda,
                             cuComplex       *B, size_t ldb)
cublasStatus_t cublasXtZtrsm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             cublasOperation_t trans, cublasXtDiagType_t diag,
                             size_t m, size_t n,
                             const cuDoubleComplex *alpha,
                             const cuDoubleComplex *A, size_t lda,
                             cuDoubleComplex *B, size_t ldb)
```

This function solves the triangular linear system with multiple right-hand-sides

$$\begin{cases} \mathrm{op}(A)X = \alpha B & \text{if side == CUBLAS\_SIDE\_LEFT} \\ X\mathrm{op}(A) = \alpha B & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A$ is a triangular matrix stored in lower or upper mode with or without the main diagonal, $X$ and $B$ are $m \times n$ matrices, and $\alpha$ is a scalar. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

The solution $X$ overwrites the right-hand-sides $B$ on exit.

No test for singularity or near-singularity is included in this function.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| side | | input | indicates if matrix **A** is on the left or right of **X**. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix **A** are unity and should not be accessed. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| m | | input | number of rows of matrix **B**, with matrix **A** sized accordingly. |
| n | | input | number of columns of matrix **B**, with matrix **A** is sized accordingly. |
| alpha | host | input | <type> scalar used for multiplication, if **alpha==0** then **A** is not referenced and **B** does not have to be a valid input. |
| A | host or device | input | <type> array of dimension **lda x m** with **lda>=max(1,m)** if **side == CUBLAS_SIDE_LEFT** and **lda x n** with **lda>=max(1,n)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | host or device | in/out | <type> array. It has dimensions **ldb x n** with **ldb>=max(1,m)**. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

strsm, dtrsm, ctrsm, ztrsm

## 3.4.11. cublasXt<t>trmm()

```
cublasStatus_t cublasXtStrmm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             cublasOperation_t trans, cublasDiagType_t diag,
                             size_t m, size_t n,
                             const float           *alpha,
                             const float           *A, size_t lda,
                             const float           *B, size_t ldb,
                             float                 *C, size_t ldc)
cublasStatus_t cublasXtDtrmm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             cublasOperation_t trans, cublasDiagType_t diag,
                             size_t m, size_t n,
                             const double          *alpha,
                             const double          *A, size_t lda,
                             const double          *B, size_t ldb,
                             double                *C, size_t ldc)
cublasStatus_t cublasXtCtrmm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             cublasOperation_t trans, cublasDiagType_t diag,
                             size_t m, size_t n,
                             const cuComplex       *alpha,
                             const cuComplex       *A, size_t lda,
                             const cuComplex       *B, size_t ldb,
                             cuComplex             *C, size_t ldc)
cublasStatus_t cublasXtZtrmm(cublasXtHandle_t handle,
                             cublasSideMode_t side, cublasFillMode_t uplo,
                             cublasOperation_t trans, cublasDiagType_t diag,
                             size_t m, size_t n,
                             const cuDoubleComplex *alpha,
                             const cuDoubleComplex *A, size_t lda,
                             const cuDoubleComplex *B, size_t ldb,
                             cuDoubleComplex       *C, size_t ldc)
```

This function performs the triangular matrix-matrix multiplication

$$C = \begin{cases} \alpha \mathrm{op}(A) B & \text{if side == CUBLAS\_SIDE\_LEFT} \\ \alpha B \mathrm{op}(A) & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A$ is a triangular matrix stored in lower or upper mode with or without the main diagonal, $B$ and $C$ are $m \times n$ matrix, and $\alpha$ is a scalar. Also, for matrix $A$

$$\mathrm{op}(A) = \begin{cases} A & \text{if transa == CUBLAS\_OP\_N} \\ A^T & \text{if transa == CUBLAS\_OP\_T} \\ A^H & \text{if transa == CUBLAS\_OP\_C} \end{cases}$$

Notice that in order to achieve better parallelism, similarly to the cublas API, cublasXT API differs from the BLAS API for this routine. The BLAS API assumes an in-place implementation (with results written back to B), while the cublasXt API assumes an out-of-place implementation (with results written into C). The application can still obtain the in-place functionality of BLAS in the cublasXT API by passing the address of the matrix B in place of the matrix C. No other overlapping in the input parameters is supported.

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| side | | input | indicates if matrix **A** is on the left or right of **B**. |

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. |
| trans | | input | operation op(**A**) that is non- or (conj.) transpose. |
| diag | | input | indicates if the elements on the main diagonal of matrix **A** are unity and should not be accessed. |
| m | | input | number of rows of matrix **B**, with matrix **A** sized accordingly. |
| n | | input | number of columns of matrix **B**, with matrix **A** sized accordingly. |
| alpha | host | input | <type> scalar used for multiplication, if **alpha==0** then **A** is not referenced and **B** does not have to be a valid input. |
| A | host or device | input | <type> array of dimension **lda x m** with **lda>=max(1,m)** if **side == CUBLAS_SIDE_LEFT** and **lda x n** with **lda>=max(1,n)** otherwise. |
| lda | | input | leading dimension of two-dimensional array used to store matrix **A**. |
| B | host or device | input | <type> array of dimension **ldb x n** with **ldb>=max(1,m).** |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| C | host or device | in/out | <type> array of dimension **ldc x n** with **ldc>=max(1,m).** |
| ldc | | input | leading dimension of two-dimensional array used to store matrix C. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |
| **CUBLAS_STATUS_INVALID_VALUE** | the parameters **m,n<0** |
| **CUBLAS_STATUS_ARCH_MISMATCH** | the device does not support double-precision |
| **CUBLAS_STATUS_EXECUTION_FAILED** | the function failed to launch on the GPU |

For references please refer to:

strmm, dtrmm, ctrmm, ztrmm

## 3.4.12. cublasXt<t>spmm()

