

PG-00000-002_V3.0 February, 2010

CUBLAS Library PG-00000-002_V3.0

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

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

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

Copyright

© 2005–2010 by NVIDIA Corporation. All rights reserved.

CUBLAS Library PG-00000-002_V3.0

Portions of the SGEMM, DGEMM and ZGEMM library routines were written by Vasily Volkov and are subject to the Modified Berkeley Software Distribution License as follows:

Copyright (c) 2007-2009, Regents of the University of California

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the University of California, Berkeley nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CUBLAS Library PG-00000-002_V3.0

Portions of the SGEMM, DGEMM and ZGEMM library routines were written by Davide Barbieri and are subject to the Modified Berkeley Software Distribution License as follows:

Copyright (c) 2008-2009 Davide Barbieri @ University of Rome Tor Vergata.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3- The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table of Contents

1. The CUBLAS Library	1
CUBLAS Types	
CUBLAS Helper Functions	
Function cublasShutdown()	
Function cublasAlloc()	.10
Function cublasSetVector()	.11
Function cublasSetMatrix()	.13
2. BLAS1 Functions	
Single-Precision BLAS1 Functions	
Function cublasIsamax()	
Function cublasIsamin()	
Function cublasSasum()	
Function cublasSaxpy()	
Function cublasScopy()	
Function cublasSdot()	
Function cublasSnrm2()	
Function cublasSrot()	
Function cublasSrotg()	
Function cublasSrotm()	
Function cublasSrotmg()	
Function cublasSscal()	
Function cublasSswap()	
Single-Precision Complex BLAS1 Functions	.29
Function cublasCaxpy()	.30
Function cublasCcopy()	.31
Function cublasCdotc()	
Function cublasCdotu()	
Function cublasCrot()	

	Function cublasCrotg()	
	Function cublasCscal()	.36
	Function cublasCsrot()	.36
	Function cublasCsscal()	.37
	Function cublasCswap()	.38
	Function cublasIcamax()	
	Function cublasIcamin()	
	Function cublasScasum()	
	Function cublasScnrm2()	
	· ·	
Do	buble-Precision BLAS1 Functions	
	Function cublasIdamax()	
	Function cublasIdamin()	
	Function cublasDasum()	
	Function cublasDaxpy()	.46
	Function cublasDcopy()	.47
	Function cublasDdot()	
	Function cublasDnrm2()	.49
	Function cublasDrot()	
	Function cublasDrotg()	
	Function cublasDrotm()	
	Function cublasDrotmg()	
	Function cublasDscal()	
	Function cubiasDswap()	.5-
Do	buble-Precision Complex BLAS1 functions	.57
	Function cublasDzasum()	
	Function cublasDznrm2()	
	Function cublasIzamax()	
	Function cublasIzamin()	
	Function cublasZaxpy()	.61
	Function cublasZcopy()	.62
	Function cublasZdotc()	.63
	Function cublasZdotu()	.64
	Function cublasZdrot()	.65
	Function cublasZdscal()	
	Function cublasZrot()	
	Function cublasZrotg()	
	Function cublasZscal()	
	Function cublasZswap()	
	Tunotion addiases wap ()	.07
3.	Single-Precision BLAS2 Functions	.71
	ngle-Precision BLAS2 Functions	
SİI	ngle-Precision BLAS2 Functions	. /2
	Function cublasSgbmv()	
	Function cublasSgemv()	
	Function cublasSger()	.75

	i cubiasSsbmv()	
	ı cublasSspmv()	
	ı cublasSspr()	
Function	cublasSspr2()	80
	cublasSsymv()	
Function	ı cublasSsyr()	83
Function	cublasSsyr2()	84
	cublasStbmv()	
Function	cublasStbsv()	87
	ı cublasStpmv()	
	cublasStpsv()	
	ı cublasStrmv()	
Function	ı cublasStrsv()	93
Sinale-Pre	ision Complex BLAS2 Functions	95
Function	ı cublasCgbmv()	96
	ı cublasCgemv()	
	ı cublasCgerc()	
	ı cublasCgeru()	
	ı cublasChbmv()	
	ı cublasChemv()	
	ı cublasCher()	
	ı cublasCher2()	
	ı cublasChpmv()	
	ı cublasChpr()	
	ı cublasChpr2()	
Function	ı cublasCtbmv()	111
Function	ı cublasCtbsv()	113
Function	ı cublasCtpmv()	114
Function	ı cublasCtpsv()	116
Function	ı cublasCtrmv()	117
Function	ı cublasCtrsv()	119
4. Doubl	-Precision BLAS2 Functions	21
Double-Pr	cision BLAS2 Functions	22
	ı cublasDgbmv()	
	ı cublasDgemv()	
	ı cublasDger()	
	ı cublasDsbmv()	
	o cublasDspmv()	
	ı cublasDspr()	
	ı cublasDspr2()	
	ı cublasDsymv()	
	ı cublasDsyr()	
	ı cublasDsyr2()	
	ı cublasDtbmv()	

Function cublasDtpmv()	38
Function cublasDtpsv()	
Function cublasDtrmv()	
Function cublasDtrsv()	44
Double-Precision Complex BLAS2 functions	46
Function cublasZgbmv()	47
Function cublasZgemv()	49
Function cublasZgerc()	
Function cublasZgeru()	
Function cublasZhbmv()	
Function cublasZhemv()	
Function cublasZher()	
Function cublasZher2()	
Function cublasZhpmv()	
Function cublasZhpr()	
Function cublasZhpr2()	
Function cublasZtbmv()	
Function cublasZtbsv()	
Function cublasZtpmv()	
Function cublasZtpsv()	08
Function cublasZtrmv()	
Function cublasZtrsv()1	/ 1
E. DI AC2 Functions	72
5. BLAS3 Functions	
Single-Precision BLAS3 Functions	74
Single-Precision BLAS3 Functions	74 75
Single-Precision BLAS3 Functions	74 75 76
Single-Precision BLAS3 Functions	74 75 76
Single-Precision BLAS3 Functions1Function cublasSgemm()1Function cublasSsymm()1Function cublasSsyrk()1Function cublasSsyr2k()1	74 75 76 78 80
Single-Precision BLAS3 Functions 1 Function cublasSgemm() 1 Function cublasSsymm() 1 Function cublasSsyrk() 1 Function cublasSsyr2k() 1 Function cublasStrmm() 1	74 75 76 78 80 82
Single-Precision BLAS3 Functions1Function cublasSgemm()1Function cublasSsymm()1Function cublasSsyrk()1Function cublasSsyr2k()1	74 75 76 78 80 82
Single-Precision BLAS3 Functions 1 Function cublasSgemm() 1 Function cublasSsymm() 1 Function cublasSsyrk() 1 Function cublasSsyr2k() 1 Function cublasStrmm() 1 Function cublasStrsm() 1	74 75 76 78 80 82 84
Single-Precision BLAS3 Functions 1 Function cublasSgemm() 1 Function cublasSsymm() 1 Function cublasSsyrk() 1 Function cublasSsyr2k() 1 Function cublasStrmm() 1 Function cublasStrsm() 1 Single-Precision Complex BLAS3 Functions 1	74 75 76 78 80 82 84
Single-Precision BLAS3 Functions 1 Function cublasSgemm() 1 Function cublasSsymm() 1 Function cublasSsyrk() 1 Function cublasSsyr2k() 1 Function cublasStrmm() 1 Function cublasStrsm() 1 Single-Precision Complex BLAS3 Functions 1 Function cublasCgemm() 1	74 75 76 78 80 82 84 87 88
Single-Precision BLAS3 Functions 1 Function cublasSgemm() 1 Function cublasSsymm() 1 Function cublasSsyrk() 1 Function cublasSsyr2k() 1 Function cublasStrmm() 1 Function cublasStrsm() 1 Single-Precision Complex BLAS3 Functions 1 Function cublasCgemm() 1 Function cublasChemm() 1	74 75 76 78 80 82 84 87 88
Single-Precision BLAS3 Functions 1 Function cublasSgemm() 1 Function cublasSsymm() 1 Function cublasSsyrk() 1 Function cublasSsyr2k() 1 Function cublasStrmm() 1 Function cublasStrsm() 1 Single-Precision Complex BLAS3 Functions 1 Function cublasCgemm() 1 Function cublasChemm() 1 Function cublasChemk() 1 Function cublasCherk() 1	74 75 76 78 80 82 84 87 88 91
Single-Precision BLAS3 Functions 1 Function cublasSgemm() 1 Function cublasSsymm() 1 Function cublasSsyrk() 1 Function cublasSsyr2k() 1 Function cublasStrmm() 1 Function cublasStrsm() 1 Single-Precision Complex BLAS3 Functions 1 Function cublasCgemm() 1 Function cublasChemm() 1 Function cublasCherk() 1 Function cublasCherk() 1 Function cublasCher2k() 1	74 75 76 78 80 82 84 87 88 89 91 93
Single-Precision BLAS3 Functions 1 Function cublasSgemm() 1 Function cublasSsymm() 1 Function cublasSsyrk() 1 Function cublasSsyr2k() 1 Function cublasStrmm() 1 Function cublasStrsm() 1 Single-Precision Complex BLAS3 Functions. 1 Function cublasCgemm() 1 Function cublasChemm() 1 Function cublasCherk() 1 Function cublasCher2k() 1 Function cublasCsymm() 1	74 75 76 78 80 82 84 87 88 91 93 95
Single-Precision BLAS3 Functions1Function cublasSgemm()1Function cublasSsymm()1Function cublasSsyrk()1Function cublasSsyr2k()1Function cublasStrmm()1Function cublasStrsm()1Single-Precision Complex BLAS3 Functions1Function cublasCgemm()1Function cublasChemm()1Function cublasCherk()1Function cublasCher2k()1Function cublasCsymm()1Function cublasCsymm()1Function cublasCsymk()1Function cublasCsyrk()1	74 75 76 78 80 82 84 87 88 91 93 95 97
Single-Precision BLAS3 Functions1Function cublasSgemm()1Function cublasSsymm()1Function cublasSsyrk()1Function cublasSsyr2k()1Function cublasStrmm()1Function cublasStrsm()1Single-Precision Complex BLAS3 Functions1Function cublasCgemm()1Function cublasChemm()1Function cublasCherk()1Function cublasCher2k()1Function cublasCsymm()1Function cublasCsymm()1Function cublasCsyrk()1Function cublasCsyrk()1Function cublasCsyr2k()1	74 75 76 78 80 82 84 87 88 91 93 95 97
Single-Precision BLAS3 Functions1Function cublasSgemm()1Function cublasSsymm()1Function cublasSsyrk()1Function cublasSsyr2k()1Function cublasStrmm()1Function cublasStrsm()1Single-Precision Complex BLAS3 Functions1Function cublasCgemm()1Function cublasChemm()1Function cublasCherk()1Function cublasCher2k()1Function cublasCsymm()1Function cublasCsyrk()1Function cublasCsyrk()1Function cublasCsyr2k()1Function cublasCtrmm()2	74 75 76 78 80 82 84 87 88 91 93 95 97 99
Single-Precision BLAS3 Functions1Function cublasSgemm()1Function cublasSsymm()1Function cublasSsyrk()1Function cublasSsyr2k()1Function cublasStrmm()1Function cublasStrsm()1Single-Precision Complex BLAS3 Functions1Function cublasCgemm()1Function cublasChemm()1Function cublasChemk()1Function cublasCherk()1Function cublasCher2k()1Function cublasCsymm()1Function cublasCsyrk()1Function cublasCsyr2k()1Function cublasCtrmm()2Function cublasCtrsm()2Function cublasCtrsm()2	74 75 76 78 80 82 84 87 88 91 93 95 97 99 01
Single-Precision BLAS3 Functions1Function cublasSgemm()1Function cublasSsymm()1Function cublasSsyrk()1Function cublasSsyr2k()1Function cublasStrmm()1Function cublasStrsm()1Single-Precision Complex BLAS3 Functions1Function cublasCgemm()1Function cublasChemm()1Function cublasCherk()1Function cublasCher2k()1Function cublasCsymm()1Function cublasCsyrk()1Function cublasCsyrk()1Function cublasCsyr2k()1Function cublasCtrmm()2	74 75 76 78 80 82 84 87 88 91 93 95 97 99 901

Function cublasZtrmm()	. 234
Function cublasZsymm()	
Function cublasZher2k()	
Function cublasZgemm()	. 220
ouble-Precision Complex BLAS3 Functions	. 219
Function cublasDtrmm() Function cublasDtrsm()	. 214
Function cublasDsymm()	. 210
C	Function cublasDsyrk() Function cublasDsyr2k() Function cublasDtrmm() Function cublasDtrsm() buble-Precision Complex BLAS3 Functions Function cublasZgemm() Function cublasZhemm() Function cublasZherk() Function cublasZher2k() Function cublasZsymm() Function cublasZsyrk() Function cublasZsyrk() Function cublasZsyr2k()

CHAPTER

1

The CUBLAS Library

CUBLAS is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA CUDA runtime. It allows access to the computational resources of NVIDIA GPUs. The library is self-contained at the API level, that is, no direct interaction with the CUDA driver is necessary. CUBLAS attaches to a single GPU and does not auto-parallelize across multiple GPUs.

The basic model by which applications use the CUBLAS library is to create matrix and vector objects in GPU memory space, fill them with data, call a sequence of CUBLAS functions, and, finally, upload the results from GPU memory space back to the host. To accomplish this, CUBLAS provides helper functions for creating and destroying objects in GPU space, and for writing data to and retrieving data from these objects.

For maximum compatibility with existing Fortran environments, CUBLAS uses column-major storage and 1-based indexing. Since C and C++ use row-major storage, applications cannot 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, 1d refers to the leading dimension of the matrix as allocated, which in the case of column-major storage is the number of rows. For natively written C and C++ code, one would most likely chose 0-based indexing, in which case the indexing macro becomes

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

Please refer to the code examples at the end of this section, which show a tiny application implemented in Fortran on the host (Example 1. "Fortran 77 Application Executing on the Host") and show versions of the application written in C using CUBLAS for the indexing styles described above (Example 2. "Application Using C and CUBLAS: 1-based Indexing" and Example 3. "Application Using C and CUBLAS: 0-based Indexing").

Because the CUBLAS core functions (as opposed to the helper functions) do not return error status directly (for reasons of compatibility with existing BLAS libraries), CUBLAS provides a separate function to aid in debugging that retrieves the last recorded error.

The interface to the CUBLAS library is the header file cublas.h. Applications using CUBLAS need to link against the DSO cublas.so (Linux), the DLL cublas.dll (Windows), or the dynamic library cublas.dylib (Mac OS X) when building for the device, and against the DSO cublasemu.so (Linux), the DLL cublasemu.dll (Windows), or the dynamic library cublasemu.dylib (Mac OS X) when building for device emulation.

Following these three examples, the remainder of this chapter discusses "CUBLAS Types" on page 8 and "CUBLAS Helper Functions" on page 9.

The CUBLAS Library

Example 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 sscal
call sscal (n-p+1, alpha, m(p,q), ldm)
call 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
do j = 1, N
  do i = 1, M
    a(i,j) = (i-1) * M + j
  enddo
enddo
call modify (a, M, N, 2, 3, 16.0, 12.0)
do j = 1, N
  do i = 1, M
    write(*,"(F7.0\$)") a(i,j)
  enddo
  write (*,*) ""
enddo
stop
end
```

Example 2. Application Using C and CUBLAS: 1-based Indexing

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cublas.h"
#define IDX2F(i,j,ld) ((((j)-1)*(ld))+((i)-1))
void modify (float *m, int ldm, int n, int p, int q, float alpha,
             float beta)
    \verb|cublasSscal| (n-p+1, alpha, &m[IDX2F(p,q,ldm)], ldm); \\
    cublasSscal (ldm-p+1, beta, &m[IDX2F(p,q,ldm)], 1);
}
#define M 6
#define N 5
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 \le N; j++) {
        for (i = 1; i <= M; i++) {
            a[IDX2F(i,j,M)] = (i-1) * M + j;
    cublasInit();
    stat = cublasAlloc (M*N, sizeof(*a), (void**)&devPtrA);
```

CHAPTER 1 The CUBLAS Library

Example 2. Application Using C and CUBLAS: 1-based Indexing (continued)

```
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 \le N; j++) {
    for (i = 1; i <= M; i++) {
        printf ("%7.0f", a[IDX2F(i,j,M)]);
   printf ("\n");
return EXIT_SUCCESS;
```

Example 3. Application Using C and CUBLAS: 0-based Indexing

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cublas.h"
#define IDX2C(i,j,ld) (((j)*(ld))+(i))
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);
}
#define M 6
#define N 5
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)] = i * M + j + 1;
    cublasInit();
    stat = cublasAlloc (M*N, sizeof(*a), (void**)&devPtrA);
    if (stat != CUBLAS_STATUS_SUCCESS) {
```

Chapter 1 The CUBLAS Library

Example 3. Application Using C and CUBLAS: 0-based Indexing (continued)

```
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");
return EXIT_SUCCESS;
```

CUBLAS Types

The only CUBLAS type is cublasStatus.

Type cublasStatus

The type cublasstatus is used for function status returns. CUBLAS helper functions return status directly, while the status of CUBLAS core functions can be retrieved via cublasGetError(). Currently, the following values are defined:

cublasStatus Values

CUBLAS_STATUS_SUCCESS	operation completed successfully
CUBLAS_STATUS_NOT_INITIALIZED	CUBLAS library not initialized
CUBLAS_STATUS_ALLOC_FAILED	resource allocation failed
CUBLAS_STATUS_INVALID_VALUE	unsupported numerical value was passed to function
CUBLAS_STATUS_ARCH_MISMATCH	function requires an architectural feature absent from the architecture of the device
CUBLAS_STATUS_MAPPING_ERROR	access to GPU memory space failed
CUBLAS_STATUS_EXECUTION_FAILED	GPU program failed to execute
CUBLAS_STATUS_INTERNAL_ERROR	an internal CUBLAS operation failed

CUBLAS Helper Functions

The following are the CUBLAS helper functions:

- □ "Function cublasInit()" on page 9
- □ "Function cublasShutdown()" on page 10
- □ "Function cublasGetError()" on page 10
- □ "Function cublasAlloc()" on page 10
- □ "Function cublasFree()" on page 11
- □ "Function cublasSetVector()" on page 11
- □ "Function cublasGetVector()" on page 12
- □ "Function cublasSetMatrix()" on page 13
- □ "Function cublasGetMatrix()" on page 13

Function cublasInit()

cublasStatus
cublasInit (void)

initializes the CUBLAS library and must be called before any other CUBLAS API function is invoked. It allocates hardware resources necessary for accessing the GPU. It attaches CUBLAS to whatever GPU is currently bound to the host thread from which it was invoked.

Return Values

CUBLAS_STATUS_ALLOC_FAILED if resources could not be allocated

CUBLAS_STATUS_SUCCESS if CUBLAS library initialized successfully

Function cublasShutdown()

cublasStatus cublasShutdown (void)

releases CPU-side resources used by the CUBLAS library. The release of GPU-side resources may be deferred until the application shuts down.

Return Values

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized
CUBLAS library shut down successfully

Function cublasGetError()

cublasStatus
cublasGetError (void)

returns the last error that occurred on invocation of any of the CUBLAS core functions. While the CUBLAS helper functions return status directly, the CUBLAS core functions do not, improving compatibility with those existing environments that do not expect BLAS functions to return status. Reading the error status via cublasGetError() resets the internal error state to CUBLAS_STATUS_SUCCESS.

Function cublasAlloc()

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

creates an object in GPU memory space capable of holding an array of n elements, where each element requires elemSize bytes of storage. If the function call is successful, a pointer to the object in GPU memory space is placed in devicePtr. Note that this is a device pointer that cannot be dereferenced in host code. Function cublasAlloc() is a wrapper around cudaMalloc(). Device pointers returned by

cublasAlloc() can therefore be passed to any CUDA device kernels, not just CUBLAS functions.

Return Values

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	$if n \le 0 or elemSize \le 0$
CUBLAS_STATUS_ALLOC_FAILED	if the object could not be allocated due to lack of resources.
CUBLAS_STATUS_SUCCESS	if storage was successfully allocated

Function cublasFree()

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

destroys the object in GPU memory space referenced by devicePtr.

Return Values

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INTERNAL_ERROR	if the object could not be deallocated
CUBLAS_STATUS_SUCCESS	if object was deallocated successfully

Function cublasSetVector()

copies n elements from a vector x in CPU memory space to a vector y in GPU memory space. Elements in both vectors are assumed to have a size of elemsize bytes. Storage spacing between consecutive elements is incx for the source vector x and incy for the destination vector y. In general, y points to an object, or part of an object, allocated via cublasAlloc(). Column-major format for two-dimensional matrices is assumed throughout CUBLAS. If the vector is part of a matrix, a vector increment equal to 1 accesses a (partial) column of the matrix.

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

Return Values

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx, incy, or elemSize <= 0
CUBLAS_STATUS_MAPPING_ERROR	if error accessing GPU memory
CUBLAS_STATUS_SUCCESS	if operation completed successfully

Function cublasGetVector()

```
cublasStatus
```

copies n elements from a vector x in GPU memory space to a vector y in CPU memory space. Elements in both vectors are assumed to have a size of elemSize bytes. Storage spacing between consecutive elements is incx for the source vector x and incy for the destination vector y. In general, x points to an object, or part of an object, allocated via **cublasAlloc()**. Column-major format for two-dimensional matrices is assumed throughout CUBLAS. If the vector is part of a matrix, a vector increment equal to 1 accesses a (partial) column of the matrix. Similarly, using an increment equal to the leading dimension of the matrix accesses a (partial) row.

Return Values

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx, incy, or elemSize <= 0
CUBLAS_STATUS_MAPPING_ERROR	if error accessing GPU memory
CUBLAS_STATUS_SUCCESS	if operation completed successfully

Function cublasSetMatrix()

copies a tile of rowsxcols elements from a matrix A in CPU memory space to a matrix B in GPU memory space. Each element requires storage of elemSize bytes. Both matrices are assumed to be stored in column-major format, with the leading dimension (that is, the number of rows) of source matrix A provided in lda, and the leading dimension of destination matrix B provided in ldb. 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 Values

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	$if \ {\tt rows} \ {\tt or} \ {\tt cols} \ {\tt <0}; \ {\tt or} \ {\tt elemSize},$
	lda, Or ldb <= 0
CUBLAS_STATUS_MAPPING_ERROR	if error accessing GPU memory
CUBLAS_STATUS_SUCCESS	if operation completed successfully

Function cublasGetMatrix()

int ldb)

copies a tile of rowsxcols elements from a matrix A in GPU memory space to a matrix B in CPU memory space. Each element requires

storage of elemSize bytes. Both matrices are assumed to be stored in

column-major format, with the leading dimension (that is, the number of rows) of source matrix A provided in lda, and the leading dimension of destination matrix B provided in ldb. 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 Values

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if rows Or cols < 0; Or elemSize,
	lda, or ldb <= 0
CUBLAS_STATUS_MAPPING_ERROR	if error accessing GPU memory
CUBLAS_STATUS_SUCCESS	if operation completed successfully

CHAPTER

2

BLAS1 Functions

Level 1 Basic Linear Algebra Subprograms (BLAS1) are functions that perform scalar, vector, and vector-vector operations. The CUBLAS BLAS1 implementation is described in these sections:

- □ "Single-Precision BLAS1 Functions" on page 16
- □ "Single-Precision Complex BLAS1 Functions" on page 29
- □ "Double-Precision BLAS1 Functions" on page 43
- □ "Double-Precision Complex BLAS1 functions" on page 57

Single-Precision BLAS1 Functions

The single-precision BLAS1 functions are as follows:

- □ "Function cublasIsamax()" on page 17
- □ "Function cublasIsamin()" on page 17
- □ "Function cublasSasum()" on page 18
- □ "Function cublasSaxpy()" on page 19
- □ "Function cublasScopy()" on page 20
- □ "Function cublasSdot()" on page 21
- □ "Function cublasSnrm2()" on page 22
- □ "Function cublasSrot()" on page 22
- □ "Function cublasSrotg()" on page 23
- □ "Function cublasSrotm()" on page 24
- □ "Function cublasSrotmg()" on page 26
- □ "Function cublasSscal()" on page 27
- □ "Function cublasSswap()" on page 28

CHAPTER 2 BLAS1 Functions

Function cublasIsamax()

int

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

finds the smallest index of the maximum magnitude element of single-precision vector \mathbf{x} ; that is, the result is the first \mathbf{i} , \mathbf{i} = 0 to \mathbf{n} -1, that maximizes abs($\mathbf{x}[1+\mathbf{i}*incx]$). The result reflects 1-based indexing for compatibility with Fortran.

Input

•	
n	number of elements in input vector
x	single-precision vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the smallest index (returns zero if n <= 0 or incx <= 0)

Reference: http://www.netlib.org/blas/isamax.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasIsamin()

int

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

finds the smallest index of the minimum magnitude element of single-precision vector x; that is, the result is the first i, i = 0 to n-1, that minimizes abs(x[1+i*incx]). The result reflects 1-based indexing for compatibility with Fortran.

Input

n	number of elements in input vector
x	single-precision vector with n elements
incx	storage spacing between elements of x

Output

returns the smallest index (returns zero if n <= 0 or incx <= 0)

Reference: http://www.netlib.org/scilib/blass.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasSasum()

float

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

computes the sum of the absolute values of the elements of single-precision vector \mathbf{x} ; that is, the result is the sum from $\mathbf{i} = 0$ to $\mathbf{n}-1$ of $abs(\mathbf{x}[1+\mathbf{i}*incx])$.

Input

•	
n	number of elements in input vector
x	single-precision vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the single-precision sum of absolute values (returns zero if n <= 0 or incx <= 0, or if an error occurred)

Reference: http://www.netlib.org/blas/sasum.f

Error status for this function can be retrieved via cublasGetError().

Frror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

CHAPTER 2 BLAS1 Functions

Function cublasSaxpy()

void

multiplies single-precision vector \mathbf{x} by single-precision scalar alpha and adds the result to single-precision vector \mathbf{y} ; that is, it overwrites single-precision \mathbf{y} with single-precision alpha * $\mathbf{x} + \mathbf{y}$.

For i = 0 to n-1, it replaces

```
y[ly+i*incy] with alpha * x[lx+i*incx]+y[ly+i*incy],
```

where

```
1x = 0 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
alpha	single-precision scalar multiplier
x	single-precision vector with n elements
incx	storage spacing between elements of x
У	single-precision vector with n elements
incy	storage spacing between elements of $_{\mathtt{Y}}$

Output

Reference: http://www.netlib.org/blas/saxpy.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasScopy()

void

copies the single-precision vector \mathbf{x} to the single-precision vector \mathbf{y} . For i = 0 to n-1, it copies

```
x[lx+i*incx] to y[ly+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1 - n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
x	single-precision vector with n elements
incx	storage spacing between elements of x
У	single-precision vector with n elements
incy	storage spacing between elements of y

Output

У	contains single-precision vector x

Reference: http://www.netlib.org/blas/scopy.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

CHAPTER 2 BLAS1 Functions

Function cublasSdot()

float

computes the dot product of two single-precision vectors. It returns the dot product of the single-precision vectors x and y if successful, and 0.0f otherwise. It computes the sum for i = 0 to n-1 of

```
x[lx+i*incx]*y[ly+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar way using incy.

Input

1-	
n	number of elements in input vectors
x	single-precision vector with n elements
incx	storage spacing between elements of x
У	single-precision vector with n elements
incy	storage spacing between elements of y

Output

returns single-precision dot product (returns zero if n <= 0)

Reference: http://www.netlib.org/blas/sdot.f

Error status for this function can be retrieved via **cublasGetError()**.

Frror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to execute on GPU

Function cublasSnrm2()

float

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

computes the Euclidean norm of the single-precision n-vector x (with storage increment incx). This code uses a multiphase model of accumulation to avoid intermediate underflow and overflow.

Input

n	number of elements in input vector
x	single-precision vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the Euclidian norm (returns zero if n <= 0, incx <= 0, or if an error occurred)

Reference: http://www.netlib.org/blas/snrm2.f

Reference: http://www.netlib.org/slatec/lin/snrm2.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasSrot()

void

multiplies a 2×2 matrix
$$\begin{bmatrix} sc ss \\ -ss sc \end{bmatrix}$$
 with the 2×n matrix $\begin{bmatrix} x^T \\ y^T \end{bmatrix}$.

The elements of x are in x[lx+i*incx], i = 0 to n-1, where

$$1x = 1$$
 if incx >= 0, else
 $1x = 1 + (1-n) * incx;$

CHAPTER 2 BLAS1 Functions

y is treated similarly using ly and incy.

Input

n	number of elements in input vectors	
x	single-precision vector with ${\tt n}$ elements	
incx	storage spacing between elements of x	
У	single-precision vector with ${\tt n}$ elements	
incy	storage spacing between elements of $_{ m Y}$	
sc	element of rotation matrix	
ss	element of rotation matrix	
Output		
х	rotated vector x (unchanged if $n \le 0$)	
У	rotated vector y (unchanged if n <= 0)	

Reference: http://www.netlib.org/blas/srot.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasSrotg()

```
void
```

constructs the Givens transformation

$$G = \begin{bmatrix} sc ss \\ -ss sc \end{bmatrix}, \quad sc^2 + ss^2 = 1$$

which zeros the second entry of the 2-vector $\begin{bmatrix} sa & sb \end{bmatrix}^T$.

The quantity $r = \pm \sqrt{sa^2 + sb^2}$ overwrites sa in storage. The value of sb is overwritten by a value z which allows sc and ss to be recovered by the following algorithm:

```
if z = 1 set sc = 0.0 and ss = 1.0.

if abs(z) < 1 set sc = \sqrt{1-z^2} and ss = z.

if abs(z) > 1 set sc = 1/z and ss = \sqrt{1-sc^2}.
```

The function **cublasSrot** (n, x, incx, y, incy, sc, ss) normally is called next to apply the transformation to a $2 \times n$ matrix. Note that this function is provided for completeness and is run exclusively on the host.

Input

sa	single-precision scalar
sb	single-precision scalar
Output	
sa	single-precision r
sb	single-precision z
sc	single-precision result
SS	single-precision result

Reference: http://www.netlib.org/blas/srotg.f

This function does not set any error status.

Function cublasSrotm()

applies the modified Givens transformation, h, to the $2 \times n$ matrix $\begin{bmatrix} x^T \\ y^T \end{bmatrix}$

The elements of x are in x[lx+i*incx], i = 0 to n-1, where

$$1x = 1$$
 if incx >= 0, else
 $1x = 1 + (1-n) * incx;$

CHAPTER 2 BLAS1 Functions

y is treated similarly using ly and incy.

With sparam[0] = sflag, h has one of the following forms:

Input

n	number of elements in input vectors.
	•
X	single-precision vector with n elements.
incx	storage spacing between elements of x.
У	single-precision vector with ${\tt n}$ elements.
incy	storage spacing between elements of y.
sparam	5-element vector. sparam[0] is sflag described above. sparam[1] through sparam[4] contain the 2×2 rotation matrix h: sparam[1] contains sh00, sparam[2] contains sh10, sparam[3] contains sh01, and sparam[4] contains sh11.

Output

x	rotated vector x (unchanged if n <= 0)
У	rotated vector y (unchanged if n <= 0)

Reference: http://www.netlib.org/blas/srotm.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasSrotmg()

void

constructs the modified Givens transformation matrix h which zeros the second component of the 2-vector $(\sqrt{\text{sd1}}*\text{sx1}, \sqrt{\text{sd2}}*\text{sy1})^T$.

With sparam[0] = sflag, h has one of the following forms:

sparam[1] through sparam[4] contain sh00, sh10, sh01, and sh11, respectively. Values of 1.0f, -1.0f, or 0.0f implied by the value of sflag are not stored in sparam. Note that this function is provided for completeness and is run exclusively on the host.

Input

sd1	single-precision scalar.	
sd2	single-precision scalar.	
sx1	single-precision scalar.	
sy1	single-precision scalar.	

Output

sd1	changed to represent the effect of the transformation.
sd2	changed to represent the effect of the transformation.
sx1	changed to represent the effect of the transformation.
sparam	5-element vector. sparam[0] is sflag described above. sparam[1] through sparam[4] contain the 2×2 rotation matrix h: sparam[1] contains sh00, sparam[2] contains sh10, sparam[3] contains sh01, and sparam[4] contains sh11.

Reference: http://www.netlib.org/blas/srotmg.f

This function does not set any error status.

Function cublasSscal()

```
void
```

cublasSscal (int n, float alpha, float *x, int incx)

replaces single-precision vector $\mathbf x$ with single-precision alpha * $\mathbf x$. For i = 0 to n-1, it replaces

```
x[lx+i*incx] with alpha * x[lx+i*incx],
```

where

```
lx = 1 if incx >= 0, else
```

$$lx = 1 + (1 - n) * incx.$$

Input

-	
n	number of elements in input vector
alpha	single-precision scalar multiplier
х	single-precision vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

х	single-precision result (unchanged if n <= 0 or incx <= 0)
	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Reference: http://www.netlib.org/blas/sscal.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasSswap()

void

interchanges single-precision vector \mathbf{x} with single-precision vector \mathbf{y} . For $\mathbf{i} = 0$ to $\mathbf{n} - 1$, it interchanges

```
x[lx+i*incx] with y[ly+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1 - n) * incx;
```

ly is defined in a similar manner using incy.

Input

n	number of elements in input vectors
x	single-precision vector with n elements
incx	storage spacing between elements of x
У	single-precision vector with n elements
incy	storage spacing between elements of y

Output

х	single-precision vector y (unchanged from input if n <= 0)
У	single-precision vector x (unchanged from input if $n \le 0$)

Reference: http://www.netlib.org/blas/sswap.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Single-Precision Complex BLAS1 Functions

The single-precision complex BLAS1 functions are as follows:

- □ "Function cublasCaxpy()" on page 30
- □ "Function cublasCcopy()" on page 31
- □ "Function cublasCdotc()" on page 32
- □ "Function cublasCdotu()" on page 33
- □ "Function cublasCrot()" on page 34
- □ "Function cublasCrotg()" on page 35
- □ "Function cublasCscal()" on page 36
- □ "Function cublasCsrot()" on page 36
- □ "Function cublasCsscal()" on page 37
- □ "Function cublasCswap()" on page 38
- □ "Function cublasIcamax()" on page 39
- □ "Function cublasIcamin()" on page 40
- □ "Function cublasScasum()" on page 40
- □ "Function cublasScnrm2()" on page 41

Function cublasCaxpy()

void

multiplies single-precision complex vector \mathbf{x} by single-precision complex scalar alpha and adds the result to single-precision complex vector \mathbf{y} ; that is, it overwrites single-precision complex \mathbf{y} with single-precision complex alpha * $\mathbf{x} + \mathbf{y}$.

```
For i = 0 to n-1, it replaces
```

```
y[ly+i*incy] with alpha * x[lx+i*incx]+y[ly+i*incy],
```

where

```
1x = 0 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
alpha	single-precision complex scalar multiplier
x	single-precision complex vector with n elements
incx	storage spacing between elements of x
У	single-precision complex vector with n elements
incy	storage spacing between elements of $_{\mathtt{Y}}$

Output

```
y single-precision complex result (unchanged if n \le 0)
```

Reference: http://www.netlib.org/blas/caxpy.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasCcopy()

void

```
cublasCcopy (int n, const cuComplex *x, int incx,
             cuComplex *y, int incy)
```

copies the single-precision complex vector x to the single-precision complex vector y.

```
For i = 0 to n-1, it copies
```

```
x[lx+i*incx] to y[ly+i*incy],
```

where

```
1x = 1 \text{ if incx} >= 0, \text{ else}
1x = 1 + (1 - n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
x	single-precision complex vector with n elements
incx	storage spacing between elements of x
У	single-precision complex vector with n elements
incy	storage spacing between elements of _Y

Output

contains single-precision complex vector x

Reference: http://www.netlib.org/blas/ccopy.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasCdotc()

cuComplex

computes the dot product of two single-precision complex vectors, the first of which is conjugated. It returns the dot product of the complex conjugate of single- precision complex vector \mathbf{x} and the single-precision complex vector \mathbf{y} if successful, and complex zero otherwise. For $\mathbf{i} = 0$ to $\mathbf{n} - 1$, it sums the products

```
\overline{x[1x+i*incx]} * y[1y+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar way using incy.

Input

•	
n	number of elements in input vectors
x	single-precision complex vector with n elements
incx	storage spacing between elements of x
У	single-precision complex vector with n elements
incy	storage spacing between elements of $_{\mathtt{Y}}$

Output

returns single-precision complex dot product (zero if n <= 0)

Reference: http://www.netlib.org/blas/cdotc.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_ALLOC_FAILED if function could not allocate reduction buffer

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasCdotu()

cuComplex

computes the dot product of two single-precision complex vectors. It returns the dot product of the single-precision complex vectors \mathbf{x} and \mathbf{y} if successful, and complex zero otherwise. For $\mathbf{i} = 0$ to $\mathbf{n} - 1$, it sums the products

```
x[lx+i*incx]*y[ly+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
х	single-precision complex vector with n elements
incx	storage spacing between elements of x
У	single-precision complex vector with n elements
incy	storage spacing between elements of y

Output

returns single-precision complex dot product (returns zero if $n \le 0$)

Reference: http://www.netlib.org/blas/cdotu.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_ALLOC_FAILED if function could not allocate reduction buffer

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasCrot()

void

multiplies a 2×2 matrix
$$\begin{bmatrix} sccs \\ -csssc \end{bmatrix}$$
 with the 2×n matrix $\begin{bmatrix} x^T \\ y^T \end{bmatrix}$.

The elements of x are in x[lx+i*incx], i = 0 to n-1, where

$$1x = 1$$
 if incx >= 0, else
 $1x = 1 + (1-n) * incx;$

y is treated similarly using ly and incy.

Input

n	number of elements in input vectors
x	single-precision complex vector with ${\tt n}$ elements
incx	storage spacing between elements of x
У	single-precision complex vector with ${\tt n}$ elements
incy	storage spacing between elements of $_{ m Y}$
sc	single-precision cosine component of rotation matrix
CS	single-precision complex sine component of rotation matrix

Output

х	rotated vector x (unchanged if $n \le 0$)
У	rotated vector y (unchanged if $n \le 0$)

Reference: http://netlib.org/lapack/explore-html/crot.f.html

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasCrotg()

void

constructs the complex Givens transformation

$$G = \begin{bmatrix} sc cs \\ -cs sc \end{bmatrix}, sc*sc+cs*cs = 1$$

which zeros the second entry of the complex 2-vector $\begin{bmatrix} ca & cb \end{bmatrix}^T$.

The quantity ca/|ca|*||ca, cb|| overwrites ca in storage. In this case,

$$||ca, cb|| = scale*\sqrt{|ca/scale|^2 + |cb/scale|^2}$$
, where $scale = |ca| + |cb|$.

The function **cublasCrot** (n, x, incx, y, incy, sc, cs) normally is called next to apply the transformation to a $2 \times n$ matrix. Note that this function is provided for completeness and is run exclusively on the host.

Input

CS

ca	single-precision complex scalar
cb	single-precision complex scalar
Output	
ca	single-precision complex ca/ ca * ca, cb
sc	single-precision cosine component of rotation matrix

single-precision complex sine component of rotation matrix

Reference: http://www.netlib.org/blas/crotg.f

This function does not set any error status.

Function cublasCscal()

void

replaces single-precision complex vector \mathbf{x} with single-precision complex alpha * \mathbf{x} .

For i = 0 to n-1, it replaces

$$x[1x+i*incx]$$
 with alpha * $x[1x+i*incx]$,

where

```
1x = 1 \text{ if incx} >= 0, \text{ else}
```

$$lx = 1 + (1 - n) * incx.$$

Input

n	number of elements in input vector
alpha	single-precision complex scalar multiplier
x	single-precision complex vector with n elements
incx	storage spacing between elements of x

Output

x single-precision complex result (unchanged if $n \le 0$ or incx ≤ 0)

Reference: http://www.netlib.org/blas/cscal.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasCsrot()

biov

multiplies a 2×2 matrix
$$\begin{bmatrix} scss \\ -sssc \end{bmatrix}$$
 with the 2×n matrix $\begin{bmatrix} x^T \\ y^T \end{bmatrix}$.

The elements of x are in x[lx+i*incx], i = 0 to n-1, where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

y is treated similarly using ly and incy.

Input

n	number of elements in input vectors
x	single-precision complex vector with n elements
incx	storage spacing between elements of x
У	single-precision complex vector with n elements
incy	storage spacing between elements of $_{ m Y}$
sc	single-precision cosine component of rotation matrix
ss	single-precision sine component of rotation matrix

Output

х	rotated vector x (unchanged if n <= 0)
У	rotated vector y (unchanged if n <= 0)

Reference: http://www.netlib.org/blas/csrot.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasCsscal()

void

cublasCsscal (int n, float alpha, cuComplex *x, int incx) replaces single-precision complex vector x with single-precision complex alpha * x. For i = 0 to n-1, it replaces

```
x[lx+i*incx] with alpha * x[lx+i*incx],
```

where

```
1x = 1 \text{ if incx} >= 0, \text{ else}

1x = 1 + (1 - n) * \text{incx}.
```

Input		
n	number of elements in input vector	
alpha	single-precision scalar multiplier	
x	single-precision complex vector with n elements	
incx	storage spacing between elements of \mathbf{x}	
Output		
х	single-precision complex result (unchanged if n <= 0 or incx <= 0)	

Reference: http://www.netlib.org/blas/csscal.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasCswap()

void

interchanges the single-precision complex vector \mathbf{x} with the single-precision complex vector \mathbf{y} . For $\mathbf{i} = 0$ to $\mathbf{n} - 1$, it interchanges

```
x[lx+i*incx] with y[ly+i*incy],
```

where

```
1x = 1 \text{ if incx} >= 0, \text{ else}

1x = 1 + (1 - n) * \text{incx};
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
x	single-precision complex vector with ${\tt n}$ elements
incx	storage spacing between elements of x
У	single-precision complex vector with n elements
incy	storage spacing between elements of y

Output

х	single-precision complex vector y (unchanged from input if n <= 0)
У	single-precision complex vector x (unchanged from input if $n \le 0$)

Reference: http://www.netlib.org/blas/cswap.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasIcamax()

int

cublasIcamax (int n, const cuComplex *x, int incx)

finds the smallest index of the maximum magnitude element of single-precision complex vector \mathbf{x} ; that is, the result is the first \mathbf{i} , $\mathbf{i} = 0$ to $\mathbf{n} - 1$, that maximizes $\mathtt{abs}(\mathbf{x}[1+\mathbf{i}*\mathtt{incx}])$. The result reflects 1-based indexing for compatibility with Fortran.

Input

n	number of elements in input vector
x	single-precision complex vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the smallest index (returns zero if $n \le 0$ or $incx \le 0$)

Reference: http://www.netlib.org/blas/icamax.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

PG-00000-002_V3.0

Function cublasIcamin()

int

```
cublasIcamin (int n, const cuComplex *x, int incx)
```

finds the smallest index of the minimum magnitude element of single-precision complex vector x; that is, the result is the first i, i = 0 to n-1, that minimizes abs(x[1+i*incx]). The result reflects 1-based indexing for compatibility with Fortran.

Input

•	
n	number of elements in input vector
x	single-precision complex vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the smallest index (returns zero if n <= 0 or incx <= 0)

Reference: Analogous to http://www.netlib.org/blas/icamax.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasScasum()

float

```
cublasScasum (int n, const cuDouble *x, int incx)
```

takes the sum of the absolute values of a complex vector and returns a single-precision result. Note that this is not the L1 norm of the vector. The result is the sum from 0 to n-1 of

```
abs(real(x[lx+i*incx])) + abs(imag(x[lx+i*incx])),
```

where

```
1x = 1 if incx <= 0, else 1x = 1 + (1 - n) * incx.
```

In	put

n	number of elements in input vector
x	single-precision complex vector with ${\tt n}$ elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the single-precision sum of absolute values of real and imaginary parts (returns zero if n <= 0, incx <= 0, or if an error occurred)

Reference: http://www.netlib.org/blas/scasum.f

Error status for this function can be retrieved via cublasGetError().

Frror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasScnrm2()

float

cublasScnrm2 (int n, const cuComplex *x, int incx)

computes the Euclidean norm of single-precision complex n-vector x. This implementation uses simple scaling to avoid intermediate underflow and overflow.

Input

n	number of elements in input vector
x	single-precision complex vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the Euclidian norm (returns zero if n <= 0, incx <= 0, or if an error occurred)

Reference: http://www.netlib.org/blas/scnrm2.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS STATUS NOT INITIALIZED	if CUBLAS library was not initialized
COBLAS_STATOS_NOT_INTITALIZED	II CODLAS library was not illitialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Double-Precision BLAS1 Functions

Note: Double-precision functions are only supported on GPUs with double-precision hardware.

The double-precision BLAS1 functions are as follows:

- □ "Function cublasIdamax()" on page 44
- □ "Function cublasIdamin()" on page 44
- □ "Function cublasDasum()" on page 45
- □ "Function cublasDaxpy()" on page 46
- □ "Function cublasDcopy()" on page 47
- □ "Function cublasDdot()" on page 48
- □ "Function cublasDnrm2()" on page 49
- □ "Function cublasDrot()" on page 50
- □ "Function cublasDrotg()" on page 51
- □ "Function cublasDrotm()" on page 52
- □ "Function cublasDrotmg()" on page 53
- □ "Function cublasDscal()" on page 54
- □ "Function cublasDswap()" on page 55

Function cublasIdamax()

int

cublasIdamax (int n, const double *x, int incx)

finds the smallest index of the maximum magnitude element of double-precision vector \mathbf{x} ; that is, the result is the first \mathbf{i} , \mathbf{i} = 0 to \mathbf{n} -1, that maximizes $\mathtt{abs}(\mathbf{x}[1+\mathbf{i}*\mathtt{incx}])$. The result reflects 1-based indexing for compatibility with Fortran.

Input

•	
n	number of elements in input vector
x	double-precision vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the smallest index (returns zero if n <= 0 or incx <= 0)

Reference: http://www.netlib.org/blas/idamax.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasIdamin()

int

```
cublasIdamin (int n, const double *x, int incx)
```

finds the smallest index of the minimum magnitude element of double-precision vector \mathbf{x} ; that is, the result is the first \mathbf{i} , \mathbf{i} = 0 to n-1, that minimizes $abs(\mathbf{x}[1+\mathbf{i}*incx])$. The result reflects 1-based indexing for compatibility with Fortran.

n	number of elements in input vector
х	double-precision vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the smallest index (returns zero if n <= 0 or incx <= 0)

Analogous to http://www.netlib.org/blas/idamax.f

Error status for this function can be retrieved via cublasGetError().

Frror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDasum()

double

cublasDasum (int n, const double *x, int incx)

computes the sum of the absolute values of the elements of double-precision vector \mathbf{x} ; that is, the result is the sum from $\mathbf{i} = 0$ to $\mathbf{n} - 1$ of $abs(\mathbf{x}[1+\mathbf{i}*incx])$.

Input

•	
n	number of elements in input vector
x	double-precision vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the double-precision sum of absolute values (returns zero if n <= 0 or incx <= 0, or if an error occurred)

Reference: http://www.netlib.org/blas/dasum.f

Error status for this function can be retrieved via cublasGetError() .
Error Status

	· · · · · · · · · · · · · · · · · · ·
CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDaxpy()

void

multiplies double-precision vector \mathbf{x} by double-precision scalar alpha and adds the result to double-precision vector \mathbf{y} ; that is, it overwrites double-precision \mathbf{y} with double-precision alpha * $\mathbf{x} + \mathbf{y}$.

```
For i = 0 to n-1, it replaces
```

```
y[ly+i*incy] with alpha * x[lx+i*incx]+y[ly+i*incy],
```

where

```
1x = 0 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
alpha	double-precision scalar multiplier
x	double-precision vector with n elements
incx	storage spacing between elements of x
У	double-precision vector with n elements
incy	storage spacing between elements of $_{\mathtt{Y}}$

Output

y double-precision result (unchanged if n <= 0)

Reference: http://www.netlib.org/blas/daxpy.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDcopy()

void

copies the double-precision vector \mathbf{x} to the double-precision vector \mathbf{y} . For $\mathbf{i} = 0$ to $\mathbf{n}-1$, it copies

```
x[lx+i*incx] to y[ly+i*incy],
```

where

```
1x = 1 \text{ if incx} >= 0, \text{ else}

1x = 1 + (1 - n) * \text{incx};
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
x	double-precision vector with n elements
incx	storage spacing between elements of x
У	double-precision vector with n elements
incy	storage spacing between elements of $_{ m Y}$

Output

y contains double-precision vector x

Reference: http://www.netlib.org/blas/dcopy.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDdot()

double

computes the dot product of two double-precision vectors. It returns the dot product of the double-precision vectors x and y if successful, and 0.0 otherwise. It computes the sum for i = 0 to n-1 of

```
x[lx+i*incx]*y[ly+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
x	double-precision vector with n elements
incx	storage spacing between elements of x
У	double-precision vector with n elements
incy	storage spacing between elements of _Y

Output

returns double-precision dot product (returns zero if $n \le 0$)

Reference: http://www.netlib.org/blas/ddot.f

Error status for this function can be retrieved via **cublasGetError()**.

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDnrm2()

double

cublasDnrm2 (int n, const double *x, int incx)

computes the Euclidean norm of the double-precision n-vector x (with storage increment incx). This code uses a multiphase model of accumulation to avoid intermediate underflow and overflow.

Input

n	number of elements in input vector
x	double-precision vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the Euclidian norm (returns zero if n <= 0, incx <= 0, or if an error occurred)

Reference: http://www.netlib.org/blas/dnrm2.f

Reference: http://www.netlib.org/slatec/lin/dnrm2.f

Error status for this function can be retrieved via **cublasGetError()**.

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDrot()

void

multiplies a 2×2 matrix
$$\begin{bmatrix} dc & ds \\ -ds & dc \end{bmatrix}$$
 with the 2×n matrix $\begin{bmatrix} x^T \\ y^T \end{bmatrix}$.

The elements of x are in x[1x + i * incx], i = 0 to n-1, where

$$1x = 1 \text{ if incx} >= 0, \text{ else}$$

 $1x = 1 + (1-n) * \text{incx};$

y is treated similarly using ly and incy.

Input

n	number of elements in input vectors
х	double-precision vector with n elements
incx	storage spacing between elements of x
У	double-precision vector with n elements
incy	storage spacing between elements of $_{ m Y}$
dc	element of rotation matrix
ds	element of rotation matrix

Output

x	rotated vector x (unchanged if n <= 0)
У	rotated vector y (unchanged if n <= 0)

Reference: http://www.netlib.org/blas/drot.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDrotg()

void

```
cublasDrotg (double *host_da, double *host_db,
             double *host_dc, double *host_ds)
```

constructs the Givens transformation

$$G = \begin{bmatrix} dc & ds \\ -ds & dc \end{bmatrix}, \qquad dc^2 + ds^2 = 1$$

which zeros the second entry of the 2-vector da db.

The quantity $r = \pm \sqrt{da^2 + db^2}$ overwrites da in storage. The value of db is overwritten by a value z which allows dc and ds to be recovered by the following algorithm:

if
$$z = 1$$
 set $dc = 0.0$ and $ds = 1.0$.
if $abs(z) < 1$ set $dc = \sqrt{1 - z^2}$ and $ds = z$.
if $abs(z) > 1$ set $dc = 1/z$ and $ds = \sqrt{1 - dc^2}$.

The function **cublasDrot**(n, x, incx, y, incy, dc, ds) normally is called next to apply the transformation to a 2×n matrix. Note that this function is provided for completeness and is run exclusively on the host.

Input

da	double-precision scalar
db	double-precision scalar
Output	

da	double-precision r
db	double-precision z
dc	double-precision result
ds	double-precision result

Reference: http://www.netlib.org/blas/drotg.f

This function does not set any error status.

Function cublasDrotm()

void

applies the modified Givens transformation, h, to the $2 \times n$ matrix $\begin{bmatrix} x^T \\ y^T \end{bmatrix}$

The elements of x are in x[1x + i * incx], i = 0 to n-1, where

$$1x = 1$$
 if incx >= 0, else
 $1x = 1 + (1-n) * incx;$

y is treated similarly using ly and incy.

With dparam[0] = dflag, h has one of the following forms:

$$\begin{array}{ll} \mbox{dflag} = -1.0 & \mbox{dflag} = 0.0 \\ \mbox{h} = \begin{bmatrix} \mbox{dh00 dh01} \\ \mbox{dh10 dh11} \end{bmatrix} & \mbox{h} = \begin{bmatrix} \mbox{1.0 dh01} \\ \mbox{dh10 1.0} \end{bmatrix} \\ \mbox{dflag} = 1.0 & \mbox{dflag} = -2.0 \\ \mbox{h} = \begin{bmatrix} \mbox{dh00 1.0} \\ \mbox{-1.0 dh11} \end{bmatrix} & \mbox{h} = \begin{bmatrix} \mbox{1.0 0.0} \\ \mbox{0.0 1.0} \end{bmatrix} \end{array}$$

Input

n	number of elements in input vectors.
x	double-precision vector with ${\tt n}$ elements.
incx	storage spacing between elements of x.
У	double-precision vector with ${\tt n}$ elements.
incy	storage spacing between elements of y.
dparam	5-element vector. dparam[0] is dflag described above. dparam[1] through dparam[4] contain the 2×2 rotation matrix h: dparam[1] contains dh00, dparam[2] contains dh10, dparam[3] contains dh01, and dparam[4] contains dh11.

Output

х	rotated vector \mathbf{x} (unchanged if $\mathbf{n} \le 0$)
У	rotated vector y (unchanged if $n \le 0$)

Reference: http://www.netlib.org/blas/drotm.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDrotmg()

void

constructs the modified Givens transformation matrix h which zeros the second component of the 2-vector $(\sqrt{dd1}*dx1, \sqrt{dd2}*dy1)^T$.

With dparam[0] = dflag, h has one of the following forms:

$$\begin{array}{ll} \text{dflag} = -1.0 & \text{dflag} = 0.0 \\ h = \begin{bmatrix} \text{dh00 dh01} \\ \text{dh10 dh11} \end{bmatrix} & h = \begin{bmatrix} 1.0 & \text{dh01} \\ \text{dh10 1.0} \end{bmatrix} \\ \\ \text{dflag} = 1.0 & \text{dflag} = -2.0 \\ h = \begin{bmatrix} \text{dh00 1.0} \\ -1.0 & \text{dh11} \end{bmatrix} & h = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix} \end{array}$$

dparam[1] through dparam[4] contain dh00, dh10, dh01, and dh11, respectively. Values of 1.0, -1.0, or 0.0 implied by the value of dflag are not stored in dparam. Note that this function is provided for completeness and is run exclusively on the host.

Input

•		
dd1	double-precision scalar	
dd2	double-precision scalar	
dx1	double-precision scalar	
dy1	double-precision scalar	

Out	ากเมโ

dd1	changed to represent the effect of the transformation
dd2	changed to represent the effect of the transformation
dx1	changed to represent the effect of the transformation
dparam	5-element vector. dparam[0] is dflag described above. dparam[1] through dparam[4] contain the 2×2 rotation matrix h: dparam[1] contains dh00, dparam[2] contains dh10, dparam[3] contains dh01, and dparam[4] contains dh11.

Reference: http://www.netlib.org/blas/drotmg.f

This function does not set any error status.

Function cublasDscal()

```
void
```

cublasDscal (int n, double alpha, double *x, int incx) replaces double-precision vector x with double-precision alpha * x. For i = 0 to n-1, it replaces

```
x[lx+i*incx] with alpha * x[lx+i*incx],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx.
```

Input

n	number of elements in input vector
alpha	double-precision scalar multiplier
х	double-precision vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

x double-precision result (unchanged if n <= 0 or incx <= 0)</pre>

Reference: http://www.netlib.org/blas/dscal.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDswap()

void

interchanges double-precision vector \mathbf{x} with double-precision vector \mathbf{y} . For $\mathbf{i} = 0$ to $\mathbf{n}-1$, it interchanges

```
x[1x+i*incx] with y[1y+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar manner using incy.

Input

n	number of elements in input vectors
x	double-precision vector with n elements
incx	storage spacing between elements of x
У	double-precision vector with n elements
incy	storage spacing between elements of _Y

Output

х	double-precision vector y (unchanged from input if n <= 0)
У	double-precision vector x (unchanged from input if $n \le 0$)

Reference: http://www.netlib.org/blas/dswap.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Double-Precision Complex BLAS1 functions

Note: Double-precision functions are only supported on GPUs with double-precision hardware.

The double-precision complex BLAS1 functions are listed below:

- □ "Function cublasDzasum()" on page 58
- □ "Function cublasDznrm2()" on page 59
- □ "Function cublasIzamax()" on page 59
- □ "Function cublasIzamin()" on page 60
- □ "Function cublasZaxpy()" on page 61
- □ "Function cublasZcopy()" on page 62
- □ "Function cublasZdotc()" on page 63
- □ "Function cublasZdotu()" on page 64
- □ "Function cublasZdrot()" on page 65
- □ "Function cublasZdscal()" on page 66
- □ "Function cublasZrot()" on page 67
- □ "Function cublasZrotg()" on page 68
- □ "Function cublasZscal()" on page 68
- □ "Function cublasZswap()" on page 69

Function cublasDzasum()

double

cublasDzasum (int n, const cuDoubleComplex *x, int incx)

takes the sum of the absolute values of a complex vector and returns a double-precision result. Note that this is not the L1 norm of the vector. The result is the sum from 0 to n-1 of

```
abs(real(x[lx+i*incx])) + abs(imag(x[lx+i*incx])),
```

where

```
1x = 1 if incx <= 0, else

1x = 1 + (1-n) * incx.
```

Input

n	number of elements in input vector
x	double-precision complex vector with n elements
incx	storage spacing between elements of x

Output

returns the double-precision sum of absolute values of real and imaginary parts (returns zero if n <= 0, incx <= 0, or if an error occurred)

Reference http://www.netlib.org/blas/dzasum.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that
	does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDznrm2()

double

cublasDznrm2 (int n, const cuDoubleComplex *x, int incx)

computes the Euclidean norm of double-precision complex n-vector x. This implementation uses simple scaling to avoid intermediate underflow and overflow.

Input

n	number of elements in input vector
x	double-precision complex vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the Euclidian norm (returns zero if n <= 0, incx <= 0, or if an error occurred)

Reference: http://www.netlib.org/blas/dznrm2.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasIzamax()

int

cublasIzamax (int n, const cuDoubleComplex *x, int incx)

finds the smallest index of the maximum magnitude element of double-precision complex vector \mathbf{x} ; that is, the result is the first i, i = 0 to n-1, that maximizes

```
abs(real(x[1+i*incx])) + abs(imag(x[1+i*incx])).
```

Input

n	number of elements in input vector
x	double-precision complex vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the smallest index	returns zero if n <= 0 or incx <= 0)
----------------------------	-------------------------------------	---

Reference: http://www.netlib.org/blas/izamax.f

Error status for this function can be retrieved via cublasGetError().

Frror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasIzamin()

int

cublasIzamin (int n, const cuDoubleComplex *x, int incx)

finds the smallest index of the minimum magnitude element of double-precision complex vector \mathbf{x} ; that is, the result is the first i, i = 0 to n-1, that minimizes

abs(real(x[1+i*incx])) + abs(imag(x[1+i*incx])).

Input

n	number of elements in input vector
x	double-precision complex vector with ${\tt n}$ elements
incx	storage spacing between elements of \mathbf{x}

Output

returns the smallest index (returns zero if n <= 0 or incx <= 0)

Reference: analogous to "Function cublasIzamax()" on page 59.

Error status for this function can be retrieved via **cublasGetError()**.

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZaxpy()

void

multiplies double-precision complex vector \mathbf{x} by double-precision complex scalar alpha and adds the result to double-precision complex vector \mathbf{y} ; that is, it overwrites double-precision complex \mathbf{y} with double-precision complex alpha * $\mathbf{x} + \mathbf{y}$.

```
For i = 0 to n-1, it replaces
```

```
y[ly+i*incy] with alpha *x[lx+i*incx]+y[ly+i*incy],
```

where

```
1x = 0 if incx >= 0, else 1x = 1 + (1 - n) * incx;
```

ly is defined in a similar way using incy.

Input

•	
n	number of elements in input vectors
alpha	double-precision complex scalar multiplier
x	double-precision complex vector with n elements
incx	storage spacing between elements of x
У	double-precision complex vector with n elements
incy	storage spacing between elements of $_{ m Y}$

Output

У	double-precision complex result (unchanged if n <= 0)

Reference: http://www.netlib.org/blas/zaxpy.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

CUBLAS_STATUS_ARCH_MISMATCH if function invoked on device that does not support double precision
```

Function cublasZcopy()

void

copies the double-precision complex vector x to the double-precision complex vector y. For i = 0 to n-1, it copies

```
x[lx+i*incx] to y[ly+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1 - n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
x	double-precision complex vector with n elements
incx	storage spacing between elements of x
У	double-precision complex vector with n elements
incy	storage spacing between elements of $_{ m Y}$

Output

y contains double-precision complex vector x

Reference: http://www.netlib.org/blas/zcopy.f

Error status for this function can be retrieved via **cublasGetError()**.

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

CUBLAS_STATUS_ARCH_MISMATCH if function invoked on device that does not support double precision
```

CHAPTER 2 BLAS1 Functions

Function cublasZdotc()

cuDoubleComplex

computes the dot product of two double-precision complex vectors. It returns the dot product of the double-precision complex vectors x and y if successful, and complex zero otherwise. For i = 0 to n-1, it sums the products

```
\overline{x[1x+i*incx]} * y[1y+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
х	double-precision complex vector with n elements
incx	storage spacing between elements of x
У	double-precision complex vector with n elements
incy	storage spacing between elements of $_{\mathtt{Y}}$

Output

returns double-precision complex dot product (zero if n <= 0)

Reference: http://www.netlib.org/blas/zdotc.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_ALLOC_FAILED if function could not allocate reduction buffer

CUBLAS_STATUS_ARCH_MISMATCH if function invoked on device that does not support double precision

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasZdotu()

cuDoubleComplex

computes the dot product of two double-precision complex vectors. It returns the dot product of the double-precision complex vectors \mathbf{x} and \mathbf{y} if successful, and complex zero otherwise. For $\mathbf{i} = 0$ to $\mathbf{n} - 1$, it sums the products

```
x[lx+i*incx]*y[ly+i*incy],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx;
```

ly is defined in a similar way using incy.

Input

n	number of elements in input vectors
х	double-precision complex vector with n elements
incx	storage spacing between elements of x
У	double-precision complex vector with n elements
incy	storage spacing between elements of $_{\mathtt{Y}}$

Output

returns double-precision complex dot product (returns zero if n <= 0)

Reference: http://www.netlib.org/blas/zdotu.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ALLOC_FAILED	if function could not allocate reduction buffer
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

PG-00000-002_V3.0 NVIDIA CHAPTER 2 BLAS1 Functions

Function cublasZdrot()

void

multiplies a 2×2 matrix
$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$
 with the 2×n matrix $\begin{bmatrix} x^T \\ y^T \end{bmatrix}$.

The elements of x are in x[lx+i*incx], i = 0 to n-1, where

$$1x = 1$$
 if incx >= 0, else
 $1x = 1 + (1-n) * incx;$

y is treated similarly using 1y and incy.

Input

n	number of elements in input vectors
x	double-precision complex vector with n elements
incx	storage spacing between elements of x
У	double-precision complex vector with n elements
incy	storage spacing between elements of $_{ m Y}$
С	double-precision cosine component of rotation matrix
s	double-precision sine component of rotation matrix

Output

х	rotated vector \mathbf{x} (unchanged if $n \le 0$)
У	rotated vector y (unchanged if n <= 0)

Reference: http://www.netlib.org/blas/zdrot.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_ARCH_MISMATCH if function invoked on device that does not support double precision

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasZdscal()

void

replaces double-precision complex vector \mathbf{x} with double-precision complex alpha * \mathbf{x} .

```
For i = 0 to n-1, it replaces
```

```
x[1x+i*incx] with alpha * x[1x+i*incx],
```

where

```
1x = 1 if incx >= 0, else

1x = 1 + (1-n) * incx.
```

Input

n	number of elements in input vector
alpha	double-precision scalar multiplier
x	double-precision complex vector with n elements
incx	storage spacing between elements of \times

Output

x double-precision complex result (unchanged if n <= 0 or incx <= 0)</p>

Reference: http://www.netlib.org/blas/zdscal.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_ARCH_MISMATCH if function invoked on device that does not support double precision

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

PG-00000-002_V3.0 NVIDIA CHAPTER 2 BLAS1 Functions

Function cublasZrot()

void

multiplies a 2×2 matrix
$$\begin{bmatrix} sccs \\ -css \end{bmatrix}$$
 with the 2×n matrix $\begin{bmatrix} x^T \\ y^T \end{bmatrix}$.

The elements of x are in x[lx+i*incx], i = 0 to n-1, where

$$1x = 1$$
 if incx >= 0, else
 $1x = 1 + (1-n) * incx;$

y is treated similarly using ly and incy.

Input

n	number of elements in input vectors
x	double-precision complex vector with n elements
incx	storage spacing between elements of x
У	double-precision complex vector with n elements
incy	storage spacing between elements of y
sc	double-precision cosine component of rotation matrix
CS	double-precision complex sine component of rotation matrix

Output

х	rotated double-precision complex vector \mathbf{x} (unchanged if $\mathbf{n} \le 0$)
У	rotated double-precision complex vector y (unchanged if $n \le 0$)

Reference: http://netlib.org/lapack/explore-html/zrot.f.html

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZrotg()

void

constructs the complex Givens transformation

$$G = \begin{bmatrix} sc cs \\ -cs sc \end{bmatrix}, sc^2 + |cs|^2 = 1$$

which zeros the second entry of the complex 2-vector $\begin{bmatrix} ca & cb \end{bmatrix}^T$.

The quantity $ca/|ca|^*||ca|$, cb|| overwrites ca in storage. The function cublasCrot (n, x, incx, y, incy, sc, cs) normally is called next to apply the transformation to a $2\times n$ matrix. Note that this function is provided for completeness and is run exclusively on the host.

Input

ca	double-precision complex scalar
cb	double-precision complex scalar
Output	
ca	double-precision complex ca/ ca * ca, cb
sc	double-precision cosine component of rotation matrix
CS	double-precision complex sine component of rotation matrix

Reference: http://www.netlib.org/blas/zrotg.f

This function does not set any error status.

Function cublasZscal()

void

replaces double-precision complex vector \mathbf{x} with double-precision complex alpha * \mathbf{x} .

68 PG-00000-002_V3.0 NVIDIA

CHAPTER 2 BLAS1 Functions

```
For i = 0 to n-1, it replaces
    x[lx+i*incx] with alpha * x[lx+i*incx],
where
    lx = 1 if incx >= 0, else
    lx = 1+(1-n)*incx.
```

Input

n	number of elements in input vector
alpha	double-precision complex scalar multiplier
x	double-precision complex vector with n elements
incx	storage spacing between elements of \mathbf{x}

Output

x double-precision complex result (unchanged if $n \le 0$ or $incx \le 0$)

Reference: http://www.netlib.org/blas/zscal.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

CUBLAS_STATUS_ARCH_MISMATCH if function invoked on device that does not support double precision
```

Function cublasZswap()

```
void
```

interchanges double-precision complex vector \mathbf{x} with double-precision complex vector \mathbf{y} . For $\mathbf{i} = 0$ to $\mathbf{n} - 1$, it interchanges

```
x[lx+i*incx] with y[ly+i*incy],
where
lx = 1 if incx >= 0, else
lx = 1+(1-n)*incx;
```

ly is defined in a similar manner using incy.