```
cublasStatus_t cublasXtSspmm( cublasXtHandle_t handle,
                              cublasSideMode_t side,
                              cublasFillMode_t uplo,
                              size_t m,
                              size_t n,
                              const float *alpha,
                              const float *AP,
                              const float *B,
                              size_t ldb,
                              const float *beta,
                              float *C,
                              size_t ldc );

cublasStatus_t cublasXtDspmm( cublasXtHandle_t handle,
                              cublasSideMode_t side,
                              cublasFillMode_t uplo,
                              size_t m,
                              size_t n,
                              const double *alpha,
                              const double *AP,
                              const double *B,
                              size_t ldb,
                              const double *beta,
                              double *C,
                              size_t ldc );

cublasStatus_t cublasXtCspmm( cublasXtHandle_t handle,
                              cublasSideMode_t side,
                              cublasFillMode_t uplo,
                              size_t m,
                              size_t n,
                              const cuComplex *alpha,
                              const cuComplex *AP,
                              const cuComplex *B,
                              size_t ldb,
                              const cuComplex *beta,
                              cuComplex *C,
                              size_t ldc );

cublasStatus_t cublasXtZspmm( cublasXtHandle_t handle,
                              cublasSideMode_t side,
                              cublasFillMode_t uplo,
                              size_t m,
                              size_t n,
                              const cuDoubleComplex *alpha,
                              const cuDoubleComplex *AP,