Input

n	number of elements in input vectors
х	double-precision complex vector with n elements
incx	storage spacing between elements of x
У	double-precision complex vector with n elements
incy	storage spacing between elements of $_{\mathtt{Y}}$
Output	
х	double-precision complex vector y (unchanged from input if $n \le 0$)
У	double-precision complex vector \mathbf{x} (unchanged from input if $n \le 0$)

Reference: http://www.netlib.org/blas/zswap.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision

CHAPTER

3

Single-Precision BLAS2 Functions

The Level 2 Basic Linear Algebra Subprograms (BLAS2) are functions that perform matrix-vector operations. The CUBLAS implementations of single-precision BLAS2 functions are described in these sections:

- □ "Single-Precision BLAS2 Functions" on page 72
- □ "Single-Precision Complex BLAS2 Functions" on page 95

Single-Precision BLAS2 Functions

The single-precision BLAS2 functions are as follows:

- □ "Function cublasSgbmv()" on page 73
- □ "Function cublasSgemv()" on page 74
- □ "Function cublasSger()" on page 75
- □ "Function cublasSsbmv()" on page 76
- □ "Function cublasSspmv()" on page 78
- □ "Function cublasSspr()" on page 79
- □ "Function cublasSspr2()" on page 80
- □ "Function cublasSsymv()" on page 82
- □ "Function cublasSsyr()" on page 83
- □ "Function cublasSsyr2()" on page 84
- □ "Function cublasStbmv()" on page 86
- □ "Function cublasStbsv()" on page 87
- □ "Function cublasStpmv()" on page 89
- □ "Function cublasStpsv()" on page 90
- □ "Function cublasStrmv()" on page 92
- □ "Function cublasStrsv()" on page 93

Function cublasSgbmv()

void

performs one of the matrix-vector operations

```
y = alpha * op(A) * x + beta * y,
where op(A) = A or op(A) = A^{T},
```

alpha and beta are single-precision scalars, and x and y are single-precision vectors. A is an $m \times n$ band matrix consisting of single-precision elements with kl subdiagonals and ku superdiagonals.

Input

```
specifies op(A). If trans == 'N' or 'n', op(A) = A.
trans
         If trans == 'T', 't', 'C', Or 'c', op(A) = A^{T}.
         the number of rows of matrix A: m must be at least zero.
m
         the number of columns of matrix A: n must be at least zero.
n
k1
         the number of subdiagonals of matrix A; k1 must be at least zero.
ku
         the number of superdiagonals of matrix A; ku must be at least zero.
         single-precision scalar multiplier applied to op (A).
         single-precision array of dimensions (lda, n). The leading
Α
         (kl + ku + 1) \times n part of array A must contain the band matrix A,
         supplied column by column, with the leading diagonal of the matrix in
         row ku+1 of the array, the first superdiagonal starting at position 2 in
         row ku, the first subdiagonal starting at position 1 in row ku+2, and so
         on. Elements in the array A that do not correspond to elements in the
         band matrix (such as the top left ku×ku triangle) are not referenced.
lda
         leading dimension of A; 1da must be at least k1 + ku + 1.
х
         single-precision array of length at least (1 + (n-1) * abs(incx))
         when trans == 'N' or 'n', and at least (1 + (m-1) * abs(incx))
         otherwise.
incx
         storage spacing between elements of x; incx must not be zero.
beta
         single-precision scalar multiplier applied to vector y. If beta is zero, y
         is not read.
```

Input (continued)

```
single-precision array of length at least (1 + (m-1) * abs(incy))

when trans == 'N' or 'n' and at least (1 + (n-1) * abs(incy))

otherwise. If beta is zero, y is not read.

incy storage spacing between elements of y; incy must not be zero.
```

Output

```
y updated according to y = alpha * op(A) * x + beta * y.
```

Reference: http://www.netlib.org/blas/sgbmv.f

Error status for this function can be retrieved via cublasGetError().

Frror Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if m < 0, n < 0, kl < 0, ku < 0,

incx == 0, or incy == 0

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasSgemv()

77014

performs one of the matrix-vector operations

```
y = alpha * op(A) * x + beta * y,
where op(A) = A or op(A) = A^{T},
```

alpha and beta are single-precision scalars, and x and y are single-precision vectors. A is an $m \times n$ matrix consisting of single-precision elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array in which A is stored.

Input

74 PG-00000-002_V3.0

Input (continued)

alpha	single-precision scalar multiplier applied to op(A).
A	single-precision array of dimensions (lda, n) if trans == 'N' or 'n', of dimensions (lda, m) otherwise; lda must be at least $max(1, m)$ if trans == 'N' or 'n' and at least $max(1, n)$ otherwise.
lda	leading dimension of two-dimensional array used to store matrix A.
x	single-precision array of length at least $(1+(n-1)*abs(incx))$ if
	trans == 'N' or 'n', else at least $(1+(m-1)*abs(incx))$.
incx	specifies the storage spacing for elements of x ; incx must not be zero.
beta	single-precision scalar multiplier applied to vector \mathbf{y} . If beta is zero, \mathbf{y} is not read.
У	single-precision array of length at least $(1 + (m-1) * abs(incy))$ if
	trans == 'N' or 'n', else at least $(1+(n-1)*abs(incy))$.
incy	the storage spacing between elements of y; incy must not be zero.

Output

```
y updated according to y = alpha * op(A) * x + beta * y.
```

Reference: http://www.netlib.org/blas/sgemv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

Function cublasSger()

void

performs the symmetric rank 1 operation

$$A = alpha * x * y^T + A$$
,

where alpha is a single-precision scalar, x is an m-element single-precision vector, y is an n-element single-precision vector, and A is an $m \times n$ matrix consisting of single-precision elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array used to store A.

Input

m	specifies the number of rows of the matrix A; m must be at least zero.
n	specifies the number of columns of matrix A ; n must be at least zero.
alpha	single-precision scalar multiplier applied to $\mathbf{x} \star \mathbf{y}^{\mathrm{T}}$.
х	single-precision array of length at least $(1 + (m-1) * abs(incx))$.
incx	the storage spacing between elements of x ; incx must not be zero.
У	single-precision array of length at least $(1+(n-1)*abs(incy))$.
incy	the storage spacing between elements of y; incy must not be zero.
A	single-precision array of dimensions (lda, n).
lda	leading dimension of two-dimensional array used to store matrix A.

Output

```
A updated according to A = alpha * x * y^T + A.
```

Reference: http://www.netlib.org/blas/sger.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

Function cublasSsbmv()

```
----
```

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

76 PG-00000-002_V3.0

where alpha and beta are single-precision scalars, and x and y are n-element single-precision vectors. A is an $n \times n$ symmetric band matrix consisting of single-precision elements, with k superdiagonals and the same number of subdiagonals.

Input

uplo	specifies whether the upper or lower triangular part of the symmetric band matrix A is being supplied. If uplo == 'U' or 'u', the upper triangular part is being supplied. If uplo == 'L' or 'l', the lower triangular part is being supplied.
n	specifies the number of rows and the number of columns of the symmetric matrix A; n must be at least zero.
k	specifies the number of superdiagonals of matrix A. Since the matrix is symmetric, this is also the number of subdiagonals; ${\tt k}$ must be at least zero.
alpha	single-precision scalar multiplier applied to A * x.
A	single-precision array of dimensions (lda, n). When uplo == 'U' or 'u', the leading (k+1)×n part of array A must contain the upper triangular band of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row k+1 of the array, the first superdiagonal starting at position 2 in row k, and so on. The top left k×k triangle of the array A is not referenced. When uplo == 'L' or 'l', the leading (k+1)×n part of the array A must contain the lower triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first subdiagonal starting at position 1 in row 2, and so on. The bottom right k×k triangle of the array A is not referenced.
lda	leading dimension of A; 1da must be at least k+1.
x	single-precision array of length at least $(1 + (n-1) * abs(incx))$.
incx	storage spacing between elements of x; incx must not be zero.
beta	single-precision scalar multiplier applied to vector \mathbf{y} . If beta is zero, \mathbf{y} is not read.
У	single-precision array of length at least $(1+(n-1)*abs(incy))$. If beta is zero, y is not read.
incy	storage spacing between elements of y ; incy must not be zero.
Output	
У	updated according to y = alpha * A * x + beta * y.

Reference: http://www.netlib.org/blas/ssbmv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if k < 0, n < 0, incx == 0, or
	incy == 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasSspmv()

```
void
```

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are single-precision scalars, and x and y are n-element single-precision vectors. A is a symmetric $n \times n$ matrix that consists of single-precision elements and is supplied in packed form.

Input

```
uplo
         specifies whether the matrix data is stored in the upper or the lower
         triangular part of array AP. If uplo == 'U' or 'u', the upper triangular
         part of A is supplied in AP. If uplo == 'L' or 'l', the lower triangular
         part of A is supplied in AP.
         the number of rows and columns of matrix A; n must be at least zero.
n
         single-precision scalar multiplier applied to A * x.
alpha
ΑP
         single-precision array with at least (n * (n + 1))/2 elements. If
         uplo == 'U' or 'u', array AP contains the upper triangular part of the
         symmetric matrix A, packed sequentially, column by column; that is, if
         i \le j, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo == 'L'
         or 'l', the array AP contains the lower triangular part of the
         symmetric matrix A, packed sequentially, column by column; that is, if
         i >= j, A[i,j] is stored in AP[i + ((2 * n - j + 1) * j)/2].
x
         single-precision array of length at least (1 + (n-1) * abs(incx)).
incx
         storage spacing between elements of x; incx must not be zero.
```

78 PG-00000-002_V3.0
NVIDIA

Input (continued)

beta	single-precision scalar multiplier applied to vector y. If beta is zero, y is not read.
У	single-precision array of length at least $(1+(n-1)*abs(incy))$. If beta is zero, y is not read.
incy	storage spacing between elements of y ; incy must not be zero.

Output

```
y updated according to y = alpha * A * x + beta * y.
```

Reference: http://www.netlib.org/blas/sspmv.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0, incx == 0, or incy == 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasSspr()

void

performs the symmetric rank 1 operation

$$A = alpha * x * x^T + A$$
,

where alpha is a single-precision scalar, and x is an n-element single-precision vector. A is a symmetric $n \times n$ matrix that consists of single-precision elements and is supplied in packed form.

Input

```
specifies whether the matrix data is stored in the upper or the lower
triangular part of array AP. If uplo == 'U' or 'u', the upper triangular
part of A is supplied in AP. If uplo == 'L' or 'l', the lower triangular
part of A is supplied in AP.

the number of rows and columns of matrix A; n must be at least zero.

single-precision scalar multiplier applied to x * x<sup>T</sup>.

single-precision array of length at least (1 + (n - 1) * abs(incx)).
```

Input (continued)

incx storage spacing between elements of x; incx must not be zero.

AP single-precision array with at least (n * (n+1))/2 elements. If uplo == 'U' or 'u', array AP contains the upper triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if i <= j, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo == 'L' or 'l', the array AP contains the lower triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if i >= j, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2].

Output

A updated according to $A = alpha * x * x^T + A$.

Reference: http://www.netlib.org/blas/sspr.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0 or incx == 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasSspr2()

```
void
```

performs the symmetric rank 2 operation

```
A = alpha * x * y^T + alpha * y * x^T + A,
```

where alpha is a single-precision scalar, and x and y are n-element single-precision vectors. A is a symmetric $n \times n$ matrix that consists of single-precision elements and is supplied in packed form.

80 PG-00000-002_V3.0 NVIDIA

Input

uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array A. If uplo == 'U' or 'u', only the upper
	triangular part of A may be referenced and the lower triangular part of A is inferred. If uplo == 'L' or 'l', only the lower triangular part of A may be referenced and the upper triangular part of A is inferred.
n	the number of rows and columns of matrix A; n must be at least zero.
alpha	single-precision scalar multiplier applied to $x * y^T + alpha * y * x^T$.
arpiia	single-precision scalar multiplier applied to x * y + alpha * y * x 1.
х	single-precision array of length at least $(1+(n-1)*abs(incx))$.
incx	storage spacing between elements of x; incx must not be zero.
У	single-precision array of length at least $(1+(n-1)*abs(incy))$.
incy	storage spacing between elements of y ; incy must not be zero.
AP	single-precision array with at least $(n * (n + 1))/2$ elements. If
	uplo == 'U' or 'u', array AP contains the upper triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if
	$i \le j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo == 'L'
	or 'l', the array AP contains the lower triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if
	i >= j, A[i,j] is stored in AP[i + ((2 * n - j + 1) * j)/2].

Output

A updated according to $A = alpha * x * y^T + alpha * y * x^T + A$.

Reference: http://www.netlib.org/blas/sspr2.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if n < 0, incx == 0, or incy == 0

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasSsymv()

void

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are single-precision scalars, and x and y are n-element single-precision vectors. A is a symmetric $n \times n$ matrix that consists of single-precision elements and is stored in either upper or lower storage mode.

Input

- specifies whether the upper or lower triangular part of the array A is referenced. If uplo == 'u' or 'u', the symmetric matrix A is stored in upper storage mode; that is, only the upper triangular part of A is referenced while the lower triangular part of A is inferred. If uplo == 'L' or 'l', the symmetric matrix A is stored in lower storage mode; that is, only the lower triangular part of A is referenced while the upper triangular part of A is inferred.
- n specifies the number of rows and the number of columns of the symmetric matrix A; n must be at least zero.
- alpha single-precision scalar multiplier applied to A \star x.
- single-precision array of dimensions (lda, n). If uplo == 'U' or 'u', the leading $n \times n$ upper triangular part of the array A must contain the upper triangular part of the symmetric matrix, and the strictly lower triangular part of A is not referenced. If uplo == 'L' or 'l', the leading $n \times n$ lower triangular part of the array A must contain the lower triangular part of the symmetric matrix, and the strictly upper triangular part of A is not referenced.
- lda leading dimension of A; lda must be at least max(1, n).
- single-precision array of length at least (1 + (n-1) * abs(incx)).
- incx storage spacing between elements of x; incx must not be zero.
- beta single-precision scalar multiplier applied to vector y.

82 PG-00000-002_V3.0 NVIDIA

Input (continued)

У	single-precision array of length at least $(1+(n-1)*abs(incy))$. If beta is zero, y is not read.
incy	storage spacing between elements of y; incy must not be zero.

Output

```
y updated according to y = alpha * A * x + beta * y.
```

Reference: http://www.netlib.org/blas/ssymv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if n < 0, incx == 0, or incy == 0

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasSsyr()

void

```
A = alpha * x * x^T + A,
```

where alpha is a single-precision scalar, x is an n-element single-precision vector, and A is an $n \times n$ symmetric matrix consisting of single-precision elements. A is stored in column-major format, and lda is the leading dimension of the two-dimensional array containing A.

Input

```
specifies whether the matrix data is stored in the upper or the lower
triangular part of array A. If uplo == 'U' or 'u', only the upper
triangular part of A is referenced. If uplo == 'L' or 'l', only the
lower triangular part of A is referenced.

n the number of rows and columns of matrix A; n must be at least zero.

alpha single-precision scalar multiplier applied to x * x<sup>T</sup>.

x single-precision array of length at least (1+(n-1) * abs(incx)).
```

Input ((continued)	١
IIIPGL (COLLINIACA	,

incx	the storage spacing between elements of x; incx must not be zero.
A	single-precision array of dimensions (lda, n). If uplo == 'U' or 'u', A contains the upper triangular part of the symmetric matrix, and the strictly lower triangular part is not referenced. If uplo == 'L' or 'l', A contains the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.
lda	leading dimension of the two-dimensional array containing A; lda must be at least $max(1, n)$.

Output

A updated according to $A = alpha * x * x^T + A$.

Reference: http://www.netlib.org/blas/ssyr.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasSsyr2()

```
void
```

performs the symmetric rank 2 operation

```
A = alpha * x * y^T + alpha * y * x^T + A,
```

where alpha is a single-precision scalar, x and y are n-element single-precision vectors, and A is an $n \times n$ symmetric matrix consisting of single-precision elements.

84 PG-00000-002_V3.0 NVIDIA

	ш	.,	u	

uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array A. If $uplo == 'u'$ or $'u'$, only the upper triangular part of A is referenced and the lower triangular part of A is inferred. If $uplo == 'L'$ or $'l'$, only the lower triangular part of A is referenced and the upper triangular part of A is inferred.
n	the number of rows and columns of matrix \mathtt{A} ; \mathtt{n} must be at least zero.
alpha	single-precision scalar multiplier applied to $\mathbf{x} \star \mathbf{y}^{\mathtt{T}} + \mathbf{y} \star \mathbf{x}^{\mathtt{T}}$.
x	single-precision array of length at least $(1 + (n-1) * abs(incx))$.
incx	storage spacing between elements of x; incx must not be zero.
У	single-precision array of length at least $(1+(n-1)*abs(incy))$.
incy	storage spacing between elements of y ; incy must not be zero.
А	single-precision array of dimensions (lda, n). If uplo == 'U' or 'u', A contains the upper triangular part of the symmetric matrix, and the strictly lower triangular part is not referenced. If uplo == 'L' or 'l', A contains the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.
lda	leading dimension of A; lda must be at least $max(1, n)$.

Output

updated according to $A = alpha * x * y^T + alpha * y * x^T + A$.

Reference: http://www.netlib.org/blas/ssyr2.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZEDif CUBLAS library was not initializedCUBLAS_STATUS_INVALID_VALUEif n < 0, incx == 0, or incy == 0CUBLAS_STATUS_EXECUTION_FAILEDif function failed to launch on GPU

Function cublasStbmv()

void

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A or op(A) = A^{T},
```

x is an n-element single-precision vector, and A is an $n \times n$, unit or non-unit, upper or lower, triangular band matrix consisting of single-precision elements.

Input

- uplo specifies whether the matrix A is an upper or lower triangular band
 matrix. If uplo == 'U' or 'u', A is an upper triangular band matrix. If
 uplo == 'L' or 'l', A is a lower triangular band matrix.
 trans specifies op(A). If trans == 'N' or 'n', op(A) = A.
- trans specifies op(A). If trans == 'N' or 'n', op(A) = A. If trans == 'T', 't', 'C', or 'c', op(A) = A^{T} .
- diag specifies whether or not matrix A is unit triangular. If diag == 'U' or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is not assumed to be unit triangular.
- n specifies the number of rows and columns of the matrix A; n must be at least zero.
- k specifies the number of superdiagonals or subdiagonals. If uplo == 'U' or 'u', k specifies the number of superdiagonals. If uplo == 'L' or 'l' k specifies the number of subdiagonals; k must at least be zero.
- single-precision array of dimension (lda, n). If uplo == 'U' or 'u', the leading (k+1)×n part of the array A must contain the upper triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row k+1 of the array, the first superdiagonal starting at position 2 in row k, and so on. The top left k×k triangle of the array A is not referenced. If uplo == 'L' or 'l', the leading (k+1)×n part of the array A must contain the lower triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first subdiagonal starting at position 1 in row 2, and so on. The bottom right k×k triangle of the array is not referenced.

86 PG-00000-002_V3.0 NVIDIA

Input (continued)

lda	is the leading dimension of A; 1da must be at least k+1.
х	single-precision array of length at least $(1 + (n-1) * abs(incx))$. On entry, x contains the source vector. On exit, x is overwritten with the result vector.
incx	specifies the storage spacing for elements of ${\tt x}$; incx must not be zero.

Output

x updated according to x = op(A) * x.

Reference: http://www.netlib.org/blas/stbmv.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0, k < 0, or incx == 0
CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasStbsv()

void

solves one of the systems of equations

```
op(A) * x = b,
where op(A) = A or op(A) = A^{T},
```

b and x are n-element vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular band matrix with k+1 diagonals.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

Input	
uplo	specifies whether the matrix is an upper or lower triangular band matrix: If $uplo == 'u'$ or $'u'$, A is an upper triangular band matrix. If $uplo == 'L'$ or $'l'$, A is a lower triangular band matrix.
trans	specifies $op(A)$. If trans == 'N' or 'n', $op(A) = A$.
	If trans == 'T', 't', 'C', Or 'c', op(A) = A^{T} .
diag	specifies whether A is unit triangular. If $\mathtt{diag} == '\mathtt{U'}$ or $'\mathtt{u'}$, A is assumed to be unit triangular; that is, diagonal elements are not read and are assumed to be unity. If $\mathtt{diag} == '\mathtt{N'}$ or $'\mathtt{n'}$, A is not assumed to be unit triangular.
n	the number of rows and columns of matrix A; n must be at least zero. In the current implementation n must not exceed 4070 .
k	specifies the number of superdiagonals or subdiagonals. If $uplo == 'u'$ or 'u', k specifies the number of superdiagonals. If $uplo == 'L'$ or 'l', k specifies the number of subdiagonals; k must be at least zero.
A	single-precision array of dimension (lda, n). If uplo == 'U' or 'u', the leading (k+1)×n part of the array A must contain the upper triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row k+1 of the array, the first superdiagonal starting at position 2 in row k, and so on. The top left k×k triangle of

single-precision array of length at least (1 + (n-1) * abs(incx)). On entry, x contains the n-element right-hand side vector b. On exit, it is overwritten with the solution vector x.

the array A is not referenced. If uplo == 'L' or 'l', the leading $(k+1)\times n$ part of the array A must contain the lower triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right $k\times k$ triangle of the array is not

incx storage spacing between elements of x; incx must not be zero.

Output

referenced.

x updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/stbsv.f

88 PG-00000-002_V3.0 NVIDIA

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0, n < 0, or n > 4070
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasStpmv()

void

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A or op(A) = A^T,
```

x is an n-element single-precision vector, and A is an n×n, unit or nonunit, upper or lower, triangular matrix consisting of single-precision elements.

Input

```
specifies whether the matrix A is an upper or lower triangular matrix.
uplo
         If uplo == 'U' or 'u', A is an upper triangular matrix.
         If uplo == 'L' or 'l', A is a lower triangular matrix.
trans
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
         If trans == 'T', 't', 'C', Or 'c', op(A) = A^{T}.
diag
         specifies whether or not matrix A is unit triangular.
         If diag == 'U' or 'u', A is assumed to be unit triangular.
         If diag == 'N' or 'n', A is not assumed to be unit triangular.
         specifies the number of rows and columns of the matrix A; n must be
n
         at least zero.
AΡ
         single-precision array with at least (n * (n + 1))/2 elements. If
         uplo == 'U' or 'u', the array AP contains the upper triangular part of
         the symmetric matrix A, packed sequentially, column by column; that
         is, if i \le j, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo ==
          'L' or 'l', array AP contains the lower triangular part of the
         symmetric matrix A, packed sequentially, column by column; that is, if
         i >= j, A[i,j] is stored in AP[i + ((2 * n - j + 1) * j)/2].
```

Input (continued)

х	single-precision array of length at least $(1 + (n-1) * abs(incx))$.		
	On entry, \times contains the source vector. On exit, \times is overwritten with		
	the result vector.		
incx	specifies the storage spacing for elements of x; incx must not be zero.		

Output

```
x updated according to x = op(A) * x.
```

Reference: http://www.netlib.org/blas/stpmv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if $incx == 0$ or $n < 0$
CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasStpsv()

void

solves one of the systems of equations

```
op(A) * x = b,
where op(A) = A \ or \ op(A) = A^{T},
```

b and x are n-element single-precision vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

90 PG-00000-002_V3.0 NVIDIA

Input uplo

	uplo == 'U' or 'u', A is an upper triangular matrix. If uplo == 'L' or 'l', A is a lower triangular matrix.
trans	specifies $op(A)$. If trans == 'N' or 'n', $op(A) = A$.
	If trans == 'T', 't', 'C', Or 'c', $op(A) = A^{T}$.
diag	specifies whether A is unit triangular. If $\mathtt{diag} == '\mathtt{U}'$ or $'\mathtt{u}'$, A is assumed to be unit triangular; that is, diagonal elements are not read and are assumed to be unity. If $\mathtt{diag} == '\mathtt{N}'$ or $'\mathtt{n}'$, A is not assumed to be unit triangular.
n	specifies the number of rows and columns of the matrix A; n must be at least zero. In the current implementation n must not exceed 4070.

specifies whether the matrix is an upper or lower triangular matrix. If

- single-precision array with at least (n*(n+1))/2 elements. If uplo == 'U' or 'u', array AP contains the upper triangular matrix A, packed sequentially, column by column; that is, if i <= j, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo == 'L' or 'l', array AP contains the lower triangular matrix A, packed sequentially, column by column; that is, if i >= j, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2]. When diag == 'U' or 'u', the diagonal elements of A are not referenced and are assumed to be unity.
- single-precision array of length at least (1+(n-1)*abs(incx)). On entry, x contains the n-element right-hand side vector b. On exit, it is overwritten with the solution vector x.
- incx storage spacing between elements of x; incx must not be zero.

Output

updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/stpsv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0 , $n < 0$, or $n > 4070$
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasStrmv()

void

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A or op(A) = A^{T},
```

x is an n-element single-precision vector, and A is an n×n, unit or nonunit, upper or lower, triangular matrix consisting of single-precision elements.

Input

```
specifies whether the matrix A is an upper or lower triangular matrix.
uplo
         If uplo == 'U' or 'u', A is an upper triangular matrix.
         If uplo == 'L' or 'l', A is an lower triangular matrix.
trans
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
         If trans == 'T', 't', 'C', Or 'c', op(A) = A^{T}.
diag
         specifies whether or not A is a unit triangular matrix. If diag == 'U'
         or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is
         not assumed to be unit triangular.
         specifies the number of rows and columns of the matrix A; n must be
n
         at least zero.
Α
         single-precision array of dimensions (lda, n). If uplo == 'U' or 'u',
         the leading n×n upper triangular part of the array A must contain the
         upper triangular matrix, and the strictly lower triangular part of A is
         not referenced. If uplo == 'L' or 'l', the leading n×n lower
         triangular part of the array A must contain the lower triangular matrix,
         and the strictly upper triangular part of A is not referenced. When
         diag == 'U' or 'u', the diagonal elements of A are not referenced
         either, but are assumed to be unity.
lda
         leading dimension of A; lda must be at least max(1, n).
x
         single-precision array of length at least (1 + (n-1) * abs(incx)).
         On entry, x contains the source vector. On exit, x is overwritten with
         the result vector.
         the storage spacing between elements of x; incx must not be zero.
incx
```

Output

x	updated according to $x = op(A) * x$.	
---	--	--

Reference: http://www.netlib.org/blas/strmv.f

Error status for this function can be retrieved via cublasGetError().

Frror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0 or n < 0
CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasStrsv()

void

solves a system of equations

```
op(A) * x = b,
where op(A) = A or op(A) = A^{T},
```

b and x are n-element single-precision vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix consisting of single-precision elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array containing A.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

Input

```
specifies whether the matrix data is stored in the upper or the lower
triangular part of array A. If uplo == 'U' or 'u', only the upper
triangular part of A may be referenced. If uplo == 'L' or 'l', only
the lower triangular part of A may be referenced.

trans
specifies op(A). If trans == 'N' or 'n', op(A) = A.
If trans == 'T', 't', 'C', or 'c', op(A) = A<sup>T</sup>.
```

Input (continued)

diag	specifies whether or not A is a unit triangular matrix. If diag == 'U' or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is not assumed to be unit triangular.
n	specifies the number of rows and columns of the matrix ${\tt A}$; ${\tt n}$ must be at least zero.
A	single-precision array of dimensions (lda, n). If $\mathtt{uplo} == \mathtt{'U'}$ or $\mathtt{'u'}$, A contains the upper triangular part of the symmetric matrix, and the strictly lower triangular part is not referenced. If $\mathtt{uplo} == \mathtt{'L'}$ or $\mathtt{'l'}$, A contains the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.
lda	leading dimension of the two-dimensional array containing A; lda must be at least $max(1, n)$.
х	single-precision array of length at least $(1+(n-1)*abs(incx))$. On entry, x contains the n -element, right-hand-side vector b . On exit, it is overwritten with the solution vector x .
incx	the storage spacing between elements of x; incx must not be zero.

Output

updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/strsv.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0 or n < 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

94 PG-00000-002_V3.0 NVIDIA

Single-Precision Complex BLAS2 Functions

The two single-precision complex BLAS2 functions are as follows:

- □ "Function cublasCgbmv()" on page 96
- □ "Function cublasCgemv()" on page 97
- □ "Function cublasCgerc()" on page 99
- □ "Function cublasCgeru()" on page 100
- □ "Function cublasChbmv()" on page 101
- □ "Function cublasChemv()" on page 103
- □ "Function cublasCher()" on page 104
- □ "Function cublasCher2()" on page 105
- □ "Function cublasChpmv()" on page 107
- □ "Function cublasChpr()" on page 108
- □ "Function cublasChpr2()" on page 109
- □ "Function cublasCtbmv()" on page 111
- □ "Function cublasCtbsv()" on page 113
- □ "Function cublasCtpmv()" on page 114
- □ "Function cublasCtpsv()" on page 116
- □ "Function cublasCtrmv()" on page 117
- □ "Function cublasCtrsv()" on page 119

Function cublasCgbmv()

void

performs one of the matrix-vector operations

```
y = alpha * op(A) * x + beta * y, where

op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

alpha and beta are single-precision complex scalars, and x and y are single-precision complex vectors. A is an $m \times n$ band matrix consisting of single-precision complex elements with k1 subdiagonals and ku superdiagonals.

Input

```
trans
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
         If trans == 'T' Or 't', op(A) = A^{T}.
         If trans == 'C', Or 'c', op(A) = A^{H}.
m
         specifies the number of rows of matrix A; m must be at least zero.
         specifies the number of columns of matrix A; n must be at least zero.
n
kl
         specifies the number of subdiagonals of matrix A; k1 must be at least
         specifies the number of superdiagonals of matrix A; ku must be at
ku
         least zero.
         single-precision complex scalar multiplier applied to op(A).
alpha
Α
         single-precision complex array of dimensions (1da, n). The leading
         (k1+ku+1)×n part of the array A must contain the band matrix A,
         supplied column by column, with the leading diagonal of the matrix in
         row (ku+1) of the array, the first superdiagonal starting at position 2 in
         row ku, the first subdiagonal starting at position 1 in row (ku+2), and
         so on. Elements in the array A that do not correspond to elements in
         the band matrix (such as the top left kuxku triangle) are not
         referenced.
lda
         leading dimension A; lda must be at least (k1+ku+1).
```

96 PG-00000-002_V3.0 NVIDIA

Input (continued)

х	single-precision complex array of length at least
	(1+(n-1)*abs(incx)) if trans == 'N' or 'n', else at least
	(1+(m-1)*abs(incx)).
incx	specifies the increment for the elements of x ; incx must not be zero.
beta	single-precision complex scalar multiplier applied to vector \mathbf{y} . If beta is zero, \mathbf{y} is not read.
У	single-precision complex array of length at least
	(1+(m-1)*abs(incy)) if trans == 'N' or 'n', else at least
	(1+(n-1)* abs(incy)). If beta is zero, y is not read.
incy	on entry, incy specifies the increment for the elements of y ; incy must not be zero.

Output

```
updated according to y = alpha * op(A) * x + beta * y.
```

Reference: http://www.netlib.org/blas/cgbmv.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
if CUBLAS library was not initialized
CUBLAS_STATUS_NOT_INITIALIZED
                                     if m < 0, n < 0, incx == 0, or
CUBLAS_STATUS_INVALID_VALUE
                                     incy == 0
                                     if function failed to launch on GPU
CUBLAS_STATUS_EXECUTION_FAILED
```

Function cublasCgemv()

```
void
cublasCgemv (char trans, int m, int n,
              cuComplex alpha, const cuComplex *A,
              int lda, const cuComplex *x, int incx,
              cuComplex beta, cuComplex *y, int incy)
performs one of the matrix-vector operations
  y = alpha * op(A) * x + beta * y,
```

```
PG-00000-002_V3.0
                                                                                 97
                                      NVIDIA
```

where op(A) = A, op(A) = A^{T} , or op(A) = A^{H} ;

alpha and beta are single-precision complex scalars; and x and y are single-precision complex vectors. A is an $m \times n$ matrix consisting of single-precision complex elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array in which A is stored.

Input

```
trans
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
         If trans == 'T' Or 't', op(A) = A^{T}.
         If trans == 'C' Or 'c', op(A) = A^{H}.
         specifies the number of rows of matrix A; m must be at least zero.
m
         specifies the number of columns of matrix A; n must be at least zero.
alpha
         single-precision complex scalar multiplier applied to op(A).
         single-precision complex array of dimensions (lda, n) if trans ==
Α
         'N' or 'n', of dimensions (lda, m) otherwise; lda must be at least
         \max(1, m) if trans == 'N' or 'n' and at least \max(1, n) otherwise.
         leading dimension of two-dimensional array used to store matrix A.
lda
         single-precision complex array of length at least
x
         (1+(n-1)*abs(incx)) if trans == 'N' or 'n', else at least
         (1 + (m-1) * abs(incx)).
incx
         specifies the storage spacing for elements of x; incx must not be zero.
beta
         single-precision complex scalar multiplier applied to vector y. If beta
         is zero, y is not read.
         single-precision complex array of length at least
У
         (1+(m-1)*abs(incy)) if trans == 'N' or 'n', else at least
         (1+(n-1)*abs(incy)).
incy
         the storage spacing between elements of y; incy must not be zero.
```

Output

```
y updated according to y = alpha * op(A) * x + beta * y.
```

Reference: http://www.netlib.org/blas/cgemv.f

98 PG-00000-002_V3.0 NVIDIA Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0, n < 0, incx == 0, or
	incy == 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasCgerc()

performs the symmetric rank 1 operation

```
A = alpha * x * y^{H} + A,
```

where alpha is a single-precision complex scalar, x is an m-element single-precision complex vector, y is an n-element single-precision complex vector, and A is an $m \times n$ matrix consisting of single-precision complex elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array used to store A.

Input

m	specifies the number of rows of the matrix A; m must be at least zero.
n	specifies the number of columns of matrix \mathtt{A} ; \mathtt{n} must be at least zero.
alpha	single-precision complex scalar multiplier applied to $\times \ * \ y^H$.
х	single-precision complex array of length at least
	(1+(m-1)*abs(incx)).
incx	the storage spacing between elements of x; incx must not be zero.
У	single-precision complex array of length at least
	(1+(n-1)*abs(incy)).
incy	the storage spacing between elements of y; incy must not be zero.
A	single-precision complex array of dimensions (lda, n).
lda	leading dimension of two-dimensional array used to store matrix A.

Output

Reference: http://www.netlib.org/blas/cgerc.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasCgeru()

```
void
cublasCgeru (int m, int n, cuComplex alpha,
```

const cuComplex *x, int incx,
const cuComplex *y, int incy,
cuComplex *A, int lda)

performs the symmetric rank 1 operation

$$A = alpha * x * y^T + A,$$

where alpha is a single-precision complex scalar, x is an m-element single-precision complex vector, y is an n-element single-precision complex vector, and A is an $m \times n$ matrix consisting of single-precision complex elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array used to store A.

Input

```
m specifies the number of rows of the matrix A; m must be at least zero. specifies the number of columns of matrix A; m must be at least zero. single-precision complex scalar multiplier applied to x * y^T.  x single-precision complex array of length at least \\ (1+(m-1)*abs(incx)).  incx the storage spacing between elements of x; incx must not be zero.  y single-precision complex array of length at least \\ (1+(n-1)*abs(incy)).
```

Input (continued)

incy	the storage spacing between elements of y; incy must not be zero.
A	single-precision complex array of dimensions (lda, n).
lda	leading dimension of two-dimensional array used to store matrix A.

Output

```
A updated according to A = alpha * x * y^T + A.
```

Reference: http://www.netlib.org/blas/cgeru.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasChbmv()

```
void
```

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are single-precision complex scalars, and x and y are n-element single-precision complex vectors. A is a Hermitian $n \times n$ band matrix that consists of single-precision complex elements, with k superdiagonals and the same number of subdiagonals.

Input

```
specifies whether the upper or lower triangular part of the Hermitian band matrix A is being supplied. If uplo == 'U' or 'u', the upper triangular part is being supplied. If uplo == 'L' or 'l', the lower triangular part is being supplied.

n specifies the number of rows and the number of columns of the
```

Input (continued)

k	specifies the number of superdiagonals of matrix A. Since the matrix is Hermitian, this is also the number of subdiagonals; k must be at least zero.
alpha	single-precision complex scalar multiplier applied to A * x.
A	single-precision complex array of dimensions (lda, n). If uplo == 'U' or 'u', the leading (k + 1) ×n part of array A must contain the upper triangular band of the Hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row k + 1 of the array, the first superdiagonal starting at position 2 in row k, and so on. The top left k×k triangle of array A is not referenced. When uplo == 'L' or 'l', the leading (k + 1) ×n part of array A must contain the lower triangular band part of the Hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first subdiagonal starting at position 1 in row 2, and so on. The bottom right k×k triangle of array A is not referenced. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero.
lda	leading dimension of A; 1da must be at least k + 1.
х	single-precision complex array of length at least
	(1+(n-1)*abs(incx)).
incx	storage spacing between elements of x; incx must not be zero.
beta	single-precision complex scalar multiplier applied to vector y.
У	single-precision complex array of length at least
	(1+(n-1)*abs(incy)). If beta is zero, y is not read.
incy	storage spacing between elements of y ; incy must not be zero.

Output

y updated according to y = alpha * A * x + beta * y.

Reference: http://www.netlib.org/blas/chbmv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if k < 0, n < 0, incx == 0, or
	incy == 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasChemv()

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are single-precision complex scalars, and x and y are n-element single-precision complex vectors. A is a Hermitian $n \times n$ matrix that consists of single-precision complex elements and is stored in either upper or lower storage mode.

Input

uplo specifies whether the upper or lower triangular part of the array A is referenced. If uplo == 'U' or 'u', the Hermitian matrix A is stored in upper storage mode; that is, only the upper triangular part of A is referenced while the lower triangular part of A is inferred. If uplo == 'L' or 'l', the Hermitian matrix A is stored in lower storage mode; that is, only the lower triangular part of A is referenced while the upper triangular part of A is inferred. specifies the number of rows and the number of columns of the n symmetric matrix A; n must be at least zero. alpha single-precision complex scalar multiplier applied to A * x. single-precision complex array of dimensions (lda, n). If uplo == Α 'U' or 'u', the leading n×n upper triangular part of the array A must contain the upper triangular part of the Hermitian matrix, and the strictly lower triangular part of A is not referenced. If uplo == 'L' or '1', the leading n×n lower triangular part of the array A must contain the lower triangular part of the Hermitian matrix, and the strictly upper triangular part of A is not referenced. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero. leading dimension of A; 1da must be at least max(1, n). lda х single-precision complex array of length at least (1+(n-1)*abs(incx)).incx storage spacing between elements of x; incx must not be zero.

Input (continued)

incy	storage spacing between elements of y ; incy must not be zero.
	(1+(n-1)* abs(incy)). If beta is zero, y is not read.
У	single-precision complex array of length at least
beta	single-precision complex scalar multiplier applied to vector y.

Output

```
y updated according to y = alpha * A * x + beta * y.
```

Reference: http://www.netlib.org/blas/chemv.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if n < 0, incx == 0, or incy == 0

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
```

Function cublasCher()

```
void
```

performs the Hermitian rank 1 operation

```
A = alpha * x * x^{H} + A,
```

where alpha is a single-precision scalar, x is an n-element single-precision complex vector, and A is an $n \times n$ Hermitian matrix consisting of single-precision complex elements. A is stored in column-major format, and lda is the leading dimension of the two-dimensional array containing A.

Input

uplo	specifies whether the matrix data is stored in the upper or the lower
	triangular part of array A. If uplo == 'U' or 'u', only the upper
	triangular part of A is referenced. If uplo == 'L' or 'l', only the
	lower triangular part of A is referenced.

n the number of rows and columns of matrix A; n must be at least zero.

Input (continued)

alpha	single-precision scalar multiplier applied to
	$x * x^{H}$.
x	single-precision complex array of length at least
	(1+(n-1)*abs(incx)).
incx	the storage spacing between elements of x; incx must not be zero.
Α	single-precision complex array of dimensions (lda, n). If uplo == 'U' or 'u', A contains the upper triangular part of the Hermitian matrix, and the strictly lower triangular part is not referenced. If uplo == 'L' or 'l', A contains the lower triangular part of the Hermitian matrix, and the strictly upper triangular part is not referenced. The imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.
lda	leading dimension of the two-dimensional array containing A; lda must be at least $\max(1, n)$.

Output

A updated according to $A = alpha * x * x^H + A.$

Reference: http://www.netlib.org/blas/cher.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if n < 0 or incx == 0

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasCher2()

```
void
```

performs the Hermitian rank 2 operation

$$A = alpha * x * y^{H} + \overline{alpha} * y * x^{H} + A,$$

where alpha is a single-precision complex scalar, x and y are nelement single-precision complex vectors, and A is an $n \times n$ Hermitian matrix consisting of single-precision complex elements.

Input

uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array A. If $uplo == 'u'$ or $'u'$, only the upper triangular part of A may be referenced and the lower triangular part of A is inferred. If $uplo == 'L'$ or $'l'$, only the lower triangular part of A may be referenced and the upper triangular part of A is inferred.
n	the number of rows and columns of matrix A ; n must be at least zero.
alpha	single-precision complex scalar multiplier applied to
	$x * y^H$ and whose conjugate is applied to $y * x^H$.
х	single-precision array of length at least $(1 + (n-1) * abs(incx))$.
incx	the storage spacing between elements of x ; incx must not be zero.
У	single-precision array of length at least $(1 + (n-1) * abs(incy))$.
incy	the storage spacing between elements of y ; incy must not be zero.
А	single-precision complex array of dimensions (lda, n). If uplo == 'U' or 'u', A contains the upper triangular part of the Hermitian matrix, and the strictly lower triangular part is not referenced. If uplo == 'L' or 'l', A contains the lower triangular part of the Hermitian matrix, and the strictly upper triangular part is not referenced. The imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.
lda	leading dimension of A; $1da$ must be at least $max(1, n)$.

Output

A	updated according to
	$A = alpha * x * y^{H} + \overline{alpha} * y * x^{H} + A$

Reference: http://www.netlib.org/blas/cher2.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if $n < 0$, incx == 0, or incy == 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasChpmv()

void

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are single-precision complex scalars, and x and y are n-element single-precision complex vectors. A is a Hermitian $n \times n$ matrix that consists of single-precision complex elements and is supplied in packed form.

Input

mput	
uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array AP. If $uplo == 'U'$ or $'u'$, the upper triangular part of A is supplied in AP. If $uplo == 'L'$ or $'l'$, the lower triangular part of A is supplied in AP.
n	the number of rows and columns of matrix A; n must be at least zero.
alpha	single-precision complex scalar multiplier applied to A * \mathbf{x} .
AP	single-precision complex array with at least $(n*(n+1))/2$ elements. If $uplo=='u'$ or $'u'$, array AP contains the upper triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if $i <= j$, A[i,j] is stored in AP[$i+(j*(j+1)/2)$]. If $uplo=='L'$ or 'l', the array AP contains the lower triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if $i >= j$, A[i,j] is stored in AP[$i+((2*n-j+1)*j)/2$]. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero.
x	single-precision complex array of length at least $(1+(n-1)*abs(incx))$.
incx	storage spacing between elements of x; incx must not be zero.
beta	single-precision scalar multiplier applied to vector y.
У	single-precision array of length at least $(1+(n-1)*abs(incy))$. If beta is zero, y is not read.
incy	storage spacing between elements of y ; incy must not be zero.

Output

У	updated according to $y = alpha * A * x + beta * y$.
---	---

Reference: http://www.netlib.org/blas/chpmv.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZEDif CUBLAS library was not initializedCUBLAS_STATUS_INVALID_VALUEif n < 0, incx == 0, or incy == 0CUBLAS_STATUS_EXECUTION_FAILEDif function failed to launch on GPU
```

Function cublasChpr()

```
void
```

```
A = alpha * x * x^H + A,
```

where alpha is a single-precision scalar, x is an n-element single-precision complex vector, and A is an $n \times n$ Hermitian matrix consisting of single-precision complex elements that is supplied in packed form.

Input

```
specifies whether the matrix data is stored in the upper or the lower
triangular part of array AP. If uplo == 'U' or 'u', the upper triangular
part of A is supplied in AP. If uplo == 'L' or 'l', the lower triangular
part of A is supplied in AP.

the number of rows and columns of matrix A; n must be at least zero.

single-precision scalar multiplier applied to x * x<sup>H</sup>.

single-precision complex array of length at least
(1+(n-1) * abs(incx)).
```

Input (continued)

the storage spacing between elements of x; incx must not be zero. single-precision complex array with at least (n*(n+1))/2 elements. If uplo == 'U' or 'u', array AP contains the upper triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if i <= j, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo == 'L' or 'l', the array AP contains the lower triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if i >= j, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2]. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.

Output

A updated according to $A = alpha * x * x^{H} + A$.

Reference: http://www.netlib.org/blas/chpr.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if n < 0 or incx == 0

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasChpr2()

```
void
```

performs the Hermitian rank 2 operation

$$A = alpha * x * y^{H} + \overline{alpha} * y * x^{H} + A$$
,

where alpha is a single-precision complex scalar, x and y are nelement single-precision complex vectors, and A is an $n \times n$ Hermitian

matrix consisting of single-precision complex elements that is supplied in packed form.

Input

uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array A. If $uplo == 'U'$ or $'u'$, only the upper triangular part of A may be referenced and the lower triangular part of A is inferred. If $uplo == 'L'$ or $'l'$, only the lower triangular part of A may be referenced and the upper triangular part of A is inferred.
n	the number of rows and columns of matrix A; n must be at least zero.
alpha	single-precision complex scalar multiplier applied to
	$x * y^H$ and whose conjugate is applied to $y * x^H$.
x	single-precision complex array of length at least
	(1+(n-1)*abs(incx)).
incx	the storage spacing between elements of x; incx must not be zero.
У	single-precision complex array of length at least
	(1+(n-1)*abs(incy)).
incy	the storage spacing between elements of y; incy must not be zero.
АР	single-precision complex array with at least $(n*(n+1))/2$ elements. If $uplo=='u'$ or $'u'$, array AP contains the upper triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if $i <= j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If $uplo=='L'$ or 'l', the array AP contains the lower triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if $i >= j$, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2]. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.

Output

```
A updated according to A = alpha * x * y^H + \overline{alpha} * y * x^H + A
```

Reference: http://www.netlib.org/blas/chpr2.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0, incx == 0, or incy == 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasCtbmv()

void

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

x is an n-element single-precision complex vector, and A is an $n \times n$, unit or non-unit, upper or lower, triangular band matrix consisting of single-precision complex elements.

Input

specifies whether the matrix A is an upper or lower triangular band uplo matrix. If uplo == 'U' or 'u', A is an upper triangular band matrix. If uplo == 'L' or 'l', A is a lower triangular band matrix. trans specifies op(A). If trans == 'N' or 'n', op(A) = A. If trans == 'T' Or 't', $op(A) = A^{T}$. If trans == 'C', Or 'c', op(A) = A^{H} . diag specifies whether or not matrix A is unit triangular. If diag == 'U' or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is not assumed to be unit triangular. specifies the number of rows and columns of the matrix A; n must be n at least zero. k specifies the number of superdiagonals or subdiagonals. If uplo == 'U' or 'u', k specifies the number of superdiagonals. If uplo == 'L'

or 'l' k specifies the number of subdiagonals; k must at least be zero.

Input (continued)

A	single-precision complex array of dimension (lda, n). If uplo ==
	'U' or 'u', the leading (k+1)×n part of the array A must contain the
	upper triangular band matrix, supplied column by column, with the
	leading diagonal of the matrix in row k+1 of the array, the first
	superdiagonal starting at position 2 in row k, and so on. The top left
	kxk triangle of the array A is not referenced. If uplo == 'L' or 'l',
	the leading $(k+1) \times n$ part of the array A must contain the lower
	triangular band matrix, supplied column by column, with the leading
	diagonal of the matrix in row 1 of the array, the first subdiagonal
	starting at position 1 in row 2, and so on. The bottom right k×k
	triangle of the array is not referenced.
	· · ·

is the leading dimension of A; lda must be at least k+1.

x single-precision complex array of length at least (1+(n-1)*abs(incx)).

On entry, $\mathbf x$ contains the source vector. On exit, $\mathbf x$ is overwritten with the result vector.

incx specifies the storage spacing for elements of x; incx must not be zero.

Output

x updated according to x = op(A) * x.

Reference: http://www.netlib.org/blas/ctbmv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0, k < 0, or n < 0
CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasCtbsv()

void

solves one of the systems of equations

be at least zero.

```
op(A) * x = b,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

b and x are n-element vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular band matrix with k+1 diagonals.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

Input

uplo specifies whether the matrix is an upper or lower triangular band matrix: If uplo == 'U' or 'u', A is an upper triangular band matrix. If uplo == 'L' or 'l', A is a lower triangular band matrix. specifies op(A). If trans == 'N' or 'n', op(A) = A. trans If trans == 'T' Or 't', $op(A) = A^{T}$. If trans == 'C', Or 'c', op(A) = A^{H} . specifies whether A is unit triangular. If diag == 'U' or 'u', A is diag assumed to be unit triangular; that is, diagonal elements are not read and are assumed to be unity. If diag == 'N' or 'n', A is not assumed to be unit triangular. the number of rows and columns of matrix A: n must be at least zero. n In the current implementation n must not exceed 2035. k specifies the number of superdiagonals or subdiagonals. If uplo == 'U' or 'u', k specifies the number of superdiagonals. If

uplo == 'L' or 'l', k specifies the number of subdiagonals; k must

Input (continued)

A	single-precision complex array of dimension (lda, n). If uplo ==
	'U' or 'u', the leading (k+1) ×n part of the array A must contain the
	upper triangular band matrix, supplied column by column, with the
	leading diagonal of the matrix in row k+1 of the array, the first
	superdiagonal starting at position 2 in row k, and so on. The top left
	k×k triangle of the array A is not referenced. If uplo == 'L' or 'l',
	the leading $(k+1) \times n$ part of the array A must contain the lower
	triangular band matrix, supplied column by column, with the leading
	diagonal of the matrix in row 1 of the array, the first sub-diagonal
	starting at position 1 in row 2, and so on. The bottom right k×k
	triangle of the array is not referenced.
	δ , δ

x single-precision complex array of length at least (1+(n-1)*abs(incx)).

storage spacing between elements of x; incx must not be zero.

Output

incx

updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/ctbsv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZEDif CUBLAS library was not initializedCUBLAS_STATUS_INVALID_VALUEif incx == 0, n < 0, or n > 2035CUBLAS_STATUS_EXECUTION_FAILEDif function failed to launch on GPU

Function cublasCtpmv()

```
void
```

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A, op(A) = A^T, or op(A) = A^H;
```

x is an n-element single-precision complex vector, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix consisting of single-precision complex elements.

Input

uplo	specifies whether the matrix A is an upper or lower triangular matrix. If uplo == 'U' or 'u', A is an upper triangular matrix. If uplo == 'L' or 'l', A is a lower triangular matrix.
trans	specifies op(A). If trans == 'N' or 'n', op(A) = A.
	If trans == 'T' or 't', $op(A) = A^{T}$.
	If trans == 'C', Or 'c', $op(A) = A^{H}$.
diag	specifies whether or not matrix A is unit triangular. If diag == 'U' or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is not assumed to be unit triangular.
n	specifies the number of rows and columns of the matrix A; $\tt n$ must be at least zero.
АР	single-precision complex array with at least $(n*(n+1))/2$ elements. If $uplo == 'U'$ or $'u'$, the array AP contains the upper triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if $i <= j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If $uplo == 'L'$ or 'l', array AP contains the lower triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if $i >= j$, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2].
х	single-precision complex array of length at least $(1+(n-1)*abs(incx))$. On entry, x contains the source vector. On exit, x is overwritten with the result vector.
incx	specifies the storage spacing for elements of \mathtt{x} ; inc \mathtt{x} must not be zero.

Output

x updated according to x = op(A) * x.

Reference: http://www.netlib.org/blas/ctpmv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if incx == 0 or n < 0
```

CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough
	internal scratch vector memory
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasCtpsv()

void

solves one of the systems of equations

$$op(A) * x = b$$
,
where $op(A) = A$, $op(A) = A^{T}$, $or op(A) = A^{H}$;

b and x are n-element complex vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

Input

```
specifies whether the matrix is an upper or lower triangular matrix. If
uplo
         uplo == 'U' or 'u', A is an upper triangular matrix. If uplo == 'L'
         or 'l', A is a lower triangular matrix.
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
trans
         If trans == 'T' Or 't', op(A) = A^{T}.
         If trans == 'C', Or 'c', op(A) = A^{H}.
diag
         specifies whether A is unit triangular. If diag == 'U' or 'u', A is
         assumed to be unit triangular; that is, diagonal elements are not read
         and are assumed to be unity. If diag == 'N' or 'n', A is not assumed
         to be unit triangular.
         specifies the number of rows and columns of the matrix A; n must be
n
         at least zero. In the current implementation n must not exceed 2035.
```

Input (continued)

AP	single-precision complex array with at least $(n * (n + 1))/2$ elements.
	If uplo == 'U' or 'u', array AP contains the upper triangular matrix
	A, packed sequentially, column by column; that is, if $i \le j$, A[i,j] is
	stored in $AP[i+(j*(j+1)/2)]$. If uplo == 'L' or 'l', array AP
	contains the lower triangular matrix A, packed sequentially, column by
	column; that is, if $i >= j$, A[i,j] is stored in
	AP[i + ((2 * n - j + 1) * j)/2]. When diag == 'U' or 'u', the
	diagonal elements of A are not referenced and are assumed to be unity.
х	single-precision complex array of length at least
	(1+(n-1)*abs(incx)).
incx	storage spacing between elements of x; incx must not be zero.

Output

updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/ctpsv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZEDif CUBLAS library was not initializedCUBLAS_STATUS_INVALID_VALUEif incx == 0, n < 0, or n > 2035CUBLAS_STATUS_EXECUTION_FAILEDif function failed to launch on GPU
```

Function cublasCtrmv()

void

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A, op(A) = A^T, or op(A) = A^H;
```

x is an n-element single-precision complex vector; and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix consisting of single-precision complex elements.

Input	
-------	--

```
uplo
         specifies whether the matrix A is an upper or lower triangular matrix.
         If uplo == 'U' or 'u', A is an upper triangular matrix.
         If uplo == 'L' or 'l', A is an lower triangular matrix.
trans
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
         If trans == 'T' Or 't', op(A) = A^{T}.
         If trans == 'C' or 'c', op(A) = A^{H}.
diag
         specifies whether or not A is a unit triangular matrix. If diag == 'U'
         or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is
         not assumed to be unit triangular.
         specifies the number of rows and columns of the matrix A; n must be
n
         at least zero.
         single-precision complex array of dimensions (lda, n). If uplo ==
Α
         'U' or 'u', the leading n×n upper triangular part of the array A must
         contain the upper triangular matrix, and the strictly lower triangular
         part of A is not referenced. If uplo == 'L' or 'l', the leading nxn
         lower triangular part of the array A must contain the lower triangular
         matrix, and the strictly upper triangular part of A is not referenced.
         When diag == 'U' or 'u', the diagonal elements of A are not
         referenced either, but are assumed to be unity.
lda
         leading dimension of A; lda must be at least max(1, n).
         single-precision complex array of length at least
х
         (1+(n-1)*abs(incx)). On entry, x contains the source vector.
         On exit. x is overwritten with the result vector.
incx
         the storage spacing between elements of x; incx must not be zero.
```

Output

x updated according to x = op(A) * x.

Reference: http://www.netlib.org/blas/ctrmv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if incx == 0 or n < 0

Error Status (continued)

CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough
	internal scratch vector memory
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasCtrsv()

void

solves a system of equations

```
op(A) * x = b,
where op(A) = A, op(A) = A^T, or op(A) = A^H;
```

b and x are n-element single-precision complex vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix consisting of single-precision elements. Matrix A is stored in column-major format, and 1da is the leading dimension of the two-dimensional array containing A.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

Input

```
specifies whether the matrix data is stored in the upper or the lower
uplo
         triangular part of array A. If uplo == 'U' or 'u', only the upper
         triangular part of A may be referenced. If uplo == 'L' or 'l', only
         the lower triangular part of A may be referenced.
trans
         specifies op (A). If trans == 'N' or 'n', op (A) = A.
         If trans == 'T' Or 't', op(A) = A^{T}.
         If trans == 'C' Or 'c', op(A) = A^{H}.
diag
         specifies whether or not A is a unit triangular matrix.
         If diag == 'U' or 'u', A is assumed to be unit triangular.
         If diag == 'N' or 'n', A is not assumed to be unit triangular.
         specifies the number of rows and columns of the matrix A; n must be
n
         at least zero.
```

Input (continued)

A	single-precision complex array of dimensions (lda, n). If uplo ==
	'U' or 'u', A contains the upper triangular part of the symmetric
	matrix, and the strictly lower triangular part is not referenced. If uplo
	== 'L' or 'l', A contains the lower triangular part of the symmetric
	matrix, and the strictly upper triangular part is not referenced.
lda	leading dimension of the two-dimensional array containing A;
	lda must be at least $max(1, n)$.
x	single-precision complex array of length at least
	(1+(n-1)*abs(incx)). On entry, x contains the n-element, right-
	hand-side vector b . On exit, it is overwritten with solution vector x .
incx	the storage spacing between elements of x ; incx must not be zero.

Output

updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/ctrsv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0 or n < 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

CHAPTER

4

Double-Precision BLAS2 Functions

The Level 2 Basic Linear Algebra Subprograms (BLAS2) are functions that perform matrix-vector operations. The CUBLAS implementations of double-precision BLAS2 functions are described in these sections:

- □ "Double-Precision BLAS2 Functions" on page 122
- □ "Double-Precision Complex BLAS2 functions" on page 146

Double-Precision BLAS2 Functions

Note: Double-precision functions are only supported on GPUs with double-precision hardware.

The double-precision BLAS2 functions are as follows:

- □ "Function cublasDgbmv()" on page 123
- □ "Function cublasDgemv()" on page 124
- □ "Function cublasDger()" on page 126
- □ "Function cublasDsbmv()" on page 127
- □ "Function cublasDspmv()" on page 129
- □ "Function cublasDspr()" on page 130
- □ "Function cublasDspr2()" on page 131
- □ "Function cublasDsymv()" on page 132
- □ "Function cublasDsyr()" on page 134
- □ "Function cublasDsyr2()" on page 135
- □ "Function cublasDtbmv()" on page 136
- □ "Function cublasDtbsv()" on page 138
- □ "Function cublasDtpmv()" on page 140
- □ "Function cublasDtpsv()" on page 141
- □ "Function cublasDtrmv()" on page 142
- □ "Function cublasDtrsv()" on page 144

Function cublasDgbmv()

void

performs one of the matrix-vector operations

```
y = alpha * op(A) * x + beta * y,
where op(A) = A or op(A) = A<sup>T</sup>,
```

alpha and beta are double-precision scalars, and x and y are double-precision vectors. A is an $m \times n$ band matrix consisting of double-precision elements with kl subdiagonals and ku superdiagonals.

Input

```
specifies op(A). If trans == 'N' or 'n', op(A) = A.
trans
         If trans == 'T', 't', 'C', Or 'c', op(A) = A^{T}.
         specifies the number of rows of matrix A; m must be at least zero.
m
         specifies the number of columns of matrix A; n must be at least zero.
n
k1
         specifies the number of subdiagonals of matrix A; kl must be at least
         zero.
ku
         specifies the number of superdiagonals of matrix A; ku must be at
         least zero.
alpha
         double-precision scalar multiplier applied to op(A).
Α
         double-precision array of dimensions (lda, n). The leading
         (k1+ku+1)×n part of the array A must contain the band matrix A,
         supplied column by column, with the leading diagonal of the matrix in
         row (ku+1) of the array, the first superdiagonal starting at position 2 in
         row ku, the first subdiagonal starting at position 1 in row (ku+2), and
         so on. Elements in the array A that do not correspond to elements in
         the band matrix (such as the top left ku×ku triangle) are not
         referenced.
lda
         leading dimension A; lda must be at least (k1+ku+1).
x
         double-precision array of length at least (1 + (n-1) * abs(incx)) if
         trans == 'N' or 'n', else at least (1 + (m-1) * abs(incx)).
         specifies the increment for the elements of x; incx must not be zero.
incx
```

Input (continued)

beta	double-precision scalar multiplier applied to vector y. If beta is zero, y is not read.
У	double-precision array of length at least (1 + (m - 1) * abs(incy)) if
	trans == 'N' or 'n', else at least $(1+(n-1)*abs(incy))$. If
	beta is zero, y is not read.
incy	on entry, incy specifies the increment for the elements of y ; incy must not be zero.

Output

```
y updated according to y = alpha * op(A) * x + beta * y.
```

Reference: http://www.netlib.org/blas/dgbmv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0, n < 0, incx == 0, or
	incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that
	does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDgemv()

void

performs one of the matrix-vector operations

```
y = alpha * op(A) * x + beta * y,
where op(A) = A or op(A) = A^{T},
```

alpha and beta are double-precision scalars, and x and y are double-precision vectors. A is an $m \times n$ matrix consisting of double-precision

elements. Matrix $\mathtt A$ is stored in column-major format, and $\mathtt 1da$ is the leading dimension of the two-dimensional array in which $\mathtt A$ is stored.

Input

```
trans
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
         If trans == 'T', 't', 'C', Or 'c', Op(A) = A^{T}.
         specifies the number of rows of matrix A; m must be at least zero.
m
         specifies the number of columns of matrix A; n must be at least zero.
n
         double-precision scalar multiplier applied to op(A).
alpha
Α
         double-precision array of dimensions (lda, n) if trans == 'N' or
         'n'. of dimensions (lda. m) otherwise: lda must be at least
         \max(1, m) if trans == 'N' or 'n' and at least \max(1, n) otherwise.
lda
         leading dimension of two-dimensional array used to store matrix A.
         double-precision array of length at least (1 + (n-1) * abs(incx)) if
         trans == 'N' or 'n', else at least (1 + (m-1) * abs(incx)).
         specifies the storage spacing for elements of x; incx must not be zero.
incx
         double-precision scalar multiplier applied to vector y. If beta is zero,
beta
         y is not read.
У
         double-precision array of length at least (1 + (m-1) * abs(incy)) if
         trans == 'N' or 'n', else at least (1 + (n-1) * abs(incy)).
incy
         the storage spacing between elements of y; incy must not be zero.
```

Output

```
y updated according to y = alpha * op(A) * x + beta * y.
```

Reference: http://www.netlib.org/blas/dgemv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized if m < 0, n < 0, incx == 0, or incy == 0

CUBLAS_STATUS_ARCH_MISMATCH if function invoked on device that does not support double precision if function failed to launch on GPU
```

Function cublasDger()

void

performs the symmetric rank 1 operation

```
A = alpha * x * y^T + A,
```

where alpha is a double-precision scalar, x is an m-element double-precision vector, y is an n-element double-precision vector, and A is an mxn matrix consisting of double-precision elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array used to store A.

Input

•	
m	specifies the number of rows of the matrix A; m must be at least zero.
n	specifies the number of columns of matrix A ; n must be at least zero.
alpha	double-precision scalar multiplier applied to $\mathbf{x} * \mathbf{y}^{T}$.
x	double-precision array of length at least $(1 + (m-1) * abs(incx))$.
incx	the storage spacing between elements of x ; incx must not be zero.
У	double-precision array of length at least $(1 + (n-1) * abs(incy))$.
incy	the storage spacing between elements of y; incy must not be zero.
A	double-precision array of dimensions (lda, n).
lda	leading dimension of two-dimensional array used to store matrix A.

Output

```
A updated according to A = alpha * x * y^T + A.
```

Reference: http://www.netlib.org/blas/dger.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized 

CUBLAS_STATUS_INVALID_VALUE if m < 0, n < 0, incx == 0, or incy == 0
```

Error Status (continued)

CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that
	does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDsbmv()

void

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are double-precision scalars, and x and y are n-element double-precision vectors. A is an $n \times n$ symmetric band matrix consisting of double-precision elements, with k superdiagonals and the same number of subdiagonals.

Input

- specifies whether the upper or lower triangular part of the symmetric band matrix A is being supplied. If uplo == 'U' or 'u', the upper triangular part is being supplied. If uplo == 'L' or 'l', the lower triangular part is being supplied.
- n specifies the number of rows and the number of columns of the symmetric matrix A; n must be at least zero.
- specifies the number of superdiagonals of matrix \mathtt{A} . Since the matrix is symmetric, this is also the number of subdiagonals; \mathtt{k} must be at least zero.
- alpha double-precision scalar multiplier applied to A \star x.