                              const cuDoubleComplex *B,
                              size_t ldb,
                              const cuDoubleComplex *beta,
                              cuDoubleComplex *C,
                              size_t ldc );
```

This function performs the symmetric packed matrix-matrix multiplication

$$C = \begin{cases} \alpha AB + \beta C & \text{if side == CUBLAS\_SIDE\_LEFT} \\ \alpha BA + \beta C & \text{if side == CUBLAS\_SIDE\_RIGHT} \end{cases}$$

where $A$ is a $n \times n$ symmetric matrix stored in packed format, $B$ and $C$ are $m \times n$ matrices, and $\alpha$ and $\beta$ are scalars.

If **uplo == CUBLAS_FILL_MODE_LOWER** then the elements in the lower triangular part of the symmetric matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+((2*n-j+1)*j)/2]** for $j = 1, \ldots, n$ and $i \geq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

If **uplo == CUBLAS_FILL_MODE_UPPER** then the elements in the upper triangular part of the symmetric matrix $A$ are packed together column by column without gaps, so that the element $A(i, j)$ is stored in the memory location **AP[i+(j*(j+1))/2]** for $j = 1, \ldots, n$ and $i \leq j$. Consequently, the packed format requires only $\frac{n(n+1)}{2}$ elements for storage.

The packed matrix AP must be located on the Host whereas the other matrices can be located on the Host or any GPU device

| Param. | Memory | In/out | Meaning |
|---|---|---|---|
| handle | | input | handle to the cublasXt API context. |
| side | | input | indicates if matrix **A** is on the left or right of **B**. |
| uplo | | input | indicates if matrix **A** lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. |
| m | | input | number of rows of matrix **A** and **B**, with matrix **A** sized accordingly. |
| n | | input | number of columns of matrix **C** and **A**, with matrix **A** sized accordingly. |
| alpha | host | input | <type> scalar used for multiplication. |
| AP | host | input | <type> array with $A$ stored in packed format. |
| B | host or device | input | <type> array of dimension **ldb x n** with **ldb>=max(1,m)**. |
| ldb | | input | leading dimension of two-dimensional array used to store matrix **B**. |
| beta | host | input | <type> scalar used for multiplication, if **beta == 0** then **C** does not have to be a valid input. |
| C | host or device | in/out | <type> array of dimension **ldc x n** with **ldc>=max(1,m)**. |
| ldc | | input | leading dimension of two-dimensional array used to store matrix **C**. |

The possible error values returned by this function and their meanings are listed below.

| Error Value | Meaning |
|---|---|
| **CUBLAS_STATUS_SUCCESS** | the operation completed successfully |
| **CUBLAS_STATUS_NOT_INITIALIZED** | the library was not initialized |

| Error Value | Meaning |
|---|---|
| CUBLAS_STATUS_INVALID_VALUE | the parameters m,n<0 |
| CUBLAS_STATUS_ARCH_MISMATCH | the device does not support double-precision |
| CUBLAS_STATUS_NOT_SUPPORTED | the matrix AP is located on a GPU device |
| CUBLAS_STATUS_EXECUTION_FAILED | the function failed to launch on the GPU |

For references please refer to:

ssymm, dsymm, csymm, zsymm

# Appendix A. USING THE CUBLAS LEGACY API

This appendix does not provide a full reference of each Legacy API datatype and entry point. Instead, it describes how to use the API, especially where this is different from the regular cuBLAS API.

Note that in this section, all references to the "cuBLAS Library" refer to the Legacy cuBLAS API only.

## A.1. Error Status

The **cublasStatus** type is used for function status returns. The cuBLAS Library helper functions return status directly, while the status of core functions can be retrieved using **cublasGetError()**. Notice that reading the error status via **cublasGetError()**, resets the internal error state to **CUBLAS_STATUS_SUCCESS**. Currently, the following values for are defined:

| Value | Meaning |
|---|---|
| CUBLAS_STATUS_SUCCESS | the operation completed successfully |
| CUBLAS_STATUS_NOT_INITIALIZED | the library was not initialized |
| CUBLAS_STATUS_ALLOC_FAILED | the resource allocation failed |
| CUBLAS_STATUS_INVALID_VALUE | an invalid numerical value was used as an argument |
| CUBLAS_STATUS_ARCH_MISMATCH | an absent device architectural feature is required |
| CUBLAS_STATUS_MAPPING_ERROR | an access to GPU memory space failed |
| CUBLAS_STATUS_EXECUTION_FAILED | the GPU program failed to execute |
| CUBLAS_STATUS_INTERNAL_ERROR | an internal operation failed |
| CUBLAS_STATUS_NOT_SUPPORTED | the feature required is not supported |

This legacy type corresponds to type **cublasStatus_t** in the cuBLAS library API.

## A.2. Initialization and Shutdown

The functions **cublasInit()** and **cublasShutdown()** are used to initialize and shutdown the cuBLAS library. It is recommended for **cublasInit()** to be called before any other function is invoked. It allocates hardware resources on the GPU device that is currently bound to the host thread from which it was invoked.

The legacy initialization and shutdown functions are similar to the cuBLAS library API routines **cublasCreate()** and **cublasDestroy()**.

## A.3. Thread Safety

The legacy API is not thread safe when used with multiple host threads and devices. It is recommended to be used only when utmost compatibility with Fortran is required and when a single host thread is used to setup the library and make all the functions calls.

## A.4. Memory Management

The memory used by the legacy cuBLAS library API is allocated and released using functions **cublasAlloc()** and **cublasFree()**, respectively. These functions create and destroy an object in the GPU memory space capable of holding an array of **n** elements, where each element requires **elemSize** bytes of storage. Please see the legacy cuBLAS API header file "cublas.h" for the prototypes of these functions.

The function **cublasAlloc()** is a wrapper around the function **cudaMalloc()**, therefore device pointers returned by **cublasAlloc()** can be passed to any CUDA™ device kernel functions. However, these device pointers can not be dereferenced in the host code. The function **cublasFree()** is a wrapper around the function **cudaFree()**.

## A.5. Scalar Parameters

There are two categories of the functions that use scalar parameters :

- functions that take **alpha** and/or **beta** parameters by reference on the host or the device as scaling factors, such as **gemm**
- functions that return a scalar result on the host or the device such as **amax()**, **amin**, **asum()**, **rotg()**, **rotmg()**, **dot()** and **nrm2()**.

For the functions of the first category, when the pointer mode is set to **CUBLAS_POINTER_MODE_HOST**, the scalar parameters **alpha** and/or **beta** can be on the stack or allocated on the heap. Underneath the CUDA kernels related to that functions will be launched with the value of **alpha** and/or **beta**. Therefore if they were allocated on the heap, they can be freed just after the return of the call even though the kernel launch is asynchronous. When the pointer mode is set to **CUBLAS_POINTER_MODE_DEVICE**, **alpha** and/or **beta** must be accessible on the

device and their values should not be modified until the kernel is done. Note that since **cudaFree()** does an implicit **cudaDeviceSynchronize()**, **cudaFree()** can still be called on **alpha** and/or **beta** just after the call but it would defeat the purpose of using this pointer mode in that case.

For the functions of the second category, when the pointer mode is set to **CUBLAS_POINTER_MODE_HOST**, these functions blocks the CPU, until the GPU has completed its computation and the results has been copied back to the Host. When the pointer mode is set to **CUBLAS_POINTER_MODE_DEVICE**, these functions return immediately. In this case, similarly to matrix and vector results, the scalar result is ready only when execution of the routine on the GPU has completed. This requires proper synchronization in order to read the result from the host.

In either case, the pointer mode **CUBLAS_POINTER_MODE_DEVICE** allows the library functions to execute completely asynchronously from the Host even when **alpha** and/or **beta** are generated by a previous kernel. For example, this situation can arise when iterative methods for solution of linear systems and eigenvalue problems are implemented using the cuBLAS library.

## A.6. Helper Functions

In this section we list the helper functions provided by the legacy cuBLAS API and their functionality. For the exact prototypes of these functions please refer to the legacy cuBLAS API header file "cublas.h".

| Helper function | Meaning |
|---|---|
| **cublasInit()** | initialize the library |
| **cublasShutdown()** | shuts down the library |
| **cublasGetError()** | retrieves the error status of the library |
| **cublasSetKernelStream()** | sets the stream to be used by the library |