Input (continued)

A	double-precision array of dimensions (lda, n). When uplo == 'U'
	or 'u', the leading (k+1)×n part of array A must contain the upper
	triangular band of the symmetric matrix, supplied column by column,
	with the leading diagonal of the matrix in row k+1 of the array, the
	first superdiagonal starting at position 2 in row k, and so on. The top
	left kxk triangle of the array A is not referenced. When uplo == 'L'
	or 'l', the leading $(k+1) \times n$ part of the array A must contain the
	lower triangular band part of the symmetric matrix, supplied column
	by column, with the leading diagonal of the matrix in row 1 of the
	array, the first subdiagonal starting at position 1 in row 2, and so on.
	The bottom right kxk triangle of the array A is not referenced.
lda	leading dimension of A; 1da must be at least $k+1$.
x	double-precision array of length at least $(1+(n-1)*abs(incx))$.
incx	storage spacing between elements of x; incx must not be zero.
beta	double-precision scalar multiplier applied to vector y.
У	double-precision array of length at least $(1 + (n-1) * abs(incy))$.
	If beta is zero, y is not read.
incy	storage spacing between elements of y; incy must not be zero.
	0 1 0

Output

y updated according to $y = alpha * A * x + beta * y$.	
---	--

Reference: http://www.netlib.org/blas/dsbmv.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if k < 0, n < 0, incx == 0, or incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDspmv()

void

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are double-precision scalars, and x and y are n-element double-precision vectors. A is a symmetric $n \times n$ matrix that consists of double-precision elements and is supplied in packed form.

Input

uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array AP. If $uplo == 'U'$ or 'u', the upper triangular part of A is supplied in AP. If $uplo == 'L'$ or 'l', the lower triangular part of A is supplied in AP.
n	the number of rows and columns of matrix A; n must be at least zero.
alpha	double-precision scalar multiplier applied to A \star x.
АР	double-precision array with at least $(n*(n+1))/2$ elements. If $uplo=='U'$ or $'u'$, array AP contains the upper triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if $i <= j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If $uplo=='L'$ or 'l', the array AP contains the lower triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if $i >= j$, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2].
x	double-precision array of length at least $(1 + (n-1) * abs(incx))$.
incx	storage spacing between elements of x; incx must not be zero.
beta	double-precision scalar multiplier applied to vector y.
У	double-precision array of length at least $(1+(n-1)*abs(incy))$. If beta is zero, y is not read.
incy	storage spacing between elements of y ; incy must not be zero.
Output	

Reference: http://www.netlib.org/blas/dspmv.f

updated according to y = alpha * A * x + beta * y.

Error status for this function can be retrieved via cublasGetError() .
Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0, incx == 0, or incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDspr()

```
A = alpha * x * x^{T} + A,
```

performs the symmetric rank 1 operation

where alpha is a double-precision scalar, and x is an n-element double-precision vector. A is a symmetric $n \times n$ matrix that consists of double-precision elements and is supplied in packed form.

Input

uplo	specifies whether the matrix data is stored in the upper or the lower
	triangular part of array AP. If $uplo == 'U'$ or 'u', the upper triangular
	part of A is supplied in AP. If uplo == 'L' or 'l', the lower triangular
	part of A is supplied in AP.
n	the number of rows and columns of matrix A; n must be at least zero.
alpha	double-precision scalar multiplier applied to $\mathbf{x} \star \mathbf{x}^{\scriptscriptstyle T}$.
x	double-precision array of length at least $(1+(n-1)*abs(incx))$.
incx	storage spacing between elements of x; incx must not be zero.
AP	double-precision array with at least $(n * (n + 1))/2$ elements. If
	uplo == 'U' or 'u', array AP contains the upper triangular part of the
	symmetric matrix A, packed sequentially, column by column; that is, if
	$i \le j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo == 'L'
	or 'l', the array AP contains the lower triangular part of the
	symmetric matrix A, packed sequentially, column by column; that is, if
	i >= j, A[i,j] is stored in AP[i+((2 * n - j + 1) * j)/2].

Output

A updated according to $A = alpha * x * x^T + A$.	
--	--

Reference: http://www.netlib.org/blas/dspr.f

Error status for this function can be retrieved via cublasGetError().

Frror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0 or incx == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDspr2()

void

performs the symmetric rank 2 operation

$$A = alpha * x * y^{T} + alpha * y * x^{T} + A,$$

where alpha is a double-precision scalar, and x and y are n-element double-precision vectors. A is a symmetric $n \times n$ matrix that consists of double-precision elements and is supplied in packed form.

Input

```
specifies whether the matrix data is stored in the upper or the lower
uplo
         triangular part of array A. If uplo == 'U' or 'u', only the upper
         triangular part of A may be referenced and the lower triangular part of
         A is inferred. If uplo == 'L' or 'l', only the lower triangular part of
         A may be referenced and the upper triangular part of A is inferred.
         the number of rows and columns of matrix A: n must be at least zero.
n
alpha
         double-precision scalar multiplier applied to x * y^T and y * x^T.
х
         double-precision array of length at least (1 + (n-1) * abs(incx)).
incx
         storage spacing between elements of x; incx must not be zero.
         double-precision array of length at least (1 + (n-1) * abs(incy)).
У
```

Input (continued)

incy	storage spacing between elements of y; incy must not be zero.
AP	double-precision array with at least $(n * (n + 1))/2$ elements. If
	uplo $==$ 'U' or 'u', array AP contains the upper triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if
	$i \le j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo == 'L'
	or 'l', the array AP contains the lower triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if
	i >= j, A[i,j] is stored in AP[i + ((2 * n - j + 1) * j)/2].

Output

```
updated according to A = alpha * x * y^T + alpha * y * x^T + A.
```

Reference: http://www.netlib.org/blas/dspr2.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0, incx == 0, or incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	

Function cublasDsymv()

```
void
```

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are double-precision scalars, and x and y are n-element double-precision vectors. A is a symmetric n×n matrix that

consists of double-precision elements and is stored in either upper or lower storage mode.

Input

uplo	specifies whether the upper or lower triangular part of the array A is
	referenced. If $uplo == 'U'$ or 'u', the symmetric matrix A is stored in
	upper storage mode; that is, only the upper triangular part of A is
	referenced while the lower triangular part of A is inferred. If uplo ==
	'L' or 'l', the symmetric matrix A is stored in lower storage mode; that is, only the lower triangular part of A is referenced while the upper
	triangular part of A is inferred.
n	specifies the number of rows and the number of columns of the
	symmetric matrix A; n must be at least zero.
alpha	double-precision scalar multiplier applied to A \star x.
A	double-precision array of dimensions (lda, n). If uplo == 'U' or
	'u', the leading $n \times n$ upper triangular part of the array A must contain
	the upper triangular part of the symmetric matrix, and the strictly
	lower triangular part of A is not referenced. If uplo == 'L' or 'l',
	the leading n×n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix, and the strictly upper
	triangular part of A is not referenced.
lda	leading dimension of A; $1 ext{da}$ must be at least $max(1, n)$.
x	double-precision array of length at least $(1 + (n-1) * abs(incx))$.
incx	storage spacing between elements of x; incx must not be zero.
beta	double-precision scalar multiplier applied to vector y.
У	single-precision array of length at least $(1 + (n-1) * abs(incy))$.
	If beta is zero, y is not read.
incy	storage spacing between elements of y ; incy must not be zero.

Output

y updated according to y = alpha * A * x + beta * y.

Reference: http://www.netlib.org/blas/dsymv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if n < 0, incx == 0, or incy == 0

CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that
	does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDsyr()

void

performs the symmetric rank 1 operation

$$A = alpha * x * x^T + A$$
,

where alpha is a double-precision scalar, x is an n-element double-precision vector, and A is an $n \times n$ symmetric matrix consisting of double-precision elements. A is stored in column-major format, and lda is the leading dimension of the two-dimensional array containing A.

Input

uplo	specifies whether the matrix data is stored in the upper or the lower
	triangular part of array A. If uplo == 'U' or 'u', only the upper
	triangular part of A is referenced. If $uplo == 'L'$ or 'l', only the lower triangular part of A is referenced.
n	the number of rows and columns of matrix \mathtt{A} ; \mathtt{n} must be at least zero.
alpha	double-precision scalar multiplier applied to $\mathbf{x}~\star~\mathbf{x}^{\scriptscriptstyle T}$.
x	double-precision array of length at least $(1+(n-1)*abs(incx))$.
incx	the storage spacing between elements of x; incx must not be zero.
A	double-precision array of dimensions (lda, n). If uplo == 'U' or
	'u', A contains the upper triangular part of the symmetric matrix, and
	the strictly lower triangular part is not referenced. If uplo == 'L' or
	'1', A contains the lower triangular part of the symmetric matrix, and
	the strictly upper triangular part is not referenced.
lda	leading dimension of the two-dimensional array containing A;
	lda must be at least $max(1, n)$.

Output

Reference: http://www.netlib.org/blas/dsyr.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0 or incx == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDsyr2()

void

performs the symmetric rank 2 operation

$$A = alpha * x * y^T + alpha * y * x^T + A$$
,

where alpha is a double-precision scalar, x and y are n-element double-precision vectors, and A is an $n \times n$ symmetric matrix consisting of double-precision elements.

uplo	specifies whether the matrix data is stored in the upper or the lower
	triangular part of array A. If uplo == 'U' or 'u', only the upper
	triangular part of A is referenced and the lower triangular part of A is
	inferred. If $uplo == 'L'$ or 'l', only the lower triangular part of A is referenced and the upper triangular part of A is inferred.
n	the number of rows and columns of matrix A ; n must be at least zero.
alpha	double-precision scalar multiplier applied to $\mathbf{x} \star \mathbf{y}^{\scriptscriptstyle T}$ and $\mathbf{y} \star \mathbf{x}^{\scriptscriptstyle T}$.
x	double-precision array of length at least $(1 + (n-1) * abs(incx))$.
incx	storage spacing between elements of x; incx must not be zero.
У	double-precision array of length at least $(1 + (n-1) * abs(incy))$.

incy	storage spacing between elements of y; incy must not be zero.	
A	double-precision array of dimensions (lda, n). If $uplo == 'U'$ or	
	'u', A contains the upper triangular part of the symmetric matrix, and	
	the strictly lower triangular part is not referenced. If uplo == 'L' or	
	'1', A contains the lower triangular part of the symmetric matrix, and	
	the strictly upper triangular part is not referenced.	
lda	leading dimension of A; 1da must be at least max(1, n).	

Output

A updated according to $A = alpha * x * y^T + alpha * y * x^T + A$.

Reference: http://www.netlib.org/blas/dsyr2.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if $n < 0$, incx == 0, or incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDtbmv()

void

```
cublasDtbmv (char uplo, char trans, char diag, int n,
                int k, const double *A, int lda, double *x,
                int incx)
```

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A or op(A) = A^{T},
```

x is an n-element double-precision vector, and A is an $n \times n$, unit or non-unit, upper or lower, triangular band matrix consisting of double-precision elements.

Input	
uplo	specifies whether the matrix A is an upper or lower triangular band matrix. If $uplo == 'u'$ or $'u'$, A is an upper triangular band matrix. If $uplo == 'L'$ or $'l'$, A is a lower triangular band matrix.
trans	specifies $op(A)$. If trans == 'N' or 'n', $op(A) = A$.
	If trans == 'T', 't', 'C', Or 'c', $op(A) = A^T$.
diag	specifies whether or not matrix A is unit triangular. If $\mathtt{diag} == '\mathtt{U}'$ or $'\mathtt{u}'$, A is assumed to be unit triangular. If $\mathtt{diag} == '\mathtt{N}'$ or $'\mathtt{n}'$, A is not assumed to be unit triangular.
n	specifies the number of rows and columns of the matrix A; $\tt n$ must be at least zero.
k	specifies the number of superdiagonals or subdiagonals. If $uplo == 'u'$ or 'u', k specifies the number of superdiagonals. If $uplo == 'L'$ or 'l' k specifies the number of subdiagonals; k must at least be zero.
A	double-precision array of dimension (lda, n). If uplo == 'U' or 'u', the leading (k+1)×n part of the array A must contain the upper triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row k+1 of the array, the first superdiagonal starting at position 2 in row k, and so on. The top left k×k triangle of the array A is not referenced. If uplo == 'L' or 'l', the leading (k+1)×n part of the array A must contain the lower triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first subdiagonal starting at position 1 in row 2, and so on. The bottom right k×k triangle of the array is not referenced.
lda	is the leading dimension of A; 1da must be at least k+1.
x	double-precision array of length at least $(1 + (n-1) * abs(incx))$. On entry, x contains the source vector. On exit, x is overwritten with the result vector.
incx	specifies the storage spacing for elements of x; incx must not be zero.
Output	
x	updated according to $x = op(A) * x$.
	-

Reference: http://www.netlib.org/blas/dtbmv.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == $0, k < 0, or n < 0$
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDtbsv()

```
void
```

solves one of the systems of equations

```
op(A) * x = b,
where op(A) = A \text{ or } op(A) = A^{T},
```

b and x are n-element vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular band matrix with k+1 diagonals.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

Input

uplo	specifies whether the matrix is an upper or lower triangular band matrix: If uplo == 'U' or 'u', A is an upper triangular band matrix. If uplo == 'L' or 'l', A is a lower triangular band matrix.
trans	specifies op(A). If trans == 'N' or 'n', op(A) = A.
	If trans == 'T', 't', 'C', or 'c', op(A) = A^{T} .
diag	specifies whether A is unit triangular. If $\mathtt{diag} == '\mathtt{U}'$ or $'\mathtt{u}'$, A is assumed to be unit triangular; that is, diagonal elements are not read and are assumed to be unity. If $\mathtt{diag} == '\mathtt{N}'$ or $'\mathtt{n}'$, A is not assumed to be unit triangular.
n	the number of rows and columns of matrix A; n must be at least zero. In the current implementation n must not exceed 2035.

Input (continued)

- k specifies the number of superdiagonals or subdiagonals. If uplo == 'U' or 'u', k specifies the number of superdiagonals. If uplo == 'L' or 'l', k specifies the number of subdiagonals; k must be at least zero.
- double-precision array of dimension (lda, n). If uplo == 'U' or 'u', the leading (k+1)×n part of the array A must contain the upper triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row k+1 of the array, the first superdiagonal starting at position 2 in row k, and so on. The top left k×k triangle of the array A is not referenced. If uplo == 'L' or 'l', the leading (k+1)×n part of the array A must contain the lower triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k×k triangle of the array is not referenced.
- double-precision array of length at least (1 + (n-1) * abs(incx)). incx storage spacing between elements of x; incx must not be zero.

Output

x updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/dtbsv.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0, n < 0, or n > 2035
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDtpmv()

void

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A or op(A) = A^{T},
```

x is an n-element double-precision vector, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix consisting of double-precision elements.

Input

uplo	specifies whether the matrix A is an upper or lower triangular matrix. If uplo == 'U' or 'u', A is an upper triangular matrix. If uplo == 'L' or 'l', A is a lower triangular matrix.
trans	specifies $op(A)$. If trans == 'N' or 'n', $op(A) = A$.
	If trans == 'T', 't', 'C', Or 'c', $op(A) = A^{T}$.
diag	specifies whether or not matrix A is unit triangular. If diag == 'U' or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is not assumed to be unit triangular.
n	specifies the number of rows and columns of the matrix A; $\tt n$ must be at least zero.
АР	double-precision array with at least $(n * (n + 1))/2$ elements. If $uplo == 'U'$ or 'u', the array AP contains the upper triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if $i <= j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If $uplo == 'L'$ or 'l', array AP contains the lower triangular part of the symmetric matrix A, packed sequentially, column by column; that is, if $i >= j$, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2].
х	double-precision array of length at least $(1+(n-1)*abs(incx))$. On entry, x contains the source vector. On exit, x is overwritten with the result vector.
incx	specifies the storage spacing for elements of \mathtt{x} ; incx must not be zero.

Output

x updated according to x = op(A) * x.

Reference: http://www.netlib.org/blas/dtpmv.f

Error status for this function can be retrieved via **cublasGetError()**.

Fror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0 or n < 0
CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDtpsv()

void

solves one of the systems of equations

$$op(A) * x = b,$$

where $op(A) = A or op(A) = A^{T},$

b and x are n-element vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

```
specifies whether the matrix is an upper or lower triangular matrix. If
uplo == 'U' or 'u', A is an upper triangular matrix. If uplo == 'L'
or 'l', A is a lower triangular matrix.

trans specifies op(A). If trans == 'N' or 'n', op(A) = A.
    If trans == 'T', 't', 'C', or 'c', op(A) = A<sup>T</sup>.

diag specifies whether A is unit triangular. If diag == 'U' or 'u', A is assumed to be unit triangular; that is, diagonal elements are not read and are assumed to be unity. If diag == 'N' or 'n', A is not assumed to be unit triangular.
```

Input (continued)

n	specifies the number of rows and columns of the matrix A; n must be
	at least zero. In the current implementation n must not exceed 2035.

double-precision array with at least (n * (n+1))/2 elements. If plo = "U" or "u", array AP contains the upper triangular matrix A, packed sequentially, column by column; that is, if i <= j, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If plo = "L" or "l", array AP contains the lower triangular matrix A, packed sequentially, column by column; that is, if i >= j, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2]. When plo = "U" or "u", the diagonal elements of A are not referenced and are assumed to be unity.

** double-precision array of length at least (1+(n-1)*abs(incx)).

incx storage spacing between elements of x; incx must not be zero.

Output

updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/dtpsv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0, n < 0, or n > 2035
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDtrmv()

```
void
```

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A \text{ or } op(A) = A^{T},
```

x is an n-element double-precision vector, and A is an n×n, unit or nonunit, upper or lower, triangular matrix consisting of double-precision elements.

Input

uplo	specifies whether the matrix A is an upper or lower triangular matrix. If uplo == 'U' or 'u', A is an upper triangular matrix. If uplo == 'L' or 'l', A is an lower triangular matrix.
trans	specifies op(A). If trans == 'N' Or 'n', op(A) = A.
	If trans == 'T', 't', 'C', Or 'c', $op(A) = A^{T}$.
diag	specifies whether or not A is a unit triangular matrix. If $\mathtt{diag} == '\mathtt{U}'$ or $'\mathtt{u}'$, A is assumed to be unit triangular. If $\mathtt{diag} == '\mathtt{N}'$ or $'\mathtt{n}'$, A is not assumed to be unit triangular.
n	specifies the number of rows and columns of the matrix A; n must be at least zero.
Α	double-precision array of dimensions (lda, n). If uplo == 'U' or 'u', the leading n×n upper triangular part of the array a must contain the upper triangular matrix, and the strictly lower triangular part of a is not referenced. If uplo == 'L' or 'l', the leading n×n lower triangular part of the array a must contain the lower triangular matrix, and the strictly upper triangular part of a is not referenced. When diag == 'U' or 'u', the diagonal elements of a are not referenced either, but are assumed to be unity.
lda	leading dimension of A; lda must be at least max(1, n).
х	double-precision array of length at least $(1+(n-1)*abs(incx))$. On entry, x contains the source vector. On exit, x is overwritten with the result vector.
incx	the storage spacing between elements of x ; incx must not be zero.
Output	

```
updated according to x = op(A) * x.
```

Reference: http://www.netlib.org/blas/dtrmv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
if CUBLAS library was not initialized
CUBLAS_STATUS_NOT_INITIALIZED
CUBLAS_STATUS_INVALID_VALUE
                                    if incx == 0 or n < 0
```

Error Status (continued)		
CUBLAS_STATUS_ALLOC_FAILED		

CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDtrsv()

void

```
cublasDtrsv (char uplo, char trans, char diag, int n,
             const double *A, int lda, double *x,
             int incx)
```

solves a system of equations

```
op(A) * x = b,
where op(A) = A or op(A) = A^{T},
```

b and x are n-element double-precision vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix consisting of doubleprecision elements. Matrix A is stored in column-major format, and 1da is the leading dimension of the two-dimensional array containing A.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

Input

```
uplo
         specifies whether the matrix data is stored in the upper or the lower
         triangular part of array A. If uplo == 'U' or 'u', only the upper
         triangular part of A may be referenced. If uplo == 'L' or 'l', only
         the lower triangular part of A may be referenced.
trans
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
         If trans == 'T', 't', 'C', Or 'c', op(A) = A^{T}.
diag
         specifies whether or not A is a unit triangular matrix.
         If diag == 'U' or 'u', A is assumed to be unit triangular.
         If diag == 'N' or 'n', A is not assumed to be unit triangular.
         specifies the number of rows and columns of the matrix A; n must be
n
         at least zero.
```

Input (continued)

A	double-precision array of dimensions (lda, n). If uplo == 'U' or	
	'u', A contains the upper triangular part of the symmetric matrix, and	
	the strictly lower triangular part is not referenced. If uplo == 'L' or	
	'l', A contains the lower triangular part of the symmetric matrix, and	
	the strictly upper triangular part is not referenced.	
lda	leading dimension of the two-dimensional array containing A;	
	lda must be at least $max(1, n)$.	
х	double-precision array of length at least $(1 + (n-1) * abs(incx))$.	
	On entry, x contains the n-element, right-hand-side vector b. On exit,	
	it is overwritten with the solution vector \mathbf{x} .	
incx	the storage spacing between elements of x ; incx must not be zero.	

Output

updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/dtrsv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0 or n < 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Double-Precision Complex BLAS2 functions

Note: Double-precision functions are only supported on GPUs with double-precision hardware.

Two double-precision complex BLAS2 functions are implemented:

- □ "Function cublasZgbmv()" on page 147
- □ "Function cublasZgemv()" on page 149
- □ "Function cublasZgerc()" on page 150
- □ "Function cublasZgeru()" on page 151
- □ "Function cublasZhbmv()" on page 153
- □ "Function cublasZhemv()" on page 155
- □ "Function cublasZher()" on page 156
- □ "Function cublasZher2()" on page 158
- □ "Function cublasZhpmv()" on page 159
- □ "Function cublasZhpr()" on page 161
- □ "Function cublasZhpr2()" on page 162
- □ "Function cublasZtbmv()" on page 163
- □ "Function cublasZtbsv()" on page 165
- □ "Function cublasZtpmv()" on page 167
- □ "Function cublasZtpsv()" on page 168
- □ "Function cublasZtrmv()" on page 170
- □ "Function cublasZtrsv()" on page 171

Function cublasZgbmv()

void

performs one of the matrix-vector operations

```
y = alpha * op(A) * x + beta * y, where

op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

alpha and beta are double-precision complex scalars, and x and y are double-precision complex vectors. A is an $m \times n$ band matrix consisting of double-precision complex elements with k1 subdiagonals and ku superdiagonals.

```
trans
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
         If trans == 'T' Or 't', op(A) = A^{T}.
         If trans == 'C', Or'c', op(A) = A^{H}.
         specifies the number of rows of matrix A; m must be at least zero.
m
         specifies the number of columns of matrix A; n must be at least zero.
n
         specifies the number of subdiagonals of matrix A; kl must be at least
k1
         zero.
         specifies the number of superdiagonals of matrix A; ku must be at
ku
         least zero.
alpha
         double-precision complex scalar multiplier applied to op(A).
         double-precision complex array of dimensions (lda, n). The leading
Α
         (k1+ku+1)×n part of the array A must contain the band matrix A,
         supplied column by column, with the leading diagonal of the matrix in
         row (ku+1) of the array, the first superdiagonal starting at position 2 in
         row ku, the first subdiagonal starting at position 1 in row (ku+2), and
         so on. Elements in the array A that do not correspond to elements in
         the band matrix (such as the top left ku×ku triangle) are not
         referenced.
lda
         leading dimension A; lda must be at least (k1+ku+1).
```

Input (continued)

х	double-precision complex array of length at least	
	(1+(n-1)*abs(incx)) if trans == 'N' Or 'n', else at least	
	(1+(m-1)*abs(incx)).	
incx	specifies the increment for the elements of x ; incx must not be zero.	
beta	double-precision complex scalar multiplier applied to vector \mathbf{y} . If beta is zero, \mathbf{y} is not read.	
У	double-precision complex array of length at least	
	(1+(m-1)*abs(incy)) if trans == 'N' Or 'n', else at least	
	(1+(n-1)*abs(incy)). If beta is zero, y is not read.	
incy	on entry, incy specifies the increment for the elements of y ; incy must not be zero.	

Output

y updated according to y = alpha * op(A) * x + beta * y.

Reference: http://www.netlib.org/blas/zgbmv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0, n < 0, incx == 0, or
	incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that
	does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZgemv()

performs one of the matrix-vector operations

```
y = alpha * op(A) * x + beta * y,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

alpha and beta are double-precision complex scalars; and x and y are double-precision complex vectors. A is an $m \times n$ matrix consisting of double-precision complex elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array in which A is stored.

```
specifies op(A). If trans == 'N' Or 'n', op(A) = A.
trans
         If trans == 'T' Or 't', op(A) = A^{T}.
         If trans == 'C' Or 'c', op(A) = A^{H}.
m
         specifies the number of rows of matrix A; m must be at least zero.
n
         specifies the number of columns of matrix A; n must be at least zero.
         double-precision complex scalar multiplier applied to op(A).
         double-precision complex array of dimensions (lda, n) if trans ==
Α
         'N' or 'n', of dimensions (lda.m) otherwise: lda must be at least
         \max(1, m) if trans == 'N' or 'n' and at least \max(1, n) otherwise.
lda
         leading dimension of two-dimensional array used to store matrix A.
         double-precision complex array of length at least
х
         (1+(n-1)*abs(incx)) if trans == 'N' or 'n', else at least
         (1 + (m-1) * abs(incx)).
incx
         specifies the storage spacing for elements of x; incx must not be zero.
beta
         double-precision complex scalar multiplier applied to vector y. If beta
         is zero, y is not read.
```

Input (continued)

```
double-precision complex array of length at least
         (1+(m-1)*abs(incy)) if trans == 'N' or 'n', else at least
         (1+(n-1)*abs(incy)).
        the storage spacing between elements of y; incy must not be zero.
incy
Output
```

```
У
        updated according to y = alpha * op(A) * x + beta * y.
```

Reference: http://www.netlib.org/blas/zgemv.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0, n < 0, incx == 0, or incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZgerc()

```
void
```

```
cublasZgerc (int m, int n, cuDoubleComplex alpha,
             const cuDoubleComplex *x, int incx,
             const cuDoubleComplex *y, int incy,
             cuDoubleComplex *A, int lda)
```

performs the symmetric rank 1 operation

```
A = alpha * x * y^H + A,
```

where alpha is a double-precision complex scalar, x is an m-element double-precision complex vector, y is an n-element double-precision complex vector, and A is an m×n matrix consisting of double-precision complex elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array used to store A.

Innut

mput		
m	specifies the number of rows of the matrix A ; \mathfrak{m} must be at least zero.	
n	specifies the number of column	s of matrix A; n must be at least zero.
alpha	double-precision complex scalar multiplier applied to $x * y^H$.	
x	double-precision complex array of length at least	
	(1 + (m-1) * abs(incx)).	
incx	the storage spacing between elements of x; incx must not be zero.	
У	double-precision complex array of length at least	
	(1+(n-1)*abs(incy)).	
incy	the storage spacing between elements of y ; incy must not be zero.	
A	double-precision complex array of dimensions (lda, n).	
lda	leading dimension of two-dimensional array used to store matrix A.	
Output		
A	updated according to A = alph	na * x * y ^H + A.
Referen	ce: http://www.netlib.org/bla	as/zgerc.f
Error st	atus for this function can be	retrieved via cublasGetError().
Error S	tatus	
CUBLAS	_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS	_STATUS_INVALID_VALUE	if m < 0, n < 0, incx == 0, or

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0, n < 0, incx == 0, or
	incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that
	does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZgeru()

performs the symmetric rank 1 operation

$$A = alpha * x * y^T + A,$$

where alpha is a double-precision complex scalar, x is an m-element double-precision complex vector, y is an n-element double-precision complex vector, and A is an m×n matrix consisting of double-precision complex elements. Matrix A is stored in column-major format, and lda is the leading dimension of the two-dimensional array used to store A.

Input

m	specifies the number of rows of the matrix A; m must be at least zero.
n	specifies the number of columns of matrix A ; n must be at least zero.
alpha	double-precision complex scalar multiplier applied to $\mathbf{x} \star \mathbf{y}^{\mathrm{T}}$.
x	double-precision complex array of length at least
	(1+(m-1)*abs(incx)).
incx	the storage spacing between elements of x; incx must not be zero.
У	double-precision complex array of length at least
	(1+(n-1)*abs(incy)).
incy	the storage spacing between elements of y; incy must not be zero.
A	double-precision complex array of dimensions (lda, n).
lda	leading dimension of two-dimensional array used to store matrix A.

Output

Reference: http://www.netlib.org/blas/zgeru.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0, n < 0, incx == 0, or
	incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that
	does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZhbmv()

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are double-precision complex scalars, and x and y are n-element double-precision complex vectors. A is a Hermitian $n \times n$ band matrix that consists of double-precision complex elements, with k superdiagonals and the same number of subdiagonals.

- specifies whether the upper or lower triangular part of the Hermitian band matrix A is being supplied. If uplo == 'U' or 'u', the upper triangular part is being supplied. If uplo == 'L' or 'l', the lower triangular part is being supplied.
- n specifies the number of rows and the number of columns of the symmetric matrix A; n must be at least zero.
- k specifies the number of superdiagonals of matrix A. Since the matrix is Hermitian, this is also the number of subdiagonals; k must be at least zero.
- alpha double-precision complex scalar multiplier applied to A \star x.

Input (continued)

A	double-precision complex array of dimensions (lda, n). If uplo ==
	'U' or 'u', the leading (k + 1)×n part of array A must contain the
	upper triangular band of the Hermitian matrix, supplied column by
	column, with the leading diagonal of the matrix in row $k + 1$ of the
	array, the first superdiagonal starting at position 2 in row k, and so on.
	The top left kxk triangle of array A is not referenced. When uplo ==
	'L' or 'l', the leading $(k + 1) \times n$ part of array A must contain the
	lower triangular band part of the Hermitian matrix, supplied column
	by column, with the leading diagonal of the matrix in row 1 of the
	array, the first subdiagonal starting at position 1 in row 2, and so on.
	The bottom right k×k triangle of array A is not referenced. The
	imaginary parts of the diagonal elements need not be set; they are assumed to be zero.
lda	leading dimension of A; 1da must be at least $k + 1$.
х	double-precision complex array of length at least
	(1+(n-1)*abs(incx)).
incx	storage spacing between elements of x; incx must not be zero.
beta	double-precision complex scalar multiplier applied to vector y.
У	double-precision complex array of length at least
	(1+(n-1)*abs(incy)). If beta is zero, y is not read.
incy	storage spacing between elements of y; incy must not be zero.

Output

У	updated according to $y = alpha * A * x + beta * y$.
---	---

Reference: http://www.netlib.org/blas/zhbmv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if k < 0, n < 0, incx == 0, or
	incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZhemv()

void

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are double-precision complex scalars, and x and y are n-element double-precision complex vectors. A is a Hermitian $n \times n$ matrix that consists of double-precision complex elements and is stored in either upper or lower storage mode.

- specifies whether the upper or lower triangular part of the array A is referenced. If uplo == 'u'' or 'u'', the Hermitian matrix A is stored in upper storage mode; that is, only the upper triangular part of A is referenced while the lower triangular part of A is inferred. If uplo == 'u' or 'l', the Hermitian matrix A is stored in lower storage mode; that is, only the lower triangular part of A is referenced while the upper triangular part of A is inferred.
- n specifies the number of rows and the number of columns of the symmetric matrix A; n must be at least zero.
- alpha double-precision complex scalar multiplier applied to A * x.
- double-precision complex array of dimensions (lda, n). If $uplo == \ 'U'$ or 'u', the leading $n \times n$ upper triangular part of the array A must contain the upper triangular part of the Hermitian matrix, and the strictly lower triangular part of A is not referenced. If $uplo == \ 'L'$ or 'l', the leading $n \times n$ lower triangular part of the array A must contain the lower triangular part of the Hermitian matrix, and the strictly upper triangular part of A is not referenced. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero.
- lda leading dimension of A; lda must be at least max(1, n).
- double-precision complex array of length at least (1+(n-1)*abs(incx)).
- incx storage spacing between elements of x; incx must not be zero.

Input (continued)

beta	double-precision complex scalar multiplier applied to vector y.
У	double-precision complex array of length at least
	(1+(n-1)* abs(incy)). If beta is zero, y is not read.
incy	storage spacing between elements of y ; incy must not be zero.
Otmt	

Output

y updated according to y = alpha * A * x + beta * y.

Reference: http://www.netlib.org/blas/zhemv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0, incx == 0, or incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZher()

```
void
```

performs the Hermitian rank 1 operation

$$A = alpha * x * x^{H} + A$$
,

where alpha is a double-precision scalar, x is an n-element double-precision complex vector, and A is an $n \times n$ Hermitian matrix consisting of double-precision complex elements. A is stored in column-major format, and lda is the leading dimension of the two-dimensional array containing A.