| **cublasAlloc()** | allocates the device memory for the library |
| **cublasFree()** | releases the device memory allocated for the library |
| **cublasSetVector()** | copies a vector **x** on the host to a vector on the GPU |
| **cublasGetVector()** | copies a vector **x** on the GPU to a vector on the host |
| **cublasSetMatrix()** | copies a $m \times n$ tile from a matrix on the host to the GPU |
| **cublasGetMatrix()** | copies a $m \times n$ tile from a matrix on the GPU to the host |
| **cublasSetVectorAsync()** | similar to **cublasSetVector()**, but the copy is asynchronous |

| Helper function | Meaning |
|---|---|
| `cublasGetVectorAsync()` | similar to `cublasGetVector()`, but the copy is asynchronous |
| `cublasSetMatrixAsync()` | similar to `cublasSetMatrix()`, but the copy is asynchronous |
| `cublasGetMatrixAsync()` | similar to `cublasGetMatrix()`, but the copy is asynchronous |

## A.7. Level-1,2,3 Functions

The Level-1,2,3 cuBLAS functions (also called core functions) have the same name and behavior as the ones listed in the chapters 3, 4 and 5 in this document. Please refer to the legacy cuBLAS API header file "cublas.h" for their exact prototype. Also, the next section talks a bit more about the differences between the legacy and the cuBLAS API prototypes, more specifically how to convert the function calls from one API to another.

## A.8. Converting Legacy to the cuBLAS API

There are a few general rules that can be used to convert from legacy to the cuBLAS API.

Exchange the header file "cublas.h" for "cublas_v2.h".

Exchange the type **`cublasStatus`** for **`cublasStatus_t`**.

Exchange the function **`cublasSetKernelStream()`** for **`cublasSetStream()`**.

Exchange the function **`cublasAlloc()`** and **`cublasFree()`** for **`cudaMalloc()`** and **`cudaFree()`**, respectively. Notice that **`cudaMalloc()`** expects the size of the allocated memory to be provided in bytes (usually simply provide **`n x elemSize`** to allocate **`n`** elements, each of size **`elemSize`** bytes).

Declare the **`cublasHandle_t`** cuBLAS library handle.

Initialize the handle using **`cublasCreate()`**. Also, release the handle once finished using **`cublasDestroy()`**.

Add the handle as the first parameter to all the cuBLAS library function calls.

Change the scalar parameters to be passed by reference, instead of by value (usually simply adding "&" symbol in C/C++ is enough, because the parameters are passed by reference on the host by *default*). However, note that if the routine is running asynchronously, then the variable holding the scalar parameter cannot be changed until the kernels that the routine dispatches are completed. See the CUDA C Programming Guide for a detailed discussion of how to use streams.

Change the parameter characters **`'N'`** or **`'n'`** (non-transpose operation), **`'T'`** or **`'t'`** (transpose operation) and **`'C'`** or **`'c'`** (conjugate transpose operation) to **`CUBLAS_OP_N`**, **`CUBLAS_OP_T`** and **`CUBLAS_OP_C`**, respectively.

Change the parameter characters **'L'** or **'l'** (lower part filled) and **'U'** or **'u'** (upper part filled) to **CUBLAS_FILL_MODE_LOWER** and **CUBLAS_FILL_MODE_UPPER**, respectively.

Change the parameter characters **'N'** or **'n'** (non-unit diagonal) and **'U'** or **'u'** (unit diagonal) to **CUBLAS_DIAG_NON_UNIT** and **CUBLAS_DIAG_UNIT**, respectively.

Change the parameter characters **'L'** or **'l'** (left side) and **'R'** or **'r'** (right side) to **CUBLAS_SIDE_LEFT** and **CUBLAS_SIDE_RIGHT**, respectively.

If the legacy API function returns a scalar value, add an extra scalar parameter of the same type passed by reference, as the last parameter to the same function.

Instead of using **cublasGetError**, use the return value of the function itself to check for errors.

Finally, please use the function prototypes in the header files “cublas.h” and “cublas_v2.h” to check the code for correctness.

## A.9. Examples

For sample code references that use the legacy cuBLAS API please see the two examples below. They show an application written in C using the legacy cuBLAS library API with two indexing styles (Example A.1. "Application Using C and cuBLAS: 1-based indexing" and Example A.2. "Application Using C and cuBLAS: 0-based Indexing"). This application is analogous to the one using the cuBLAS library API that is shown in the Introduction chapter.

Example A.1. Application Using C and cuBLAS: 1-based indexing