Ir	n	П	t
	יףי	ч	ι

•	
uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array A. If $uplo == 'U'$ or $'u'$, only the upper triangular part of A is referenced. If $uplo == 'L'$ or $'l'$, only the lower triangular part of A is referenced.
n	the number of rows and columns of matrix A ; n must be at least zero.
alpha	double-precision scalar multiplier applied to \times * \times^{H} .
x	double-precision complex array of length at least
	(1+(n-1)*abs(incx)).
incx	the storage spacing between elements of x; incx must not be zero.
Α	double-precision complex array of dimensions (lda, n). If uplo == 'U' or 'u', A contains the upper triangular part of the Hermitian matrix, and the strictly lower triangular part is not referenced. If uplo == 'L' or 'l', A contains the lower triangular part of the Hermitian matrix, and the strictly upper triangular part is not referenced. The imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.
lda	leading dimension of the two-dimensional array containing A; lda must be at least $max(1, n)$.

Output

A updated according to $A = alpha * x * x^{H} + A$.

Reference: http://www.netlib.org/blas/zher.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0 or incx == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZher2()

void

performs the Hermitian rank 2 operation

$$A = alpha * x * y^{H} + \overline{alpha} * y * x^{H} + A$$

where alpha is a double-precision complex scalar, x and y are nelement double-precision complex vectors, and A is an $n \times n$ Hermitian matrix consisting of double-precision complex elements.

mpat	
uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array A. If $uplo == 'U'$ or $'u'$, only the upper triangular part of A may be referenced and the lower triangular part of A is inferred. If $uplo == 'L'$ or $'l'$, only the lower triangular part of A may be referenced and the upper triangular part of A is inferred.
n	the number of rows and columns of matrix A ; n must be at least zero.
alpha	double-precision complex scalar multiplier applied to
	$x * y^H$ and whose conjugate is applied to $y * x^H$.
х	double-precision array of length at least $(1+(n-1)*abs(incx))$.
incx	the storage spacing between elements of x ; incx must not be zero.
У	double-precision array of length at least $(1 + (n-1) * abs(incy))$.
incy	the storage spacing between elements of y; incy must not be zero.
А	double-precision complex array of dimensions (lda, n). If uplo == 'U' or 'u', A contains the upper triangular part of the Hermitian matrix, and the strictly lower triangular part is not referenced. If uplo == 'L' or 'l', A contains the lower triangular part of the Hermitian matrix, and the strictly upper triangular part is not referenced. The imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.
lda	leading dimension of A; lda must be at least $max(1, n)$.

Output

updated according to $A = alpha * x * y^{H} + \overline{alpha} * y * x^{H} + A$

Reference: http://www.netlib.org/blas/zher2.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0, incx == 0, or incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZhpmv()

void

performs the matrix-vector operation

```
y = alpha * A * x + beta * y,
```

where alpha and beta are double-precision complex scalars, and x and y are n-element double-precision complex vectors. A is a Hermitian $n \times n$ matrix that consists of double-precision complex elements and is supplied in packed form.

```
specifies whether the matrix data is stored in the upper or the lower triangular part of array AP. If uplo == 'U' or 'u', the upper triangular part of A is supplied in AP. If uplo == 'L' or 'l', the lower triangular part of A is supplied in AP.

the number of rows and columns of matrix A; n must be at least zero.

alpha double-precision complex scalar multiplier applied to A * x.
```

Input (continued)

AP	double-precision complex array with at least $(n * (n + 1))/2$ elements.
	If uplo == 'U' or 'u', array AP contains the upper triangular part of
	the Hermitian matrix A, packed sequentially, column by column; that
	is, if $i \le j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo ==
	'L' or 'l', the array AP contains the lower triangular part of the
	Hermitian matrix A, packed sequentially, column by column; that is, if
	i >= j, A[i,j] is stored in AP[i+((2 * n - j + 1) * j)/2]. The
	imaginary parts of the diagonal elements need not be set; they are
	assumed to be zero.
x	double-precision complex array of length at least
	(1+(n-1)*abs(incx)).
incx	storage spacing between elements of x; incx must not be zero.
beta	double-precision scalar multiplier applied to vector y.
У	double-precision array of length at least $(1 + (n-1) * abs(incy))$.
	If beta is zero, y is not read.
incy	storage spacing between elements of y; incy must not be zero.

Output

```
y updated according to y = alpha * A * x + beta * y.
```

Reference: http://www.netlib.org/blas/zhpmv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ (). Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0, incx == 0, or incy == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZhpr()

void

performs the Hermitian rank 1 operation

$$A = alpha * x * x^{H} + A$$
,

where alpha is a double-precision scalar, x is an n-element double-precision complex vector, and A is an $n \times n$ Hermitian matrix consisting of double-precision complex elements that is supplied in packed form.

Input

uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array AP. If uplo == 'U' or 'u', the upper triangular part of A is supplied in AP. If uplo == 'L' or 'l', the lower triangular part of A is supplied in AP.
n	the number of rows and columns of matrix A ; n must be at least zero.
alpha	double-precision scalar multiplier applied to $\mathbf{x} * \mathbf{x}^{H}$.
x	double-precision complex array of length at least
	(1+(n-1)*abs(incx)).
incx	the storage spacing between elements of x ; incx must not be zero.
АР	double-precision complex array with at least $(n*(n+1))/2$ elements. If $uplo=='u'$ or $'u'$, array AP contains the upper triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if $i <= j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If $uplo=='L'$ or 'l', the array AP contains the lower triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if $i >= j$, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2]. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.

Output

A updated according to $A = alpha * x * x^{H} + A$.

Reference: http://www.netlib.org/blas/zhpr.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0 or incx == 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZhpr2()

performs the Hermitian rank 2 operation

$$A = alpha * x * y^{H} + \overline{alpha} * y * x^{H} + A,$$

where alpha is a double-precision complex scalar, x and y are nelement double-precision complex vectors, and A is an $n \times n$ Hermitian matrix consisting of double-precision complex elements that is supplied in packed form.

Input

uplo	specifies whether the matrix data is stored in the upper or the lower
	triangular part of array A. If uplo == 'U' or 'u', only the upper
	triangular part of A may be referenced and the lower triangular part of
	A is inferred. If uplo == 'L' or 'l', only the lower triangular part of
	A may be referenced and the upper triangular part of A is inferred.
n	the number of rows and columns of matrix A ; n must be at least zero.
alpha	double-precision complex scalar multiplier applied to
	$x * y^H$ and whose conjugate is applied to $y * x^H$.
x	double-precision complex array of length at least
	(1+(n-1)*abs(incx)).
incx	the storage spacing between elements of x; incx must not be zero.
У	double-precision complex array of length at least
	(1+(n-1)*abs(incy)).

Input (continued)

the storage spacing between elements of y; incy must not be zero. double-precision complex array with at least (n*(n+1))/2 elements. If uplo=='U' or 'u', array AP contains the upper triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if i <= j, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If uplo=='L' or 'l', the array AP contains the lower triangular part of the Hermitian matrix A, packed sequentially, column by column; that is, if i >= j, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2]. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.

Output

updated according to $A = alpha * x * y^H + \overline{alpha} * y * x^H + A$

Reference: http://www.netlib.org/blas/zhpr2.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized

CUBLAS_STATUS_INVALID_VALUE if n < 0, incx == 0, or incy == 0

CUBLAS_STATUS_ARCH_MISMATCH if function invoked on device that does not support double precision

CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasZtbmv()

void

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

x is an n-element double-precision complex vector, and A is an $n \times n$, unit or non-unit, upper or lower, triangular band matrix consisting of double-precision complex elements.

Input

uplo	specifies whether the matrix A is an upper or lower triangular band matrix. If uplo == 'U' or 'u', A is an upper triangular band matrix. If uplo == 'L' or 'l', A is a lower triangular band matrix.
trans	specifies op(A). If trans == 'N' Or 'n', op(A) = A.
	If trans == 'T' Or 't', $op(A) = A^{T}$.
	If trans == 'C', Or 'c', $op(A) = A^{H}$.
diag	specifies whether or not matrix A is unit triangular. If $\mathtt{diag} == '\mathtt{U}'$ or $'\mathtt{u}'$, A is assumed to be unit triangular. If $\mathtt{diag} == '\mathtt{N}'$ or $'\mathtt{n}'$, A is not assumed to be unit triangular.
n	specifies the number of rows and columns of the matrix \mathtt{A} ; \mathtt{n} must be at least zero.
k	specifies the number of superdiagonals or subdiagonals. If $uplo == 'u'$ or $'u'$, k specifies the number of superdiagonals. If $uplo == 'L'$ or $'l'$ k specifies the number of subdiagonals; k must at least be zero.
Α	double-precision complex array of dimension (lda, n). If uplo == 'U' or 'u', the leading (k+1)×n part of the array A must contain the upper triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row k+1 of the array, the first superdiagonal starting at position 2 in row k, and so on. The top left k×k triangle of the array A is not referenced. If uplo == 'L' or 'l', the leading (k+1)×n part of the array A must contain the lower triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first subdiagonal starting at position 1 in row 2, and so on. The bottom right k×k triangle of the array is not referenced.
lda	is the leading dimension of A; 1da must be at least k+1.
x	double-precision complex array of length at least
	(1+(n-1)*abs(incx)).
	On entry, \times contains the source vector. On exit, \times is overwritten with the result vector.
incx	specifies the storage spacing for elements of x; incx must not be zero.
Output	

x updated according to x = op(A) * x.

Reference: http://www.netlib.org/blas/ztbmv.f

Error status for this function can be retrieved via **cublasGetError()**.

Fror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == $0, k < 0, or n < 0$
CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZtbsv()

void

solves one of the systems of equations

```
op(A) * x = b,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

b and x are n-element vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular band matrix with k+1 diagonals.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

```
specifies whether the matrix is an upper or lower triangular band
matrix: If uplo == 'U' or 'u', A is an upper triangular band matrix. If
uplo == 'L' or 'l', A is a lower triangular band matrix.

trans
specifies op(A). If trans == 'N' or 'n', op(A) = A.

If trans == 'T' or 't', op(A) = A<sup>T</sup>.

If trans == 'C', or 'c', op(A) = A<sup>H</sup>.
```

Input (continued)

diag	specifies whether A is unit triangular. If diag == 'U' or 'u', A is
	assumed to be unit triangular; that is, diagonal elements are not read
	and are assumed to be unity. If diag == 'N' or 'n', A is not assumed
	to be unit triangular.

- n the number of rows and columns of matrix A; n must be at least zero. In the current implementation n must not exceed 1016.
- k specifies the number of superdiagonals or subdiagonals. If uplo == 'U' or 'u', k specifies the number of superdiagonals. If uplo == 'L' or 'l', k specifies the number of subdiagonals; k must be at least zero.
- double-precision complex array of dimension (lda,n). If uplo == 'U' or 'u', the leading (k+1)×n part of the array A must contain the upper triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row k+1 of the array, the first superdiagonal starting at position 2 in row k, and so on. The top left k×k triangle of the array A is not referenced. If uplo == 'L' or 'l', the leading (k+1)×n part of the array A must contain the lower triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k×k triangle of the array is not referenced.
- x double-precision complex array of length at least (1+(n-1)*abs(incx)).
- incx storage spacing between elements of x; incx must not be zero.

Output

x updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/ztbsv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0, n < 0, or n > 1016
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZtpmv()

void

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

x is an n-element double-precision complex vector, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix consisting of double-precision complex elements.

```
uplo
         specifies whether the matrix A is an upper or lower triangular matrix.
         If uplo == 'U' or 'u', A is an upper triangular matrix.
         If uplo == 'L' or 'l', A is a lower triangular matrix.
trans
         specifies op(A). If trans == 'N' or 'n', op(A) = A.
         If trans == 'T' Or 't', op(A) = A^{T}.
         If trans == 'C', Or 'c', op(A) = A^{H}.
         specifies whether or not matrix A is unit triangular.
diag
         If diag == 'U' or 'u', A is assumed to be unit triangular.
         If diag == 'N' or 'n', A is not assumed to be unit triangular.
         specifies the number of rows and columns of the matrix A; n must be
n
         at least zero.
AΡ
         double-precision complex array with at least (n * (n + 1))/2 elements.
         If uplo == 'U' or 'u', the array AP contains the upper triangular part
         of the symmetric matrix A, packed sequentially, column by column;
         that is, if i \le j, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If
         uplo == 'L' or 'l', array AP contains the lower triangular part of the
         symmetric matrix A, packed sequentially, column by column; that is, if
         i >= j, A[i, j] is stored in AP[i + ((2 * n - j + 1) * j)/2].
         double-precision complex array of length at least
х
         (1 + (n-1) * abs(incx)). On entry, x contains the source vector.
         On exit. \times is overwritten with the result vector.
incx
         specifies the storage spacing for elements of x; incx must not be zero.
```

Output

х	updated according to $x = op(A) * x$.	
---	--	--

Reference: http://www.netlib.org/blas/ztpmv.f

Error status for this function can be retrieved via cublasGetError().

Frror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0 or n < 0
CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZtpsv()

void

solves one of the systems of equations

```
op(A) * x = b,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

b and x are n-element complex vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

Input

```
specifies whether the matrix is an upper or lower triangular matrix. If
uplo == 'U' Or 'u', A is an upper triangular matrix. If uplo == 'L'
or 'l', A is a lower triangular matrix.

trans
specifies op(A). If trans == 'N' Or 'n', op(A) = A.
If trans == 'T' Or 't', op(A) = A<sup>T</sup>.
If trans == 'C', Or 'c', op(A) = A<sup>H</sup>.
```

Input (continued)

diag	specifies whether A is unit triangular. If diag == 'U' or 'u', A is assumed to be unit triangular; that is, diagonal elements are not read and are assumed to be unity. If diag == 'N' or 'n', A is not assumed to be unit triangular.
n	specifies the number of rows and columns of the matrix A; n must be at least zero. In the current implementation n must not exceed 1016.
АР	double-precision complex array with at least $(n * (n+1))/2$ elements. If $uplo == 'U'$ or 'u', array AP contains the upper triangular matrix A, packed sequentially, column by column; that is, if $i <= j$, A[i,j] is stored in AP[i+(j*(j+1)/2)]. If $uplo == 'L'$ or 'l', array AP contains the lower triangular matrix A, packed sequentially, column by column; that is, if $i >= j$, A[i,j] is stored in AP[i+((2*n-j+1)*j)/2]. When $diag == 'U'$ or 'u', the diagonal elements of A are not referenced and are assumed to be unity.
х	double-precision complex array of length at least $(1+(n-1)*abs(incx))$.
incx	storage spacing between elements of x ; incx must not be zero.

Output

x updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/ztpsv.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0, n < 0, or n > 1016
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZtrmv()

void

performs one of the matrix-vector operations

```
x = op(A) * x,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

x is an n-element double-precision complex vector; and A is an n×n, unit or non-unit, upper or lower, triangular matrix consisting of double-precision complex elements.

Input

```
specifies whether the matrix A is an upper or lower triangular matrix.
uplo
         If uplo == 'U' or 'u', A is an upper triangular matrix.
         If uplo == 'L' or 'l', A is an lower triangular matrix.
trans
         specifies op (A). If trans == 'N' or 'n', op (A) = A.
         If trans == 'T' Or 't', op(A) = A^{T}.
         If trans == 'C' Or 'c', op(A) = A^{H}.
diag
         specifies whether or not A is a unit triangular matrix. If diag == 'U'
         or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is
         not assumed to be unit triangular.
         specifies the number of rows and columns of the matrix A; n must be
n
         at least zero.
         double-precision complex array of dimensions (lda, n). If uplo ==
Α
          'U' or 'u', the leading n×n upper triangular part of the array A must
         contain the upper triangular matrix, and the strictly lower triangular
         part of A is not referenced. If uplo == 'L' or 'l', the leading n×n
         lower triangular part of the array A must contain the lower triangular
         matrix, and the strictly upper triangular part of A is not referenced.
         When diag == 'U' or 'u', the diagonal elements of A are not
         referenced either, but are assumed to be unity.
lda
         leading dimension of A; lda must be at least max(1, n).
```

Input (continued)

х	double-precision complex array of length at least
	(1+(n-1)*abs(incx)). On entry, x contains the source vector.
	On exit, x is overwritten with the result vector.
incx	the storage spacing between elements of \mathbf{x} ; inc \mathbf{x} must not be zero.

Output

```
x updated according to x = op(A) * x.
```

Reference: http://www.netlib.org/blas/ztrmv.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0 or n < 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU
CUBLAS_STATUS_ALLOC_FAILED	if function cannot allocate enough internal scratch vector memory

Function cublasZtrsv()

void

solves a system of equations

```
op(A) * x = b,
where op(A) = A, op(A) = A^T, or op(A) = A^H;
```

b and x are n-element double-precision complex vectors, and A is an $n \times n$, unit or non-unit, upper or lower, triangular matrix consisting of single-precision elements. Matrix A is stored in column-major format, and 1da is the leading dimension of the two-dimensional array containing A.

No test for singularity or near-singularity is included in this function. Such tests must be performed before calling this function.

Input

uplo	specifies whether the matrix data is stored in the upper or the lower triangular part of array A. If $uplo == 'U'$ or $'u'$, only the upper triangular part of A may be referenced. If $uplo == 'L'$ or $'l'$, only the lower triangular part of A may be referenced.
trans	specifies $op(A)$. If trans == 'N' or 'n', $op(A) = A$.
	If trans == 'T' Or 't', $op(A) = A^{T}$.
	If trans == 'C' Or 'c', op(A) = A^H .
diag	specifies whether or not A is a unit triangular matrix. If diag == 'U' or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is not assumed to be unit triangular.
n	specifies the number of rows and columns of the matrix \mathtt{A} ; \mathtt{n} must be at least zero.
A	double-precision complex array of dimensions (lda, n). If uplo == 'U' or 'u', A contains the upper triangular part of the symmetric matrix, and the strictly lower triangular part is not referenced. If uplo == 'L' or 'l', A contains the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.
lda	leading dimension of the two-dimensional array containing A; lda must be at least $max(1, n)$.
x	double-precision complex array of length at least
	(1+(n-1)*abs(incx)). On entry, x contains the n-element, right-hand-side vector b. On exit, it is overwritten with solution vector x.
incx	the storage spacing between elements of \mathtt{x} ; incx must not be zero.

Output

updated to contain the solution vector x that solves op(A) * x = b.

Reference: http://www.netlib.org/blas/ztrsv.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if incx == 0 or n < 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

CHAPTER

5

BLAS3 Functions

Level 3 Basic Linear Algebra Subprograms (BLAS3) perform matrix-matrix operations. The CUBLAS implementations are described in the following sections:

- □ "Single-Precision BLAS3 Functions" on page 174
- □ "Single-Precision Complex BLAS3 Functions" on page 187
- □ "Double-Precision BLAS3 Functions" on page 206
- □ "Double-Precision Complex BLAS3 Functions" on page 219

Single-Precision BLAS3 Functions

The single-precision BLAS3 functions are listed below:

- □ "Function cublasSgemm()" on page 175
- □ "Function cublasSsymm()" on page 176
- □ "Function cublasSsyrk()" on page 178
- □ "Function cublasSsyr2k()" on page 180
- □ "Function cublasStrmm()" on page 182
- □ "Function cublasStrsm()" on page 184

Function cublasSgemm()

void

computes the product of matrix A and matrix B, multiplies the result by scalar alpha, and adds the sum to the product of matrix C and scalar beta. It performs one of the matrix-matrix operations:

```
C = alpha * op(A) * op(B) + beta * C,
where op(X) = X or op(X) = X^{T},
```

and alpha and beta are single-precision scalars. A, B, and C are matrices consisting of single-precision elements, with op(A) an $m \times k$ matrix, op(B) a $k \times n$ matrix, and C an $m \times n$ matrix. Matrices A, B, and C are stored in column-major format, and lda, ldb, and ldc are the leading dimensions of the two-dimensional arrays containing A, B, and C.

Input

```
transa specifies op(A). If transa == 'N' Or 'n', op(A) = A.
         If transa == 'T', 't', 'C', Or 'c', Op(A) = A^{T}.
transb specifies op(B). If transb == 'N' or 'n', op(B) = B.
         If transb == 'T', 't', 'C', Or 'c', Op(B) = B^T.
         number of rows of matrix op (A) and rows of matrix C; m must be at
m
         least zero.
         number of columns of matrix op (B) and number of columns of C;
n
         n must be at least zero.
         number of columns of matrix op (A) and number of rows of op (B);
k
         k must be at least zero.
alpha
         single-precision scalar multiplier applied to op(A) * op(B).
         single-precision array of dimensions (lda, k) if transa == 'N' or
Α
         'n', and of dimensions (lda, m) otherwise. If transa == 'N' or
         'n', 1da must be at least max(1, m); otherwise, 1da must be at least
         \max(1, k).
         leading dimension of two-dimensional array used to store matrix A.
lda
```

Input (continued)

В	single-precision array of dimensions (ldb, n) if transb == 'N' or
	'n', and of dimensions (ldb, k) otherwise. If transb == 'N' or
	'n', ldb must be at least $max(1, k)$; otherwise, ldb must be at least
	$\max(1,n)$.
ldb	leading dimension of two-dimensional array used to store matrix B.
beta	single-precision scalar multiplier applied to c. If zero, c does not have to be a valid input.
С	single-precision array of dimensions (ldc, n); ldc must be at least max (1, m).
ldc	leading dimension of two-dimensional array used to store matrix c.

Output

```
C updated based on C = alpha * op(A) * op(B) + beta * C.
```

Reference: http://www.netlib.org/blas/sgemm.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

Function cublasSsymm()

void

performs one of the matrix-matrix operations

```
C = alpha * A * B + beta * C  or C = alpha * B * A + beta * C,
```

where alpha and beta are single-precision scalars, A is a symmetric matrix consisting of single-precision elements and is stored in either

lower or upper storage mode. B and C are m×n matrices consisting of single-precision elements.

Input

```
side
         specifies whether the symmetric matrix A appears on the left-hand side
         or right-hand side of matrix B.
         If side == 'L' Or 'l', C = alpha * A * B + beta * C.
         If side == 'R' Or 'r', C = alpha * B * A + beta * C.
uplo
         specifies whether the symmetric matrix A is stored in upper or lower
         storage mode. If uplo == 'U' or 'u', only the upper triangular part
         of the symmetric matrix is referenced, and the elements of the strictly
         lower triangular part are inferred from those in the upper triangular
         part. If uplo == 'L' or 'l', only the lower triangular part of the
         symmetric matrix is referenced, and the elements of the strictly upper
         triangular part are inferred from those in the lower triangular part.
         specifies the number of rows of matrix c, and the number of rows of
m
         matrix B. It also specifies the dimensions of symmetric matrix A when
         side == 'L' or 'l'; m must be at least zero.
         specifies the number of columns of matrix c, and the number of
n
         columns of matrix B. It also specifies the dimensions of symmetric
         matrix A when side == 'R' or 'r'; n must be at least zero.
alpha
         single-precision scalar multiplier applied to A * B or B * A.
Α
         single-precision array of dimensions (lda, ka), where ka is m when
         side == 'L' or 'l' and is n otherwise. If side == 'L' or 'l', the
         leading m×m part of array A must contain the symmetric matrix such
         that when uplo == 'U' or 'u', the leading mxm part stores the upper
         triangular part of the symmetric matrix, and the strictly lower
         triangular part of A is not referenced; and when uplo == 'L' or 'l',
         the leading m×m part stores the lower triangular part of the symmetric
         matrix, and the strictly upper triangular part is not referenced. If
         side == 'R' or 'r', the leading n \times n part of array A must contain the
         symmetric matrix such that when uplo == 'U' or 'u', the leading
         n×n part stores the upper triangular part of the symmetric matrix, and
         the strictly lower triangular part of A is not referenced; and when
         uplo == 'L' or 'l', the leading n \times n part stores the lower triangular
         part of the symmetric matrix, and the strictly upper triangular part is
         not referenced.
lda
         leading dimension of A. When side == 'L' or 'l', it must be at least
```

 $\max(1, m)$ and at least $\max(1, n)$ otherwise.

Input (continued)

В	single-precision array of dimensions (ldb, n). On entry, the leading $m \times n$ part of the array contains the matrix B.
ldb	leading dimension of B; 1db must be at least $max(1, m)$.
beta	single-precision scalar multiplier applied to c. If beta is zero, c does not have to be a valid input.
С	single-precision array of dimensions (ldc, n).
ldc	leading dimension of c; ldc must be at least $max(1, m)$.

Output

```
updated according to C = alpha * A * B + beta * C or
C = alpha * B * A + beta * C.
```

Reference: http://www.netlib.org/blas/ssymm.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZEDif CUBLAS library was not initializedCUBLAS_STATUS_INVALID_VALUEif m < 0 or n < 0CUBLAS_STATUS_EXECUTION_FAILEDif function failed to launch on GPU
```

Function cublasSsyrk()

void

performs one of the symmetric rank k operations

```
C = alpha * A * A^T + beta * C  or C = alpha * A^T * A + beta * C,
```

where alpha and beta are single-precision scalars. C is an $n \times n$ symmetric matrix consisting of single-precision elements and is stored in either lower or upper storage mode. A is a matrix consisting of single-precision elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.

Input uplo

storage mode. If uplo == 'U' or 'u', only the upper triangular part of the symmetric matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If uplo == 'L' or 'l', only the lower triangular part of the symmetric matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part. specifies the operation to be performed. If trans == 'N' or 'n', trans $C = alpha * A * A^{T} + beta * C.$ If trans == 'T', 't', 'C', Or 'c', $C = alpha * A^T * A + beta * C.$ specifies the number of rows and the number columns of matrix c. If n trans == 'N' or 'n', n specifies the number of rows of matrix A. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrix A: n must be at least zero. If trans == 'N' or 'n', k specifies the number of columns of k matrix A. If trans == 'T', 't', 'C', or 'c', k specifies the number

specifies whether the symmetric matrix c is stored in upper or lower

alpha single-precision scalar multiplier applied to $A * A^T$ or $A^T * A$.

of rows of matrix A: k must be at least zero.

- single-precision array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading $n \times k$ part of array A contains the matrix A; otherwise, the leading $k \times n$ part of the array contains the matrix A.
- leading dimension of A. When trans == 'N' or 'n', lda must be at least max(1, n). Otherwise lda must be at least max(1, k).
- beta single-precision scalar multiplier applied to c. If beta is zero, c is not read.

Input (continued)

single-precision array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading n×n triangular part of the array c must contain the upper triangular part of the symmetric matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading n×n triangular part of the array c must contain the lower triangular part of the symmetric matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix.

ldc leading dimension of C; ldc must be at least max(1, n).

Output

```
Updated according to C = alpha * A * A^T + beta * C or C = alpha * A^T * A + beta * C.
```

Reference: http://www.netlib.org/blas/ssyrk.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasSsyr2k()

void

performs one of the symmetric rank 2k operations

```
C = alpha * A * B^T + alpha * B * A^T + beta * C  or C = alpha * A^T * B + alpha * B^T * A + beta * C,
```

where alpha and beta are single-precision scalars. C is an $n \times n$ symmetric matrix consisting of single-precision elements and is stored in either lower or upper storage mode. A and B are matrices consisting

of single-precision elements with dimension of n×k in the first case and kxn in the second case.

Input

```
uplo
         specifies whether the symmetric matrix c is stored in upper or lower
         storage mode. If uplo == 'U' or 'u', only the upper triangular part
         of the symmetric matrix is referenced, and the elements of the strictly
         lower triangular part are inferred from those in the upper triangular
         part. If uplo == 'L' or 'l', only the lower triangular part of the
         symmetric matrix is referenced, and the elements of the strictly upper
         triangular part are inferred from those in the lower triangular part.
trans
         specifies the operation to be performed. If trans == 'N' or 'n',
         C = alpha * A * B^{T} + alpha * B * A^{T} + beta * C. If trans == 'T',
         't', 'C', OI' 'c', C = alpha * A^T * B + alpha * B^T * A + beta * C.
         specifies the number of rows and the number columns of matrix c. If
n
         trans == 'N' or 'n', n specifies the number of rows of matrix A. If
         trans == 'T', 't', 'C', or 'c', n specifies the number of columns
         of matrix A: n must be at least zero.
         If trans == 'N' or 'n', k specifies the number of columns of matrix
k
         A. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows
         of matrix A; k must be at least zero.
alpha
         single-precision scalar multiplier.
         single-precision array of dimensions (lda, ka), where ka is k when
Α
         trans == 'N' or 'n' and is n otherwise. When trans == 'N' or
         'n', the leading n×k part of array A must contain the matrix A,
         otherwise the leading k \times n part of the array must contain the matrix A.
lda
         leading dimension of A. When trans == 'N' or 'n', lda must be at
         least max(1, n). Otherwise 1da must be at least max(1, k).
В
         single-precision array of dimensions (ldb, kb), where kb = k when
         trans == 'N' or 'n', and k = n otherwise. When trans == 'N' or
         'n', the leading n×k part of array B must contain the matrix B,
         otherwise the leading k×n part of the array must contain the matrix B.
ldb
         leading dimension of B. When trans == 'N' or 'n', ldb must be at
         least max(1, n). Otherwise 1db must be at least max(1, k).
beta
         single-precision scalar multiplier applied to c. If beta is zero, c does
         not have to be a valid input.
```

Input (continued)

single-precision array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading n×n triangular part of the array c must contain the upper triangular part of the symmetric matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading n×n triangular part of the array c must contain the lower triangular part of the symmetric matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix.

ldc leading dimension of C; idc must be at least max(1, n).

Output

```
Updated according to
C = \text{alpha} * A * B^{T} + \text{alpha} * B * A^{T} + \text{beta} * C \text{ or}
C = \text{alpha} * A^{T} * B + \text{alpha} * B^{T} * A + \text{beta} * C.
```

Reference: http://www.netlib.org/blas/ssyr2k.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZEDif CUBLAS library was not initializedCUBLAS_STATUS_INVALID_VALUEif n < 0 or k < 0CUBLAS_STATUS_EXECUTION_FAILEDif function failed to launch on GPU
```

Function cublasStrmm()

performs one of the matrix-matrix operations

```
B = alpha * op(A) * B or B = alpha * B * op(A),
where op(A) = A or op(A) = A^T,
```

alpha is a single-precision scalar, B is an $m \times n$ matrix consisting of single-precision elements, and A is a unit or non-unit, upper or lower triangular matrix consisting of single-precision elements.

Matrices A and B are stored in column-major format, and 1da and 1db are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

```
specifies whether op(A) multiplies B from the left or right.
side
         If side == 'L' Or 'l', B = alpha * op(A) * B.
         If side == 'R' O r' r', B = alpha * B * op(A).
uplo
         specifies whether the matrix A is an upper or lower triangular matrix.
         If uplo == 'U' or 'u', A is an upper triangular matrix.
         If uplo == 'L' or 'l', A is a lower triangular matrix.
transa specifies the form of op(A) to be used in the matrix multiplication.
         If transa == 'N' Or 'n', op(A) = A.
         If transa == 'T', 't', 'C', Or 'c', Op(A) = A^{T}.
         specifies whether or not A is a unit triangular matrix. If diag == 'U'
diag
         or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is
         not assumed to be unit triangular.
         the number of rows of matrix B: m must be at least zero.
m
         the number of columns of matrix B: n must be at least zero.
n
         single-precision scalar multiplier applied to op(A)*B or B*op(A),
         respectively. If alpha is zero, no accesses are made to matrix A, and
         no read accesses are made to matrix B.
Α
         single-precision array of dimensions (lda, k). If side == 'L' or 'l',
         k = m. If side == 'R' or 'r', k = n. If uplo == 'U' or 'u', the
         leading kxk upper triangular part of the array A must contain the
         upper triangular matrix, and the strictly lower triangular part of A is
         not referenced. If uplo == 'L' or 'l', the leading k×k lower
         triangular part of the array A must contain the lower triangular matrix,
         and the strictly upper triangular part of A is not referenced. When
         diag == 'U' or 'u', the diagonal elements of A are not referenced
         and are assumed to be unity.
lda
         leading dimension of A. When side == 'L' or 'l', it must be at least
         \max(1, m) and at least \max(1, n) otherwise.
```

Input (continued)

В	single-precision array of dimensions (ldb, n). On entry, the leading
	m×n part of the array contains the matrix B. It is overwritten with the
	transformed matrix on exit.

ldb leading dimension of B; ldb must be at least max(1, m).

Output

```
B updated according to B = alpha * op(A) * B or
B = alpha * B * op(A).
```

Reference: http://www.netlib.org/blas/strmm.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasStrsm()

```
void
```

solves one of the matrix equations

```
op(A) * X = alpha * B or X * op(A) = alpha * B,
where op(A) = A or op(A) = A^{T},
```

alpha is a single-precision scalar, and X and B are $m \times n$ matrices that consist of single-precision elements. A is a unit or non-unit, upper or lower, triangular matrix.

The result matrix X overwrites input matrix B; that is, on exit the result is stored in B. Matrices A and B are stored in column-major format, and lda and ldb are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

```
specifies whether op(A) appears on the left or right of x:
side
         side == 'L' or 'l' indicates solve op(A) * X = alpha * B;
         side == 'R' Or 'r' indicates solve X * op(A) = alpha * B.
uplo
         specifies whether the matrix A is an upper or lower triangular matrix:
         uplo == 'U' or 'u' indicates A is an upper triangular matrix;
         uplo == 'L' or 'l' indicates A is a lower triangular matrix.
transa specifies the form of op(A) to be used in matrix multiplication.
         If transa == 'N' Or 'n', op(A) = A.
         If transa == 'T', 't', 'C', Or 'c', op(A) = A^{T}.
diag
         specifies whether or not A is a unit triangular matrix.
         If diag == 'U' or 'u', A is assumed to be unit triangular.
         If diag == 'N' or 'n', A is not assumed to be unit triangular.
         specifies the number of rows of B; m must be at least zero.
m
         specifies the number of columns of B; n must be at least zero.
n
         single-precision scalar multiplier applied to B. When alpha is zero, A is
alpha
         not referenced and B does not have to be a valid input.
         single-precision array of dimensions (lda, k), where k is m when
Α
         side == 'L' or 'l' and is n when side == 'R' or 'r'. If uplo ==
         'U' or 'u', the leading kxk upper triangular part of the array A must
         contain the upper triangular matrix, and the strictly lower triangular
         matrix of A is not referenced. When uplo == 'L' or 'l', the leading
         k×k lower triangular part of the array A must contain the lower
         triangular matrix, and the strictly upper triangular part of A is not
         referenced. Note that when diag == 'U' or 'u', the diagonal
         elements of A are not referenced and are assumed to be unity.
         leading dimension of the two-dimensional array containing A.
lda
         When side == 'L' or 'l', lda must be at least max(1, m).
         When side == 'R' or 'r', 1da must be at least max(1, n).
         single-precision array of dimensions (ldb, n); ldb must be at least
В
         max(1, m). The leading m×n part of the array B must contain the right-
         hand side matrix B. On exit B is overwritten by the solution matrix x.
ldb
         leading dimension of the two-dimensional array containing B; 1db
         must be at least max(1, m).
```

Output

B contains the solution matrix x satisfying op(A) * x = alpha * B or x * op(A) = alpha * B.

Reference: http://www.netlib.org/blas/strsm.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0 or n < 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Single-Precision Complex BLAS3 Functions

These are the single-precision complex BLAS3 functions:

- □ "Function cublasCgemm()" on page 188
- □ "Function cublasChemm()" on page 189
- □ "Function cublasCherk()" on page 191
- □ "Function cublasCher2k()" on page 193
- □ "Function cublasCsymm()" on page 195
- □ "Function cublasCsyrk()" on page 197
- □ "Function cublasCsyr2k()" on page 199
- □ "Function cublasCtrmm()" on page 201
- □ "Function cublasCtrsm()" on page 203

Function cublasCgemm()

void

performs one of the matrix-matrix operations

```
C = alpha * op(A) * op(B) + beta * C,
where op(X) = X, op(X) = X^{T}, or op(X) = X^{H};
```

and alpha and beta are single-precision complex scalars. A, B, and C are matrices consisting of single-precision complex elements, with op(A) an $m \times k$ matrix, op(B) a $k \times n$ matrix, and C an $m \times n$ matrix.

Input

```
transa specifies op(A). If transa == 'N' or 'n', op(A) = A.
        If transa == 'T' Or 't', op(A) = A^{T}.
        If transa == 'C' Or 'c', op(A) = A^{H}.
transb specifies op(B). If transb == 'N' Or 'n', op(B) = B.
        If transb == 'T' Or 't', op(B) = B^{T}.
        If transb == 'C' Or 'c', op(B) = B^{H}.
        number of rows of matrix op (A) and rows of matrix C;
m
        m must be at least zero.
        number of columns of matrix op (B) and number of columns of C:
n
        n must be at least zero.
        number of columns of matrix op(A) and number of rows of op(B);
k
        k must be at least zero.
alpha
        single-precision complex scalar multiplier applied to op(A)*op(B).
        single-precision complex array of dimension (lda, k) if transa ==
Α
         'N' or 'n', and of dimension (lda, m) otherwise.
lda
        leading dimension of A. When transa == 'N' or 'n', it must be at
        least max(1, m) and at least max(1, k) otherwise.
        single-precision complex array of dimension (ldb, n) if transb ==
В
         'N' or 'n', and of dimension (ldb, k) otherwise.
ldb
        leading dimension of B. When transb == 'N' or 'n', it must be at
        least max(1, k) and at least max(1, n) otherwise.
```

Input (continued)

beta	single-precision complex scalar multiplier applied to C. If beta is zero,
	c does not have to be a valid input.
C	single-precision array of dimensions (ldc, n).
ldc	leading dimension of C; idc must be at least $max(1, m)$.

Output

```
Updated according to C = alpha * op(A) * op(B) + beta * C.
```

Reference: http://www.netlib.org/blas/cgemm.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

Function cublasChemm()

```
void
```

performs one of the matrix-matrix operations

```
C = alpha * A * B + beta * C Or C = alpha * B * A + beta * C,
```

where alpha and beta are single-precision complex scalars, A is a Hermitian matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. B and C are $m \times n$ matrices consisting of single-precision complex elements.

Input

```
specifies whether the Hermitian matrix A appears on the left-hand side
side
         or right-hand side of matrix B.
         If side == 'L' Or 'l', C = alpha * A * B + beta * C.
         If side == 'R' Or 'r', C = alpha * B * A + beta * C.
uplo
         specifies whether the Hermitian matrix A is stored in upper or lower
         storage mode. If uplo == 'U' or 'u', only the upper triangular part
         of the Hermitian matrix is referenced, and the elements of the strictly
         lower triangular part are inferred from those in the upper triangular
         part. If uplo == 'L' or 'l', only the lower triangular part of the
         Hermitian matrix is referenced, and the elements of the strictly upper
         triangular part are inferred from those in the lower triangular part.
         specifies the number of rows of matrix c, and the number of rows of
m
         matrix B. It also specifies the dimensions of Hermitian matrix A when
         side == 'L' or 'l'; m must be at least zero.
         specifies the number of columns of matrix c, and the number of
n
         columns of matrix B. It also specifies the dimensions of Hermitian
         matrix A when side == 'R' or 'r'; n must be at least zero.
alpha
         single-precision complex scalar multiplier applied to A * B or B * A.
         single-precision complex array of dimensions (lda, ka), where ka is
Α
         m when side == 'L' or 'l' and is n otherwise. If side == 'L' or
         '1', the leading m×m part of array A must contain the Hermitian
         matrix such that when uplo == 'U' or 'u', the leading mxm part
         stores the upper triangular part of the Hermitian matrix, and the
         strictly lower triangular part of A is not referenced; and when uplo ==
         'L' or 'l', the leading m×m part stores the lower triangular part of the
         Hermitian matrix, and the strictly upper triangular part is not
         referenced. If side == 'R' or 'r', the leading nxn part of array A
         must contain the Hermitian matrix such that when uplo == 'U' or
         'u', the leading n×n part stores the upper triangular part of the
         Hermitian matrix, and the strictly lower triangular part of A is not
         referenced; and when uplo == 'L' or 'l', the leading n \times n part stores
         the lower triangular part of the Hermitian matrix, and the strictly
         upper triangular part is not referenced. The imaginary parts of the
         diagonal elements need not be set; they are assumed to be zero.
lda
         leading dimension of A. When side == 'L' or 'l', it must be at least
         \max(1, m) and at least \max(1, n) otherwise.
```

Input (continued)

В	single-precision complex array of dimensions (ldb, n). On entry, the
	leading m×n part of the array contains the matrix в.
ldb	leading dimension of B; 1db must be at least $max(1, m)$.
beta	single-precision complex scalar multiplier applied to c. If beta is zero, c does not have to be a valid input.
C	single-precision complex array of dimensions (ldc, n).
ldc	leading dimension of c; ldc must be at least $max(1, m)$.

Output

```
C updated according to C = alpha * A * B + beta * C or
C = alpha * B * A + beta * C.
```

Reference: http://www.netlib.org/blas/chemm.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasCherk()

```
void
```

performs one of the Hermitian rank k operations

```
C = alpha * A * A^{H} + beta * C  or C = alpha * A^{H} * A + beta * C,
```

where alpha and beta are single-precision real scalars. C is an $n \times n$ Hermitian matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. A is a matrix consisting of single-precision complex elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.

Input

uplo	specifies whether the Hermitian matrix c is stored in upper or lower storage mode. If $uplo == 'u' or 'u'$, only the upper triangular part of the Hermitian matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular
	part. If uplo == 'L' or 'l', only the lower triangular part of the Hermitian matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part.
trans	specifies the operation to be performed. If trans == 'N' or 'n', $C = alpha * A * A^H + beta * C$. If trans == 'T', 't', 'C', or 'c',
	$C = alpha * A^{H} * A + beta * C.$
n	specifies the number of rows and the number columns of matrix c. If trans == 'N' or 'n', n specifies the number of rows of matrix A. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrix A; n must be at least zero.
k	If trans == 'N' or 'n', k specifies the number of columns of matrix A. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrix A; k must be at least zero.
alpha	single-precision scalar multiplier applied to $A * A^H$ or $A^H * A$.

- A single-precision complex array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading n×k part of array A contains the matrix A; otherwise, the leading k×n part of the array contains the matrix A.
- leading dimension of A. When trans == 'N' or 'n', lda must be at least max(1, n). Otherwise lda must be at least max(1, k).
- single-precision real scalar multiplier applied to c. If beta is zero, C does not have to be a valid input.

Input (continued)

single-precision complex array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading n×n triangular part of the array c must contain the upper triangular part of the Hermitian matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading n×n triangular part of the array c must contain the lower triangular part of the Hermitian matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.

ldc leading dimension of c; ldc must be at least max(1, n).

Output

```
Updated according to C = alpha * A * A<sup>H</sup> + beta * C Or
C = alpha * A<sup>H</sup> * A + beta * C.
```

Reference: http://www.netlib.org/blas/cherk.f

Error status for this function can be retrieved via **cublasGetError()**.

Frror Status

Function cublasCher2k()

```
77014
```

performs one of the Hermitian rank 2k operations

$$C = alpha * A * B^{H} + \overline{alpha} * B * A^{H} + beta + C \text{ or}$$
 $C = alpha * A^{H} * B + \overline{alpha} * B^{H} * A + beta + C,$

where alpha is a single-precision complex scalar and beta is a single-precision real scalar. C is an $n \times n$ Hermitian matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. A and B are matrices consisting of single-precision complex elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.

Input

specifies whether the Hermitian matrix c is stored in upper or lower uplo storage mode. If uplo == 'U' or 'u', only the upper triangular part of the Hermitian matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If uplo == 'L' or 'l', only the lower triangular part of the Hermitian matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part. trans specifies the operation to be performed. If trans == 'N' or 'n', $C = alpha * A * B^{H} + \overline{alpha} * B * A^{H} + beta + C.$ If trans == 'T', 't', 'C', Or 'c', $C = alpha * A^H * B + \overline{alpha} * B^H * A + beta + C$. specifies the number of rows and the number columns of matrix c. If n trans == 'N' or 'n', n specifies the number of rows of matrices A and B. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrices A and B; n must be at least zero. k If trans == 'N' or 'n', k specifies the number of columns of matrices A and B. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrices A and B: k must be at least zero. alpha single-precision complex scalar multiplier. Α single-precision complex array of dimensions (1da, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading $n \times k$ part of array A contains the matrix A; otherwise, the leading $k \times n$ part of the array contains the matrix A. lda leading dimension of A. When trans == 'N' or 'n', lda must be at least max(1, n). Otherwise 1da must be at least max(1, k). В single-precision complex array of dimensions (1db, kb), where kb is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading n×k part of array B contains the matrix B; otherwise, the leading k×n part of the array contains the matrix B. leading dimension of B. When trans == 'N' or 'n', ldb must be at ldb least max(1, n). Otherwise 1db must be at least max(1, k).

Input ((continued)	١
IIIPGL (COLLINIACA	,

beta	single-precision real scalar multiplier applied to c.
	If beta is zero, C does not have to be a valid input.
С	single-precision complex array of dimensions (ldc,

single-precision complex array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading n×n triangular part of the array c must contain the upper triangular part of the Hermitian matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading n×n triangular part of the array c must contain the lower triangular part of the Hermitian matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.

ldc leading dimension of C; ldc must be at least max(1, n).

Output

```
updated according to C = alpha * A * B^H + \overline{alpha} * B * A^H + beta * C

or C = alpha * A^H * B + \overline{alpha} * B^H * A + beta * C.
```

Reference: http://www.netlib.org/blas/cher2k.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasCsymm()

```
void
```

performs one of the matrix-matrix operations

```
C = alpha * A * B + beta * C  or C = alpha * B * A + beta * C,
```

where alpha and beta are single-precision complex scalars, A is a symmetric matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. B and C are $m \times n$ matrices consisting of single-precision complex elements.

Input side

```
specifies whether the symmetric matrix A appears on the left-hand side
         or right-hand side of matrix B.
         If side == 'L' Or 'l', C = alpha * A * B + beta * C.
         If side == 'R' Or 'r', C = alpha * B * A + beta * C.
uplo
         specifies whether the symmetric matrix A is stored in upper or lower
         storage mode. If uplo == 'U' or 'u', only the upper triangular part
         of the symmetric matrix is referenced, and the elements of the strictly
         lower triangular part are inferred from those in the upper triangular
         part. If uplo == 'L' or 'l', only the lower triangular part of the
         symmetric matrix is referenced, and the elements of the strictly upper
         triangular part are inferred from those in the lower triangular part.
         specifies the number of rows of matrix c, and the number of rows of
m
         matrix B. It also specifies the dimensions of symmetric matrix A when
         side == 'L' or 'l'; m must be at least zero.
n
         specifies the number of columns of matrix c, and the number of
         columns of matrix B. It also specifies the dimensions of symmetric
         matrix A when side == 'R' or 'r'; n must be at least zero.
         single-precision complex scalar multiplier applied to A * B or B * A.
alpha
         single-precision complex array of dimensions (lda, ka), where ka is
Α
         m when side == 'L' or 'l' and is n otherwise. If side == 'L' or
         'l', the leading m×m part of array A must contain the symmetric
         matrix such that when uplo == 'U' or 'u', the leading mxm part
         stores the upper triangular part of the symmetric matrix, and the
         strictly lower triangular part of A is not referenced; and when uplo ==
         'L' or 'l', the leading m×m part stores the lower triangular part of the
         symmetric matrix, and the strictly upper triangular part is not
         referenced. If side == 'R' or 'r', the leading n×n part of array A
         must contain the symmetric matrix such that when uplo == 'U' or
```

'u', the leading $n \times n$ part stores the upper triangular part of the symmetric matrix, and the strictly lower triangular part of A is not referenced; and when uplo == 'L' or 'l', the leading $n \times n$ part stores the lower triangular part of the symmetric matrix, and the strictly

upper triangular part is not referenced.

Input (continued)

lda	leading dimension of A. When $side == 'L'$ or 'l', it must be at least $max(1, m)$ and at least $max(1, n)$ otherwise.
В	single-precision complex array of dimensions (ldb, n). On entry, the leading $m \times n$ part of the array contains the matrix B.
ldb	leading dimension of B; ldb must be at least $max(1, m)$.
beta	single-precision complex scalar multiplier applied to $\tt C.$ If $\tt beta$ is zero, $\tt C$ does not have to be a valid input.
C	single-precision complex array of dimensions (ldc, n).
ldc	leading dimension of c; ldc must be at least $max(1, m)$.

Output

```
c updated according to C = alpha * A * B + beta * C or
C = alpha * B * A + beta * C.
```

Reference: http://www.netlib.org/blas/csymm.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0 or n < 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasCsyrk()

void

performs one of the symmetric rank k operations

```
C = alpha * A * A^{T} + beta * C  or C = alpha * A^{T} * A + beta * C,
```

where alpha and beta are single-precision complex scalars. C is an $n \times n$ symmetric matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. A is a matrix consisting of single-precision complex elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.

Input

specifies whether the symmetric matrix c is stored in upper or lower uplo storage mode. If uplo == 'U' or 'u', only the upper triangular part of the symmetric matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If uplo == 'L' or 'l', only the lower triangular part of the symmetric matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part. specifies the operation to be performed. If trans == 'N' or 'n', trans $C = alpha * A * A^{T} + beta * C.$ If trans == 'T', 't', 'C', Or 'c', $C = alpha * A^T * A + beta * C.$ specifies the number of rows and the number columns of matrix c. If n trans == 'N' or 'n', n specifies the number of rows of matrix A. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrix A: n must be at least zero. If trans == 'N' or 'n', k specifies the number of columns of k matrix A. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrix A: k must be at least zero.

- alpha single-precision complex scalar multiplier applied to $A * A^T$ or $A^T * A$.
- A single-precision complex array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading n×k part of array A contains the matrix A; otherwise, the leading k×n part of the array contains the matrix A.
- leading dimension of A. When trans == 'N' or 'n', lda must be at least max(1, n). Otherwise lda must be at least max(1, k).
- beta single-precision complex scalar multiplier applied to c.

 If beta is zero. c is not read.

Input (continued)

single-precision complex array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading n×n triangular part of the array c must contain the upper triangular part of the symmetric matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading n×n triangular part of the array c must contain the lower triangular part of the symmetric matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix.

ldc leading dimension of c; ldc must be at least max(1, n).

Output

```
updated according to C = alpha * A * A^{T} + beta * C or C = alpha * A^{T} * A + beta * C.
```

Reference: http://www.netlib.org/blas/csyrk.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

Function cublasCsyr2k()

void

performs one of the symmetric rank 2k operations

```
C = alpha * A * B^{T} + alpha * B * A^{T} + beta * C  or C = alpha * A^{T} * B + alpha * B^{T} * A + beta * C,
```

where alpha and beta are single-precision complex scalars. C is an $n \times n$ symmetric matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. A and B are

matrices consisting of single-precision complex elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.

Input

uplo	specifies whether the symmetric matrix c is stored in upper or lower storage mode. If $uplo == 'u'$ or $'u'$, only the upper triangular part of the symmetric matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If $uplo == 'L'$ or $'l'$, only the lower triangular part of the symmetric matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part.
trans	specifies the operation to be performed. If $trans == 'N' or 'n'$,
	$C = alpha * A * B^{T} + alpha * B * A^{T} + beta * C. If trans == 'T',$
	't', 'C', Or 'c', $C = alpha * A^T * B + alpha * B^T * A + beta * C$.
n	specifies the number of rows and the number columns of matrix c. If $trans == 'N' or 'n'$, n specifies the number of rows of matrix A. If $trans == 'T'$, 't', 'C', or 'c', n specifies the number of columns of matrix A; n must be at least zero.
k	If trans == 'N' or 'n', k specifies the number of columns of matrix A. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrix A; k must be at least zero.
alpha	single-precision complex scalar multiplier.
A	single-precision complex array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading $n \times k$ part of array A contains the matrix A; otherwise, the leading $k \times n$ part of the array contains the matrix A.
lda	leading dimension of A. When trans == 'N' or 'n', lda must be at least $max(1, n)$. Otherwise lda must be at least $max(1, k)$.
В	single-precision complex array of dimensions (1db, kb), where kb is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading $n \times k$ part of array B contains the matrix B; otherwise, the leading $k \times n$ part of the array contains the matrix B.
ldb	leading dimension of B. When trans == 'N' or 'n', ldb must be at least $max(1, n)$. Otherwise ldb must be at least $max(1, k)$.
beta	single-precision complex scalar multiplier applied to c. If beta is zero, C does not have to be a valid input.

Input (continued)

single-precision complex array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading n×n triangular part of the array c must contain the upper triangular part of the symmetric matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading n×n triangular part of the array c must contain the lower triangular part of the symmetric matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix.

ldc leading dimension of C; ldc must be at least max(1, n).

Output

```
updated according to C = alpha * A * B^{T} + alpha * B * A^{T} + beta * C

or C = alpha * A^{T} * B + alpha * B^{T} * A + beta * C.
```

Reference: http://www.netlib.org/blas/csyr2k.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZEDif CUBLAS library was not initializedCUBLAS_STATUS_INVALID_VALUEif n < 0 or k < 0CUBLAS_STATUS_EXECUTION_FAILEDif function failed to launch on GPU
```

Function cublasCtrmm()

```
void
```

performs one of the matrix-matrix operations

```
B = alpha * op(A) * B or B = alpha * B * op(A),
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

alpha is a single-precision complex scalar; B is an m×n matrix consisting of single-precision complex elements; and A is a unit or non-

unit, upper or lower triangular matrix consisting of single-precision complex elements.

Matrices A and B are stored in column-major format, and 1da and 1db are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

```
specifies whether op (A) multiplies B from the left or right.
side
         If side == 'L' Or 'l', B = alpha * op(A) * B.
         If side == 'R' Or'r', B = alpha * B * op(A).
uplo
         specifies whether the matrix A is an upper or lower triangular matrix.
         If uplo == 'U' or 'u', A is an upper triangular matrix.
         If uplo == 'L' or 'l', A is a lower triangular matrix.
transa specifies op(A). If transa == 'N' or 'n', op(A) = A.
         If transa == 'T' Or 't', op(A) = A^{T}.
         If transa == 'C' Or 'c', op(A) = A^{H}.
diag
         specifies whether or not A is a unit triangular matrix. If diag == 'U'
         or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is
         not assumed to be unit triangular.
         the number of rows of matrix B; m must be at least zero.
m
         the number of columns of matrix B; n must be at least zero.
n
         single-precision complex scalar multiplier applied to op(A)*B or
alpha
         B*op(A), respectively. If alpha is zero, no accesses are made to
         matrix A, and no read accesses are made to matrix B.
Α
         single-precision complex array of dimensions (lda, k). If side ==
         'L' Or 'l', k = m. If side == 'R' Or 'r', k = n. If uplo == 'U' Or
         'u', the leading k×k upper triangular part of the array A must contain
         the upper triangular matrix, and the strictly lower triangular part of A is
         not referenced. If uplo == 'L' or 'l', the leading k×k lower
         triangular part of the array A must contain the lower triangular matrix,
         and the strictly upper triangular part of A is not referenced. When
         diag == 'U' or 'u', the diagonal elements of A are not referenced
         and are assumed to be unity.
         leading dimension of A. When side == 'L' or 'l', it must be at least
lda
         \max(1, m) and at least \max(1, n) otherwise.
```

Input (continued)

В	single-precision complex array of dimensions (1db, n). On entry, the
leading m×n part of the array contains the matrix в. It is overwri	
	with the transformed matrix on exit.

ldb leading dimension of B; ldb must be at least max(1, m).

Output

```
B updated according to B = alpha * op(A) * B or
B = alpha * B * op(A).
```

Reference: http://www.netlib.org/blas/ctrmm.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZEDif CUBLAS library was not initializedCUBLAS_STATUS_INVALID_VALUEif m < 0 or n < 0CUBLAS_STATUS_EXECUTION_FAILEDif function failed to launch on GPU
```

Function cublasCtrsm()

```
void
```

solves one of the matrix equations

```
op(A) * X = alpha * B or X * op(A) = alpha * B,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

alpha is a single-precision complex scalar, and X and B are $m \times n$ matrices that consist of single-precision complex elements. A is a unit or non-unit, upper or lower, triangular matrix.

The result matrix X overwrites input matrix B; that is, on exit the result is stored in B. Matrices A and B are stored in column-major format, and

lda and ldb are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

```
specifies whether op(A) appears on the left or right of x:
side
         side == 'L' or 'l' indicates solve op(A) * X = alpha * B;
         side == 'R' Or 'r' indicates solve X * op(A) = alpha * B.
uplo
         specifies whether the matrix A is an upper or lower triangular matrix:
         uplo == 'U' or 'u' indicates A is an upper triangular matrix;
         uplo == 'L' or 'l' indicates A is a lower triangular matrix.
transa specifies op(A). If transa == 'N' or 'n', op(A) = A.
         If transa == 'T' Or 't', op(A) = A^{T}.
         If transa == 'C' Or 'c', op(A) = A^{H}.
diag
         specifies whether or not A is a unit triangular matrix.
         If diag == 'U' or 'u', A is assumed to be unit triangular.
         If diag == 'N' or 'n', A is not assumed to be unit triangular.
         specifies the number of rows of B; m must be at least zero.
m
         specifies the number of columns of B; n must be at least zero.
n
alpha
         single-precision complex scalar multiplier applied to B. When alpha is
         zero, A is not referenced and B does not have to be a valid input.
Α
         single-precision complex array of dimensions (lda, k), where k is m
         when side == 'L' \text{ or 'l'} and is n when side == 'R' \text{ or 'r'}. If
         uplo == 'U' or 'u', the leading kxk upper triangular part of the array
         A must contain the upper triangular matrix, and the strictly lower
         triangular matrix of A is not referenced. When uplo == 'L' or 'l',
         the leading k×k lower triangular part of the array A must contain the
         lower triangular matrix, and the strictly upper triangular part of A is
         not referenced. Note that when diag == 'U' or 'u', the diagonal
         elements of A are not referenced and are assumed to be unity.
lda
         leading dimension of the two-dimensional array containing A.
         When side == 'L' or '1', 1da must be at least max(1, m).
         When side == R' O' R'. Ida must be at least max(1, n).
В
         single-precision complex array of dimensions (ldb, n); ldb must be
         at least max(1, m). The leading m×n part of the array B must contain
         the right-hand side matrix B. On exit, B is overwritten by the solution
         matrix x.
ldb
         leading dimension of the two-dimensional array containing B; 1db
         must be at least max(1, m).
```

Output

B contains the solution matrix x satisfying op(A) * x = alpha * B or x * op(A) = alpha * B.

Reference: http://www.netlib.org/blas/ctrsm.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0 or n < 0
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Double-Precision BLAS3 Functions

Note: Double-precision functions are only supported on GPUs with double-precision hardware.

The double-precision BLAS3 functions are listed below:

- □ "Function cublasDgemm()" on page 207
- □ "Function cublasDsymm()" on page 208
- □ "Function cublasDsyrk()" on page 210
- □ "Function cublasDsyr2k()" on page 212
- □ "Function cublasDtrmm()" on page 214
- □ "Function cublasDtrsm()" on page 216

Function cublasDgemm()

void

computes the product of matrix A and matrix B, multiplies the result by scalar alpha, and adds the sum to the product of matrix C and scalar beta. It performs one of the matrix-matrix operations:

```
C = alpha * op(A) * op(B) + beta * C,
where op(X) = X or op(X) = X^{T},
```

and alpha and beta are double-precision scalars. A, B, and C are matrices consisting of double-precision elements, with op(A) an $m \times k$ matrix, op(B) a $k \times n$ matrix, and C an $m \times n$ matrix. Matrices A, B, and C are stored in column-major format, and lda, ldb, and ldc are the leading dimensions of the two-dimensional arrays containing A, B, and C.

Input

```
transa specifies op (A). If transa == 'N' or 'n', op (A) = A.
         If transa == 'T', 't', 'C', Or 'c', op(A) = A^{T}.
transb specifies op(B). If transb == 'N' or 'n', op(B) = B.
         If transb == 'T', 't', 'C', Or 'c', op(B) = B^{T}.
         number of rows of matrix op (A) and rows of matrix C; m must be at
m
         least zero.
         number of columns of matrix op (B) and number of columns of C;
n
         n must be at least zero.
         number of columns of matrix op(A) and number of rows of op(B);
k
         k must be at least zero.
alpha
         double-precision scalar multiplier applied to op(A) * op(B).
         double-precision array of dimensions (lda, k) if transa == 'N' or
Α
         'n', and of dimensions (lda, m) otherwise. If transa == 'N' or
         'n', 1da must be at least max(1, m); otherwise, 1da must be at least
lda
         leading dimension of two-dimensional array used to store matrix A.
```

Input (continued)

В	double-precision array of dimensions (ldb, n) if transb == 'N' or
	'n', and of dimensions (ldb, k) otherwise. If transb == 'N' or
	'n', ldb must be at least $max(1, k)$; otherwise, ldb must be at least
	$\max(1, n)$.
ldb	leading dimension of two-dimensional array used to store matrix B.
beta	double-precision scalar multiplier applied to c. If zero, c does not have to be a valid input.
C	double-precision array of dimensions (ldc, n); ldc must be at least $max (1, m)$.
ldc	leading dimension of two-dimensional array used to store matrix ${\tt c}.$

Output

<pre>C updated based on C = alpha *</pre>	op(A) *	op(B) +	beta * C.
---	---------	---------	-----------

Reference: http://www.netlib.org/blas/dgemm.f

Error status for this function can be retrieved via ${\tt cublasGetError}$ ().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0, n < 0, or k < 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDsymm()

void

performs one of the matrix-matrix operations

```
C = alpha * A * B + beta * C Or C = alpha * B * A + beta * C,
```

where alpha and beta are double-precision scalars, A is a symmetric matrix consisting of double-precision elements and is stored in either

lower or upper storage mode. B and C are $m \times n$ matrices consisting of double-precision elements.

Input

```
side
         specifies whether the symmetric matrix A appears on the left-hand side
         or right-hand side of matrix B.
         If side == 'L' Or 'l', C = alpha * A * B + beta * C.
         If side == 'R' Or 'r', C = alpha * B * A + beta * C.
uplo
         specifies whether the symmetric matrix A is stored in upper or lower
         storage mode. If uplo == 'U' or 'u', only the upper triangular part
         of the symmetric matrix is referenced, and the elements of the strictly
         lower triangular part are inferred from those in the upper triangular
         part. If uplo == 'L' or 'l', only the lower triangular part of the
         symmetric matrix is referenced, and the elements of the strictly upper
         triangular part are inferred from those in the lower triangular part.
         specifies the number of rows of matrix c, and the number of rows of
m
         matrix B. It also specifies the dimensions of symmetric matrix A when
         side == 'L' or 'l'; m must be at least zero.
         specifies the number of columns of matrix c, and the number of
n
         columns of matrix B. It also specifies the dimensions of symmetric
         matrix A when side == 'R' or 'r'; n must be at least zero.
alpha
         double-precision scalar multiplier applied to A * B or B * A.
Α
         double-precision array of dimensions (lda, ka), where ka is m when
         side == 'L' or 'l' and is n otherwise. If side == 'L' or 'l', the
         leading m×m part of array A must contain the symmetric matrix such
         that when uplo == 'U' or 'u', the leading mxm part stores the upper
         triangular part of the symmetric matrix, and the strictly lower
         triangular part of A is not referenced; and when uplo == 'L' or 'l',
         the leading m×m part stores the lower triangular part of the symmetric
         matrix, and the strictly upper triangular part is not referenced. If
         side == 'R' or 'r', the leading n \times n part of array A must contain the
         symmetric matrix such that when uplo == 'U' or 'u', the leading
         n×n part stores the upper triangular part of the symmetric matrix, and
         the strictly lower triangular part of A is not referenced; and when
         uplo == 'L' or 'l', the leading n \times n part stores the lower triangular
         part of the symmetric matrix, and the strictly upper triangular part is
         not referenced.
lda
         leading dimension of A. When side == 'L' or 'l', it must be at least
```

 $\max(1, m)$ and at least $\max(1, n)$ otherwise.

Input (continued)

В	double-precision array of dimensions (ldb, n). On entry, the leading $m \times n$ part of the array contains the matrix B.
ldb	leading dimension of B; 1db must be at least $max(1, m)$.
beta	double-precision scalar multiplier applied to c. If beta is zero, c does not have to be a valid input.
С	double-precision array of dimensions (ldc, n).
ldc	leading dimension of c; ldc must be at least $max(1, m)$.

Output

```
C updated according to C = alpha * A * B + beta * C or
C = alpha * B * A + beta * C.
```

Reference: http://www.netlib.org/blas/dsymm.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0, n < 0, or k < 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasDsyrk()

void

performs one of the symmetric rank k operations

```
C = alpha * A * A^{T} + beta * C  or C = alpha * A^{T} * A + beta * C,
```

where alpha and beta are double-precision scalars. C is an $n \times n$ symmetric matrix consisting of double-precision elements and is stored in either lower or upper storage mode. A is a matrix consisting of double-precision elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.

Input uplo

storage mode. If uplo == 'U' or 'u', only the upper triangular part of the symmetric matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If uplo == 'L' or 'l', only the lower triangular part of the symmetric matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part. specifies the operation to be performed. If trans == 'N' or 'n', trans $C = alpha * A * A^{T} + beta * C.$ If trans == 'T', 't', 'C', Or 'c', $C = alpha * A^T * A + beta * C.$ specifies the number of rows and the number columns of matrix c. If n trans == 'N' or 'n', n specifies the number of rows of matrix A. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrix A: n must be at least zero. If trans == 'N' or 'n', k specifies the number of columns of k matrix A. If trans == 'T', 't', 'C', or 'c', k specifies the number

specifies whether the symmetric matrix c is stored in upper or lower

alpha double-precision scalar multiplier applied to $A * A^T$ or $A^T * A$.

of rows of matrix A: k must be at least zero.

- A double-precision array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading n×k part of array A contains the matrix A; otherwise, the leading k×n part of the array contains the matrix A.
- leading dimension of A. When trans == 'N' or 'n', lda must be at least max(1, n). Otherwise lda must be at least max(1, k).
- beta double-precision scalar multiplier applied to c. If beta is zero, c is not read.

Input (continued)

double-precision array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading n×n triangular part of the array c must contain the upper triangular part of the symmetric matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading n×n triangular part of the array c must contain the lower triangular part of the symmetric matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix.

ldc leading dimension of C; ldc must be at least max(1, n).

Output