```
//-----------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cublas.h"
#define M 6
#define N 5
#define IDX2F(i,j,ld) ((((j)-1)*(ld))+((i)-1))

static __inline__ void modify (float *m, int ldm, int n, int p, int q, float
 alpha, float beta){
    cublasSscal (n-p+1, alpha, &m[IDX2F(p,q,ldm)], ldm);
    cublasSscal (ldm-p+1, beta, &m[IDX2F(p,q,ldm)], 1);
}

int main (void){
    int i, j;
    cublasStatus stat;
    float* devPtrA;
    float* a = 0;
    a = (float *)malloc (M * N * sizeof (*a));
    if (!a) {
        printf ("host memory allocation failed");
        return EXIT_FAILURE;
    }
    for (j = 1; j <= N; j++) {
        for (i = 1; i <= M; i++) {
            a[IDX2F(i,j,M)] = (float)((i-1) * M + j);
        }
    }
    cublasInit();
    stat = cublasAlloc (M*N, sizeof(*a), (void**)&devPtrA);
    if (stat != cuBLAS_STATUS_SUCCESS) {
        printf ("device memory allocation failed");
        cublasShutdown();
        return EXIT_FAILURE;
    }
    stat = cublasSetMatrix (M, N, sizeof(*a), a, M, devPtrA, M);
    if (stat != cuBLAS_STATUS_SUCCESS) {
        printf ("data download failed");
        cublasFree (devPtrA);
        cublasShutdown();
        return EXIT_FAILURE;
    }
    modify (devPtrA, M, N, 2, 3, 16.0f, 12.0f);
    stat = cublasGetMatrix (M, N, sizeof(*a), devPtrA, M, a, M);
    if (stat != cuBLAS_STATUS_SUCCESS) {
        printf ("data upload failed");
        cublasFree (devPtrA);
        cublasShutdown();
        return EXIT_FAILURE;
    }
    cublasFree (devPtrA);
    cublasShutdown();
    for (j = 1; j <= N; j++) {
        for (i = 1; i <= M; i++) {
            printf ("%7.0f", a[IDX2F(i,j,M)]);
        }
        printf ("\n");
    }
    free(a);
    return EXIT_SUCCESS;
}
```

Example A.2. Application Using C and cuBLAS: 0-based indexing

```
//-----------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cublas.h"
#define M 6
#define N 5
#define IDX2C(i,j,ld) (((j)*(ld))+(i))

static __inline__ void modify (float *m, int ldm, int n, int p, int q, float
 alpha, float beta){
    cublasSscal (n-p, alpha, &m[IDX2C(p,q,ldm)], ldm);
    cublasSscal (ldm-p, beta, &m[IDX2C(p,q,ldm)], 1);
}

int main (void){
    int i, j;
    cublasStatus stat;
    float* devPtrA;
    float* a = 0;
    a = (float *)malloc (M * N * sizeof (*a));
    if (!a) {
        printf ("host memory allocation failed");
        return EXIT_FAILURE;
    }
    for (j = 0; j < N; j++) {
        for (i = 0; i < M; i++) {
            a[IDX2C(i,j,M)] = (float)(i * M + j + 1);
        }
    }
    cublasInit();
    stat = cublasAlloc (M*N, sizeof(*a), (void**)&devPtrA);
    if (stat != cuBLAS_STATUS_SUCCESS) {
        printf ("device memory allocation failed");
        cublasShutdown();
        return EXIT_FAILURE;
    }
    stat = cublasSetMatrix (M, N, sizeof(*a), a, M, devPtrA, M);
    if (stat != cuBLAS_STATUS_SUCCESS) {
        printf ("data download failed");
        cublasFree (devPtrA);
        cublasShutdown();
        return EXIT_FAILURE;
    }
    modify (devPtrA, M, N, 1, 2, 16.0f, 12.0f);
    stat = cublasGetMatrix (M, N, sizeof(*a), devPtrA, M, a, M);
    if (stat != cuBLAS_STATUS_SUCCESS) {
        printf ("data upload failed");
        cublasFree (devPtrA);
        cublasShutdown();
        return EXIT_FAILURE;
    }
    cublasFree (devPtrA);
    cublasShutdown();
    for (j = 0; j < N; j++) {
        for (i = 0; i < M; i++) {
            printf ("%7.0f", a[IDX2C(i,j,M)]);
        }
        printf ("\n");
    }
    free(a);
    return EXIT_SUCCESS;
}
```

# Appendix B. CUBLAS FORTRAN BINDINGS

The cuBLAS library is implemented using the C-based CUDA toolchain, and thus provides a C-style API. This makes interfacing to applications written in C and C++ trivial, but the library can also be used by applications written in Fortran. In particular, the cuBLAS library uses 1-based indexing and Fortran-style column-major storage for multidimensional data to simplify interfacing to Fortran applications. Unfortunately, Fortran-to-C calling conventions are not standardized and differ by platform and toolchain. In particular, differences may exist in the following areas:

- symbol names (capitalization, name decoration)
- argument passing (by value or reference)
- passing of string arguments (length information)
- passing of pointer arguments (size of the pointer)
- returning floating-point or compound data types (for example single-precision or complex data types)

To provide maximum flexibility in addressing those differences, the cuBLAS Fortran interface is provided in the form of wrapper functions and is part of the Toolkit delivery. The C source code of those wrapper functions is located in the **`src`** directory and provided in two different forms:

- the thunking wrapper interface located in the file fortran_thunking.c
- the direct wrapper interface located in the file fortran.c

The code of one of those 2 files needs to be compiled into an application for it to call the cuBLAS API functions. Providing source code allows users to make any changes necessary for a particular platform and toolchain.

The code in those two C files has been used to demonstrate interoperability with the compilers g77 3.2.3 and g95 0.91 on 32-bit Linux, g77 3.4.5 and g95 0.91 on 64-bit Linux, Intel Fortran 9.0 and Intel Fortran 10.0 on 32-bit and 64-bit Microsoft Windows XP, and g77 3.4.0 and g95 0.92 on Mac OS X.

Note that for g77, use of the compiler flag -fno-second-underscore is required to use these wrappers as provided. Also, the use of the default calling conventions with regard to argument and return value passing is expected. Using the flag -fno-f2c changes the default calling convention with respect to these two items.

The thunking wrappers allow interfacing to existing Fortran applications without any changes to the application. During each call, the wrappers allocate GPU memory, copy source data from CPU memory space to GPU memory space, call cuBLAS, and finally copy back the results to CPU memory space and deallocate the GPU memory. As this process causes very significant call overhead, these wrappers are intended for light testing, not for production code. To use the thunking wrappers, the application needs to be compiled with the file fortran_thunking.c

The direct wrappers, intended for production code, substitute device pointers for vector and matrix arguments in all BLAS functions. To use these interfaces, existing applications need to be modified slightly to allocate and deallocate data structures in GPU memory space (using cuBLAS_ALLOC and cuBLAS_FREE) and to copy data between GPU and CPU memory spaces (using cuBLAS_SET_VECTOR, cuBLAS_GET_VECTOR, cuBLAS_SET_MATRIX, and cuBLAS_GET_MATRIX). The sample wrappers provided in fortran.c map device pointers to the OS-dependent type size_t, which is 32-bit wide on 32-bit platforms and 64-bit wide on a 64-bit platforms.

One approach to deal with index arithmetic on device pointers in Fortran code is to use C-style macros, and use the C preprocessor to expand these, as shown in the example below. On Linux and Mac OS X, one way of pre-processing is to use the option '-E -x f77-cpp-input' when using g77 compiler, or simply the option '-cpp' when using g95 or

gfortran. On Windows platforms with Microsoft Visual C/C++, using 'cl -EP' achieves similar results.

```
! Example B.1. Fortran 77 Application Executing on the Host
! ----------------------------------------------------------
    subroutine modify ( m, ldm, n, p, q, alpha, beta )
    implicit none
    integer ldm, n, p, q
    real*4 m (ldm, *) , alpha , beta
    external cublas_sscal
    call cublas_sscal (n-p+1, alpha , m(p,q), ldm)
    call cublas_sscal (ldm-p+1, beta, m(p,q), 1)
    return
    end

    program matrixmod
    implicit none
    integer M,N
    parameter (M=6, N=5)
    real*4 a(M,N)
    integer i, j
    external cublas_init
    external cublas_shutdown

    do j = 1, N
        do i = 1, M
            a(i, j) = (i-1)*M + j
        enddo
    enddo
    call cublas_init
    call modify ( a, M, N, 2, 3, 16.0, 12.0 )
    call cublas_shutdown
    do j = 1 , N
        do i = 1 , M
            write(*,"(F7.0$)") a(i,j)
        enddo
        write (*,*) ""
    enddo
    stop
    end
```

When traditional fixed-form Fortran 77 code is ported to use the cuBLAS library, line length often increases when the BLAS calls are exchanged for cuBLAS calls. Longer function names and possible macro expansion are contributing factors. Inadvertently exceeding the maximum line length can lead to run-time errors that are difficult to find, so care should be taken not to exceed the 72-column limit if fixed form is retained.

The examples in this chapter show a small application implemented in Fortran 77 on the host and the same application with the non-thunking wrappers after it has been ported to use the cuBLAS library.

The second example should be compiled with ARCH_64 defined as 1 on 64-bit OS system and as 0 on 32-bit OS system. For example for g95 or gfortran, this can be done directly on the command line by using the option '-cpp -DARCH_64=1'.

```
! Example B.2. Same Application Using Non-thunking cuBLAS Calls
!-----------------------------------------------------------------
#define IDX2F (i,j,ld) ((((j)-1)*(ld))+((i)-1))
    subroutine modify ( devPtrM, ldm, n, p, q, alpha, beta )
    implicit none
    integer sizeof_real
    parameter (sizeof_real=4)
    integer ldm, n, p, q
#if ARCH_64
    integer*8 devPtrM
#else
    integer*4 devPtrM
#endif
    real*4 alpha, beta
    call cublas_sscal ( n-p+1, alpha,
    1                   devPtrM+IDX2F(p, q, ldm)*sizeof_real,
    2                   ldm)
    call cublas_sscal(ldm-p+1, beta,
    1                 devPtrM+IDX2F(p, q, ldm)*sizeof_real,
    2                 1)
    return
    end
    program matrixmod
    implicit none
    integer M,N,sizeof_real
#if ARCH_64
    integer*8 devPtrA
#else
    integer*4 devPtrA
#endif
    parameter(M=6,N=5,sizeof_real=4)
    real*4 a(M,N)
    integer i,j,stat
    external cublas_init, cublas_set_matrix, cublas_get_matrix
    external cublas_shutdown, cublas_alloc
    integer cublas_alloc, cublas_set_matrix, cublas_get_matrix
    do j=1,N
        do i=1,M
            a(i,j)=(i-1)*M+j
        enddo
    enddo
    call cublas_init
    stat= cublas_alloc(M*N, sizeof_real, devPtrA)
    if (stat.NE.0) then
        write(*,*) "device memory allocation failed"
        call cublas_shutdown
        stop
    endif
    stat = cublas_set_matrix(M,N,sizeof_real,a,M,devPtrA,M)
    if (stat.NE.0) then
        call cublas_free( devPtrA )
        write(*,*) "data download failed"
        call cublas_shutdown
        stop
    endif
    call modify(devPtrA, M, N, 2, 3, 16.0, 12.0)
    stat = cublas_get_matrix(M, N, sizeof_real, devPtrA, M, a, M )
    if (stat.NE.0) then
    call cublas_free ( devPtrA )
    write(*,*) "data upload failed"
    call cublas_shutdown
    stop
    endif
    call cublas_free ( devPtrA )
    call cublas_shutdown
    do j = 1 , N
        do i = 1 , M
            write (*,"(F7.0$)") a(i,j)
        enddo
        write (*,*) ""
```

# Appendix C. ACKNOWLEDGEMENTS

NVIDIA would like to thank the following individuals and institutions for their contributions:

- Portions of the SGEMM, DGEMM, CGEMM and ZGEMM library routines were written by Vasily Volkov of the University of California.
- Portions of the SGEMM, DGEMM and ZGEMM library routines were written by Davide Barbieri of the University of Rome Tor Vergata.
- Portions of the DGEMM and SGEMM library routines optimized for Fermi architecture were developed by the University of Tennessee. Subsequently, several other routines that are optimized for the Fermi architecture have been derived from these initial DGEMM and SGEMM implementations.
- The substantial optimizations of the STRSV, DTRSV, CTRSV and ZTRSV library routines were developed by Jonathan Hogg of The Science and Technology Facilities Council (STFC). Subsequently, some optimizations of the STRSM, DTRSM, CTRSM and ZTRSM have been derived from these TRSV implementations.
- Substantial optimizations of the SYMV and HEMV library routines were developed by Ahmad Abdelfattah, David Keyes and Hatem Ltaief of King Abdullah University of Science and Technology (KAUST).
- Substantial optimizations of the TRMM and TRSM library routines were developed by Ali Charara, David Keyes and Hatem Ltaief of King Abdullah University of Science and Technology (KAUST).

## Notice



## Trademarks



## Copyright