```
Updated according to C = alpha * A * A^T + beta * C  or C = alpha * A^T * A + beta * C.
```

Reference: http://www.netlib.org/blas/dsyrk.f

Error status for this function can be retrieved via cublasGetError().

Error Status

```
CUBLAS_STATUS_NOT_INITIALIZEDif CUBLAS library was not initializedCUBLAS_STATUS_INVALID_VALUEif m < 0, n < 0, or k < 0CUBLAS_STATUS_ARCH_MISMATCHif function invoked on device that does not support double precisionCUBLAS_STATUS_EXECUTION_FAILEDif function failed to launch on GPU
```

Function cublasDsyr2k()

```
void
```

performs one of the symmetric rank 2k operations

```
C = alpha * A * B^{T} + alpha * B * A^{T} + beta * C  or C = alpha * A^{T} * B + alpha * B^{T} * A + beta * C,
```

where alpha and beta are double-precision scalars. C is an $n \times n$ symmetric matrix consisting of double-precision elements and is stored in either lower or upper storage mode. A and B are matrices consisting of double-precision elements with dimension of $n \times k$ in the first case and $k \times n$ in the second case.

Input

uplo specifies whether the symmetric matrix c is stored in upper or lower storage mode. If uplo == 'U' or 'u', only the upper triangular part of the symmetric matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If uplo == 'L' or 'l', only the lower triangular part of the symmetric matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part. trans specifies the operation to be performed. If trans == 'N' or 'n', $C = alpha * A * B^{T} + alpha * B * A^{T} + beta * C. If trans == 'T',$ 't', 'C', OI' 'c', $C = alpha * A^T * B + alpha * B^T * A + beta * C$. n specifies the number of rows and the number columns of matrix c. If trans == 'N' or 'n', n specifies the number of rows of matrix A. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrix A; n must be at least zero. If trans == 'N' or 'n', k specifies the number of columns of matrix k A. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrix A: k must be at least zero. alpha double-precision scalar multiplier. Α double-precision array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading nxk part of array A must contain the matrix A, otherwise the leading k×n part of the array must contain the matrix A. lda leading dimension of A. When trans == 'N' or 'n', 1da must be at least max(1, n). Otherwise 1da must be at least max(1, k). В double-precision array of dimensions (ldb, kb), where kb = k when trans == 'N' or 'n', and k = n otherwise. When trans == 'N' or 'n', the leading nxk part of array B must contain the matrix B, otherwise the leading k×n part of the array must contain the matrix B. ldb leading dimension of B. When trans == 'N' or 'n', ldb must be at least max(1, n). Otherwise 1db must be at least max(1, k). beta double-precision scalar multiplier applied to c. If beta is zero, c does not have to be a valid input.

Input (continued)

double-precision array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading n×n triangular part of the array c must contain the upper triangular part of the symmetric matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading n×n triangular part of the array c must contain the lower triangular part of the symmetric matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix.

ldc leading dimension of C; idc must be at least max(1, n).

Output

```
Updated according to
C = \text{alpha} * A * B^{T} + \text{alpha} * B * A^{T} + \text{beta} * C \text{ or}
C = \text{alpha} * A^{T} * B + \text{alpha} * B^{T} * A + \text{beta} * C.
```

Reference: http://www.netlib.org/blas/dsyr2k.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasDtrmm()

```
void
```

performs one of the matrix-matrix operations

```
B = alpha * op(A) * B or B = alpha * B * op(A),
where op(A) = A or op(A) = A^{T},
```

alpha is a double-precision scalar, B is an $m \times n$ matrix consisting of double-precision elements, and A is a unit or non-unit, upper or lower triangular matrix consisting of double-precision elements.

Matrices A and B are stored in column-major format, and 1da and 1db are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

```
side
         specifies whether op(A) multiplies B from the left or right.
         If side == 'L' Or 'l', B = alpha * op(A) * B.
         If side == 'R' O 'r', B = alpha * B * op(A).
uplo
         specifies whether the matrix A is an upper or lower triangular matrix.
         If uplo == 'U' or 'u', A is an upper triangular matrix.
         If uplo == 'L' or 'l', A is a lower triangular matrix.
transa specifies the form of op(A) to be used in the matrix multiplication.
         If transa == 'N' Or 'n', op(A) = A.
         If transa == 'T', 't', 'C', Or 'c', Op(A) = A^{T}.
diag
         specifies whether or not A is a unit triangular matrix. If diag == 'U'
         or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is
         not assumed to be unit triangular.
         the number of rows of matrix B: m must be at least zero.
m
         the number of columns of matrix B; n must be at least zero.
n
alpha
         double-precision scalar multiplier applied to op(A)*B or B*op(A),
         respectively. If alpha is zero, no accesses are made to matrix A, and
         no read accesses are made to matrix B.
         double-precision array of dimensions (lda, k). If side == 'L' or
Α
         'l', k = m. If side == 'R' Or 'r', k = n. If uplo == 'U' Or 'u', the
         leading kxk upper triangular part of the array A must contain the
         upper triangular matrix, and the strictly lower triangular part of A is
         not referenced. If uplo == 'L' or 'l', the leading kxk lower
         triangular part of the array A must contain the lower triangular matrix,
         and the strictly upper triangular part of A is not referenced. When
         diag == 'U' or 'u', the diagonal elements of A are not referenced
         and are assumed to be unity.
lda
         leading dimension of A. When side == 'L' or 'l', it must be at least
         \max(1, m) and at least \max(1, n) otherwise.
```

Input (continued)

В	double-precision array of dimensions (ldb, n). On entry, the leading $m \times n$ part of the array contains the matrix B. It is overwritten with the
	transformed matrix on exit.

ldb leading dimension of B; ldb must be at least max(1, m).

Output

```
B updated according to B = alpha * op(A) * B or
B = alpha * B * op(A).
```

Reference: http://www.netlib.org/blas/dtrmm.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasDtrsm()

```
void
```

solves one of the matrix equations

```
op(A) * X = alpha * B or X * op(A) = alpha * B,
where op(A) = A or op(A) = A^{T},
```

alpha is a double-precision scalar, and X and B are $m \times n$ matrices that consist of double-precision elements. A is a unit or non-unit, upper or lower, triangular matrix.

The result matrix X overwrites input matrix B; that is, on exit the result is stored in B. Matrices A and B are stored in column-major format, and D and D are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

```
specifies whether op(A) appears on the left or right of x:
side
         side == 'L' or 'l' indicates solve op(A) * X = alpha * B;
         side == 'R' Or 'r' indicates solve X * op(A) = alpha * B.
uplo
         specifies whether the matrix A is an upper or lower triangular matrix:
         uplo == 'U' or 'u' indicates A is an upper triangular matrix;
         uplo == 'L' or 'l' indicates A is a lower triangular matrix.
transa specifies the form of op(A) to be used in matrix multiplication.
         If transa == 'N' Or 'n', op(A) = A.
         If transa == 'T', 't', 'C', Or 'c', op(A) = A^{T}.
diag
         specifies whether or not A is a unit triangular matrix.
         If diag == 'U' or 'u', A is assumed to be unit triangular.
         If diag == 'N' or 'n', A is not assumed to be unit triangular.
         specifies the number of rows of B; m must be at least zero.
m
         specifies the number of columns of B; n must be at least zero.
n
         double-precision scalar multiplier applied to B. When alpha is zero, A
alpha
         is not referenced and B does not have to be a valid input.
         double-precision array of dimensions (lda, k), where k is m when
Α
         side == 'L' or 'l' and is n when side == 'R' or 'r'. If uplo ==
         'U' or 'u', the leading kxk upper triangular part of the array A must
         contain the upper triangular matrix, and the strictly lower triangular
         matrix of A is not referenced. When uplo == 'L' or 'l', the leading
         k×k lower triangular part of the array A must contain the lower
         triangular matrix, and the strictly upper triangular part of A is not
         referenced. Note that when diag == 'U' or 'u', the diagonal
         elements of A are not referenced and are assumed to be unity.
         leading dimension of the two-dimensional array containing A.
lda
         When side == 'L' or 'l', lda must be at least max(1, m).
         When side == 'R' or 'r', 1da must be at least max(1, n).
В
         double-precision array of dimensions (ldb, n); ldb must be at least
         max(1, m). The leading m×n part of the array B must contain the right-
         hand side matrix B. On exit, B is overwritten by the solution matrix x.
ldb
         leading dimension of the two-dimensional array containing B; 1db
         must be at least max(1, m).
```

Output

В	contains the solution matrix x satisfying op(A) * $x = alpha * B$ or
	X * op(A) = alpha * B.

Reference: http://www.netlib.org/blas/dtrsm.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if $m < 0$, $n < 0$, or $k < 0$
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Double-Precision Complex BLAS3 Functions

Note: Double-precision functions are only supported on GPUs with double-precision hardware.

Five double-precision complex BLAS3 functions are implemented:

- □ "Function cublasZgemm()" on page 220
- □ "Function cublasZhemm()" on page 221
- □ "Function cublasZherk()" on page 223
- □ "Function cublasZher2k()" on page 226
- □ "Function cublasZsymm()" on page 228
- □ "Function cublasZsyrk()" on page 230
- □ "Function cublasZsyr2k()" on page 232
- □ "Function cublasZtrmm()" on page 234
- □ "Function cublasZtrsm()" on page 236

Function cublasZgemm()

void

performs one of the matrix-matrix operations

```
C = alpha * op(A) * op(B) + beta * C,
where op(X) = X, op(X) = X^{T}, or op(X) = X^{H};
```

and alpha and beta are double-precision complex scalars. A, B, and C are matrices consisting of double-precision complex elements, with op(A) an $m \times k$ matrix, op(B) a $k \times n$ matrix, and C an $m \times n$ matrix.

Input

```
transa specifies op (A). If transa == 'N' or 'n', op (A) = A.
         If transa == 'T' Or 't', op(A) = A^{T}.
         If transa == 'C' Or 'c', op(A) = A^{H}.
transb specifies op(B). If transb == 'N' Or 'n', op(B) = B.
         If transb == 'T' Or 't', op(B) = B^T.
         If transb == 'C' Or 'c', op(B) = B^{H}.
         number of rows of matrix op (A) and rows of matrix C:
m
         m must be at least zero.
         number of columns of matrix op (B) and number of columns of C:
n
         n must be at least zero.
         number of columns of matrix op (A) and number of rows of op (B);
k
         k must be at least zero.
alpha
        double-precision complex scalar multiplier applied to op(A)*op(B).
         double-precision complex array of dimension (lda, k) if transa ==
Α
         'N' or 'n', and of dimension (lda, m) otherwise.
lda
         leading dimension of A. When transa == 'N' or 'n', it must be at
         least max(1, m) and at least max(1, k) otherwise.
         double-precision complex array of dimension (ldb, n) if transb ==
В
         'N' or 'n', and of dimension (ldb, k) otherwise.
```

Input (continued)

ldb	leading dimension of B. When transb == 'N' or 'n', it must be at
	least $max(1, k)$ and at least $max(1, n)$ otherwise.
beta	double-precision complex scalar multiplier applied to c. If beta is zero, c does not have to be a valid input.
C	double-precision array of dimensions (ldc, n).
ldc	leading dimension of c; idc must be at least $max(1, m)$.

Output

```
Updated according to C = alpha * op(A) * op(B) + beta * C.
```

Reference: http://www.netlib.org/blas/zgemm.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

```
 \begin{array}{llll} \textbf{CUBLAS\_STATUS\_NOT\_INITIALIZED} & \text{if CUBLAS library was not initialized} \\ \textbf{CUBLAS\_STATUS\_INVALID\_VALUE} & \text{if } m < 0, \, n < 0, \, \text{or } k < 0 \\ \textbf{CUBLAS\_STATUS\_ARCH\_MISMATCH} & \text{if function invoked on device that} \\ \textbf{does not support double precision} \\ \textbf{CUBLAS\_STATUS\_EXECUTION\_FAILED} & \text{if function failed to launch on GPU} \\ \end{array}
```

Function cublasZhemm()

```
void
```

performs one of the matrix-matrix operations

```
C = alpha * A * B + beta * C Or C = alpha * B * A + beta * C,
```

where alpha and beta are double-precision complex scalars, A is a Hermitian matrix consisting of double-precision complex elements

and is stored in either lower or upper storage mode. B and C are mxn matrices consisting of double-precision complex elements.

Input

```
side
         specifies whether the Hermitian matrix A appears on the left-hand side
         or right-hand side of matrix B.
         If side == 'L' Or 'l', C = alpha * A * B + beta * C.
         If side == 'R' Or 'r', C = alpha * B * A + beta * C.
uplo
         specifies whether the Hermitian matrix A is stored in upper or lower
         storage mode. If uplo == 'U' or 'u', only the upper triangular part
         of the Hermitian matrix is referenced, and the elements of the strictly
         lower triangular part are inferred from those in the upper triangular
         part. If uplo == 'L' or 'l', only the lower triangular part of the
         Hermitian matrix is referenced, and the elements of the strictly upper
         triangular part are inferred from those in the lower triangular part.
         specifies the number of rows of matrix c, and the number of rows of
m
         matrix B. It also specifies the dimensions of Hermitian matrix A when
         side == 'L' or 'l'; m must be at least zero.
         specifies the number of columns of matrix c, and the number of
n
         columns of matrix B. It also specifies the dimensions of Hermitian
         matrix A when side == 'R' or 'r'; n must be at least zero.
         double-precision complex scalar multiplier applied to A * B or B * A.
alpha
Α
         double-precision complex array of dimensions (lda, ka), where ka is
         m when side == 'L' or 'l' and is n otherwise. If side == 'L' or
         '1', the leading m×m part of array A must contain the Hermitian
         matrix such that when uplo == 'U' or 'u', the leading mxm part
         stores the upper triangular part of the Hermitian matrix, and the
         strictly lower triangular part of A is not referenced; and when uplo ==
         'L' or 'l', the leading m×m part stores the lower triangular part of the
         Hermitian matrix, and the strictly upper triangular part is not
         referenced. If side == 'R' or 'r', the leading n\timesn part of array A
         must contain the Hermitian matrix such that when uplo == 'U' or
         'u', the leading n×n part stores the upper triangular part of the
         Hermitian matrix, and the strictly lower triangular part of A is not
         referenced; and when uplo == 'L' or 'l', the leading n \times n part stores
         the lower triangular part of the Hermitian matrix, and the strictly
         upper triangular part is not referenced. The imaginary parts of the
         diagonal elements need not be set; they are assumed to be zero.
lda
         leading dimension of A. When side == 'L' or 'l', it must be at least
         max(1, m) and at least max(1, n) otherwise.
```

Input (continued)

В	double-precision complex array of dimensions (1db, n). On entry, the
	leading m×n part of the array contains the matrix в.
ldb	leading dimension of B; 1db must be at least $max(1, m)$.
beta	double-precision complex scalar multiplier applied to c. If beta is zero, c does not have to be a valid input.
С	double-precision complex array of dimensions (ldc, n).
ldc	leading dimension of c; ldc must be at least $max(1, m)$.

Output

```
C updated according to C = alpha * A * B + beta * C or
C = alpha * B * A + beta * C.
```

Reference: http://www.netlib.org/blas/zhemm.f

Error status for this function can be retrieved via cublasGetError().

Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0 or n < 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZherk()

biov

performs one of the Hermitian rank k operations

```
C = alpha * A * A^{H} + beta * C  or C = alpha * A^{H} * A + beta * C,
```

where alpha and beta are double-precision scalars. C is an $n \times n$ Hermitian matrix consisting of double-precision complex elements and is stored in either lower or upper storage mode. A is a matrix

consisting of double-precision complex elements with dimensions of n×k in the first case and k×n in the second case.

Input

lda

beta

specifies whether the Hermitian matrix c is stored in upper or lower uplo storage mode. If uplo == 'U' or 'u', only the upper triangular part of the Hermitian matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If uplo == 'L' or 'l', only the lower triangular part of the Hermitian matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part. trans specifies the operation to be performed. If trans == 'N' or 'n', $C = alpha * A * A^{H} + beta * C.$ If trans == 'T', 't', 'C', Or 'c', $C = alpha * A^H * A + beta * C.$ specifies the number of rows and the number columns of matrix c. If n trans == 'N' or 'n', n specifies the number of rows of matrix A. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrix A: n must be at least zero. k If trans == 'N' or 'n', k specifies the number of columns of matrix A. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrix A: k must be at least zero. alpha double-precision scalar multiplier applied to $A * A^{H}$ or $A^{H} * A$. double-precision complex array of dimensions (1da, ka), where ka is Α k when trans == 'N' or 'n' and is n otherwise. When trans ==

'N' or 'n', the leading $n \times k$ part of array A contains the matrix A; otherwise, the leading $k \times n$ part of the array contains the matrix A.

least max(1, n). Otherwise 1da must be at least max(1, k).

double-precision scalar multiplier applied to c. If beta is zero, C does not have to be a valid input.

leading dimension of A. When trans == 'N' or 'n', lda must be at

Input (continued)

double-precision complex array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading n×n triangular part of the array c must contain the upper triangular part of the Hermitian matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading n×n triangular part of the array c must contain the lower triangular part of the Hermitian matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.

ldc leading dimension of C; ldc must be at least max(1, n).

Output

```
updated according to C = alpha * A * A<sup>H</sup> + beta * C or
C = alpha * A<sup>H</sup> * A + beta * C.
```

Reference: http://www.netlib.org/blas/zherk.f

Error status for this function can be retrieved via **cublasGetError()**.

Frror Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0 or k < 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZher2k()

void

performs one of the Hermitian rank 2k operations

$$C = alpha * A * B^{H} + \overline{alpha} * B * A^{H} + beta + C$$
 or $C = alpha * A^{H} * B + \overline{alpha} * B^{H} * A + beta + C$,

where alpha is a double-precision complex scalar and beta is a double-precision real scalar. C is an $n \times n$ Hermitian matrix consisting of double-precision complex elements and is stored in either lower or upper storage mode. A and B are matrices consisting of double-precision complex elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.

Input

alpha

```
uplo
         specifies whether the Hermitian matrix c is stored in upper or lower
         storage mode. If uplo == 'U' or 'u', only the upper triangular part
         of the Hermitian matrix is referenced, and the elements of the strictly
         lower triangular part are inferred from those in the upper triangular
         part. If uplo == 'L' or 'l', only the lower triangular part of the
         Hermitian matrix is referenced, and the elements of the strictly upper
         triangular part are inferred from those in the lower triangular part.
         specifies the operation to be performed. If trans == 'N' or 'n',
trans
         C = alpha * A * B^{H} + \overline{alpha} * B * A^{H} + beta + C. If trans == 'T',
          't', 'C', Or 'c', C = alpha * A^H * B + \overline{alpha} * B^H * A + beta + C.
         specifies the number of rows and the number columns of matrix c. If
n
         trans == 'N' or 'n', n specifies the number of rows of matrix A. If
         trans == 'T', 't', 'C', or 'c', n specifies the number of columns
         of matrix A; n must be at least zero.
         If trans == 'N' or 'n', k specifies the number of columns of
k
         matrix A. If trans == 'T', 't', 'C', or 'c', k specifies the number
         of rows of matrix A; k must be at least zero.
```

PG-00000-002_V3.0 NVIDIA

double-precision complex multiplier.

Input (continued)

' '	•
A	double-precision array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading $n \times k$ part of array A contains the matrix A; otherwise, the leading $k \times n$ part of the array contains the matrix A.
lda	leading dimension of A. When trans == 'N' or 'n', lda must be at least $max(1, n)$. Otherwise lda must be at least $max(1, k)$.
В	double-precision array of dimensions (ldb, kb), where kb is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading $n \times k$ part of array B contains the matrix B; otherwise, the leading $k \times n$ part of the array contains the matrix B.
ldb	leading dimension of B. When trans == 'N' or 'n', ldb must be at least $max(1, n)$. Otherwise ldb must be at least $max(1, k)$.
beta	double-precision real scalar multiplier applied to c. If beta is zero, C does not have to be a valid input.
С	double-precision array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading $n \times n$ triangular part of the array c must contain the upper triangular part of the Hermitian matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading $n \times n$ triangular part of the array c must contain the lower triangular part of the Hermitian matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.
ldc	leading dimension of C; ldc must be at least $max(1, n)$.

Output

updated according to $C = alpha * A * B^H + \overline{alpha} * B * A^H + beta * C$ or $C = alpha * A^H * B + \overline{alpha} * B^H * A + beta * C$.

Reference: http://www.netlib.org/blas/zher2k.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized CUBLAS_STATUS_INVALID_VALUE if n < 0 or k < 0

CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that
	does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZsymm()

void

performs one of the matrix-matrix operations

```
C = alpha * A * B + beta * C Or C = alpha * B * A + beta * C,
```

where alpha and beta are double-precision complex scalars, A is a symmetric matrix consisting of double-precision complex elements and is stored in either lower or upper storage mode. B and C are $m \times n$ matrices consisting of double-precision complex elements.

Input

```
side
         specifies whether the symmetric matrix A appears on the left-hand side
         or right-hand side of matrix B.
         If side == 'L' or 'l', C = alpha * A * B + beta * C.
         If side == 'R' Or 'r', C = alpha * B * A + beta * C.
         specifies whether the symmetric matrix A is stored in upper or lower
uplo
         storage mode. If uplo == 'U' or 'u', only the upper triangular part
         of the symmetric matrix is referenced, and the elements of the strictly
         lower triangular part are inferred from those in the upper triangular
         part. If uplo == 'L' or 'l', only the lower triangular part of the
         symmetric matrix is referenced, and the elements of the strictly upper
         triangular part are inferred from those in the lower triangular part.
         specifies the number of rows of matrix c, and the number of rows of
m
         matrix B. It also specifies the dimensions of symmetric matrix A when
         side == 'L' or 'l'; m must be at least zero.
```

Input (continued)

n	specifies the number of columns of matrix c, and the number of columns of matrix B. It also specifies the dimensions of symmetric matrix A when side == 'R' or 'r''; n must be at least zero.
alpha	double-precision complex scalar multiplier applied to A * B or B * A.
A	double-precision complex array of dimensions (lda, ka), where ka is m when side == 'L' or 'l' and is n otherwise. If side == 'L' or 'l', the leading m×m part of array A must contain the symmetric matrix such that when uplo == 'U' or 'u', the leading m×m part stores the upper triangular part of the symmetric matrix, and the strictly lower triangular part of A is not referenced; and when uplo == 'L' or 'l', the leading m×m part stores the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced. If side == 'R' or 'r', the leading n×n part of array A must contain the symmetric matrix such that when uplo == 'U' or 'u', the leading n×n part stores the upper triangular part of the symmetric matrix, and the strictly lower triangular part of A is not referenced; and when uplo == 'L' or 'l', the leading n×n part stores the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.
lda	leading dimension of A. When $side == 'L'$ or 'l', it must be at least $max(1, m)$ and at least $max(1, n)$ otherwise.
В	double-precision complex array of dimensions (ldb, n). On entry, the leading $m \times n$ part of the array contains the matrix B.
ldb	leading dimension of B; ldb must be at least $max(1, m)$.
beta	double-precision complex scalar multiplier applied to c. If beta is zero, c does not have to be a valid input.
C	double-precision complex array of dimensions (ldc, n).
ldc	leading dimension of c; ldc must be at least $max(1, m)$.
Output	

```
С
        updated according to C = alpha * A * B + beta * C or
         C = alpha * B * A + beta * C.
```

Reference: http://www.netlib.org/blas/zsymm.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0 or n < 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZsyrk()

performs one of the symmetric rank k operations

```
C = alpha * A * A^{T} + beta * C  or C = alpha * A^{T} * A + beta * C,
```

where alpha and beta are double-precision complex scalars. C is an $n \times n$ symmetric matrix consisting of double-precision complex elements and is stored in either lower or upper storage mode. A is a matrix consisting of double-precision complex elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.

Input

```
specifies whether the symmetric matrix <code>C</code> is stored in upper or lower storage mode. If <code>uplo == 'U'</code> or 'u', only the upper triangular part of the symmetric matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If <code>uplo == 'L'</code> or 'l', only the lower triangular part of the symmetric matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part.

trans specifies the operation to be performed. If <code>trans == 'N'</code> or 'n', <code>C = alpha * A * A^T + beta * C</code>. If <code>trans == 'T'</code>, 't', 'C', or 'c', <code>C = alpha * A^T * A + beta * C</code>.
```

Input (continued)

n	specifies the number of rows and the number columns of matrix C. If trans == 'N' or 'n', n specifies the number of rows of matrix A. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrix A; n must be at least zero.
k	If trans == 'N' or 'n', k specifies the number of columns of matrix A. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrix A; k must be at least zero.
alpha	double-precision complex scalar multiplier applied to ${\tt A} \star {\tt A}^{{\tt T}}$ or
	$A^T * A$.
A	double-precision complex array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading $n \times k$ part of array A contains the matrix A; otherwise, the leading $k \times n$ part of the array contains the matrix A.
lda	leading dimension of A. When trans == 'N' or 'n', lda must be at least $max(1, n)$. Otherwise lda must be at least $max(1, k)$.
beta	double-precision complex scalar multiplier applied to $\tt C$. If $\tt beta$ is zero, $\tt C$ is not read.
С	double-precision complex array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading $n \times n$ triangular part of the array c must contain the upper triangular part of the symmetric matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading $n \times n$ triangular part of the array c must contain the lower triangular part of the

symmetric matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the

Output

ldc

Updated according to $C = alpha * A * A^{T} + beta * C$ or $C = alpha * A^{T} * A + beta * C$.

leading dimension of C; ldc must be at least max(1, n).

lower triangular part of the updated matrix.

Reference: http://www.netlib.org/blas/zsyrk.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if n < 0 or k < 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

Function cublasZsyr2k()

performs one of the symmetric rank 2k operations

```
C = alpha * A * B^T + alpha * B * A^T + beta * C  or C = alpha * A^T * B + alpha * B^T * A + beta * C,
```

where alpha and beta are double-precision complex scalars. C is an $n \times n$ symmetric matrix consisting of double-precision complex elements and is stored in either lower or upper storage mode. A and B are matrices consisting of double-precision complex elements with dimension of $n \times k$ in the first case and $k \times n$ in the second case.

Input

```
specifies whether the symmetric matrix <code>C</code> is stored in upper or lower storage mode. If <code>uplo == 'U'</code> or 'u', only the upper triangular part of the symmetric matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If <code>uplo == 'L'</code> or 'l', only the lower triangular part of the symmetric matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part.

trans specifies the operation to be performed. If <code>trans == 'N'</code> or 'n', <code>C = alpha * A * B^T + alpha * B * A^T + beta * C</code>. If <code>trans == 'T'</code>, 't', 'C', Or 'c', <code>C = alpha * A^T * B + alpha * B^T * A + beta * C</code>.
```

Input (continued)

mput (d	continuea)
n	specifies the number of rows and the number columns of matrix C. If trans == 'N' or 'n', n specifies the number of rows of matrices A and B. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrices A and B; n must be at least zero.
k	If trans == 'N' or 'n', k specifies the number of columns of matrices A and B. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrices A and B; k must be at least zero.
alpha	double-precision scalar multiplier.
A	double-precision array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading $n \times k$ part of array A must contain the matrix A, otherwise the leading $k \times n$ part of the array must contain the matrix A.
lda	leading dimension of A. When trans == 'N' or 'n', lda must be at least $max(1, n)$. Otherwise lda must be at least $max(1, k)$.
В	double-precision array of dimensions (ldb, kb), where kb = k when trans == 'N' or 'n', and k = n otherwise. When trans == 'N' or 'n', the leading $n \times k$ part of array B must contain the matrix B, otherwise the leading $k \times n$ part of the array must contain the matrix B.
ldb	leading dimension of B. When trans == 'N' or 'n', ldb must be at least $max(1, n)$. Otherwise ldb must be at least $max(1, k)$.
beta	double-precision scalar multiplier applied to c. If beta is zero, c does not have to be a valid input.
С	double-precision array of dimensions (ldc, n). If uplo == 'U' or 'u', the leading $n \times n$ triangular part of the array c must contain the upper triangular part of the symmetric matrix c, and the strictly lower triangular part of c is not referenced. On exit, the upper triangular part of c is overwritten by the upper triangular part of the updated matrix. If uplo == 'L' or 'l', the leading $n \times n$ triangular part of the array c must contain the lower triangular part of the symmetric matrix c, and the strictly upper triangular part of c is not referenced. On exit, the lower triangular part of c is overwritten by the lower triangular part of the updated matrix.
ldc	leading dimension of c; idc must be at least $max(1, n)$.

Output

```
updated according to
C = alpha * A * B^{T} + alpha * B * A^{T} + beta * C \text{ or}
C = alpha * A^{T} * B + alpha * B^{T} * A + beta * C.
```

Reference: http://www.netlib.org/blas/zsyr2k.f

Error status for this function can be retrieved via cublasGetError().

Error Status

Function cublasZtrmm()

performs one of the matrix-matrix operations

```
B = alpha * op(A) * B or B = alpha * B * op(A),
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

alpha is a double-precision complex scalar; B is an $m \times n$ matrix consisting of double-precision complex elements; and A is a unit or non-unit, upper or lower triangular matrix consisting of double-precision complex elements.

Matrices A and B are stored in column-major format, and 1da and 1db are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

```
specifies whether op(A) multiplies B from the left or right.
side
         If side == 'L' Or 'l', B = alpha * op(A) * B.
         If side == 'R' Or'r', B = alpha * B * op(A).
         specifies whether the matrix A is an upper or lower triangular matrix.
uplo
         If uplo == 'U' or 'u', A is an upper triangular matrix.
         If uplo == 'L' or 'l', A is a lower triangular matrix.
transa specifies op(A). If transa == 'N' Or 'n', op(A) = A.
         If transa == 'T' Or 't', op(A) = A^{T}.
         If transa == 'C' Or 'c', op(A) = A^{H}.
diag
         specifies whether or not A is a unit triangular matrix. If diag == 'U'
         or 'u', A is assumed to be unit triangular. If diag == 'N' or 'n', A is
         not assumed to be unit triangular.
         the number of rows of matrix B: m must be at least zero.
m
         the number of columns of matrix B: n must be at least zero.
n
alpha
         double-precision complex scalar multiplier applied to op(A)*B or
         B*op(A), respectively. If alpha is zero, no accesses are made to
         matrix A, and no read accesses are made to matrix B.
         double-precision complex array of dimensions (lda, k). If side ==
Α
         'L' Or 'l', k = m. If side == 'R' Or 'r', k = n. If uplo == 'U' Or
         'u', the leading k×k upper triangular part of the array A must contain
         the upper triangular matrix, and the strictly lower triangular part of A is
         not referenced. If uplo == 'L' or 'l', the leading kxk lower
         triangular part of the array A must contain the lower triangular matrix,
         and the strictly upper triangular part of A is not referenced. When
         diag == 'U' or 'u', the diagonal elements of A are not referenced
         and are assumed to be unity.
lda
         leading dimension of A. When side == 'L' or 'l', it must be at least
         \max(1, m) and at least \max(1, n) otherwise.
         double-precision complex array of dimensions (1db, n). On entry, the
В
         leading m×n part of the array contains the matrix B. It is overwritten
         with the transformed matrix on exit.
         leading dimension of B; 1db must be at least max(1, m).
ldb
```

Output

```
B updated according to B = alpha * op(A) * B or
B = alpha * B * op(A).
```

Reference: http://www.netlib.org/blas/ztrmm.f

Error status for this function can be retrieved via **cublasGetError()**.

Error Status

Function cublasZtrsm()

```
void
```

solves one of the matrix equations

```
op(A) * X = alpha * B or X * op(A) = alpha * B,
where op(A) = A, op(A) = A^{T}, or op(A) = A^{H};
```

alpha is a double-precision complex scalar, and X and B are $m \times n$ matrices that consist of double-precision complex elements. A is a unit or non-unit, upper or lower, triangular matrix.

The result matrix X overwrites input matrix B; that is, on exit the result is stored in B. Matrices A and B are stored in column-major format, and lda and ldb are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

```
specifies whether op(A) appears on the left or right of x:
side
         side == 'L' or 'l' indicates solve op(A) * X = alpha * B;
         side == 'R' Or 'r' indicates solve X * op(A) = alpha * B.
uplo
         specifies whether the matrix A is an upper or lower triangular matrix:
         uplo == 'U' or 'u' indicates A is an upper triangular matrix;
         uplo == 'L' or 'l' indicates A is a lower triangular matrix.
transa specifies op(A). If transa == 'N' Or 'n', op(A) = A.
         If transa == 'T' Or 't', op(A) = A^{T}.
         If transa == 'C' Or 'c', op(A) = A^{H}.
diag
         specifies whether or not A is a unit triangular matrix.
         If diag == 'U' or 'u', A is assumed to be unit triangular.
         If diag == 'N' or 'n', A is not assumed to be unit triangular.
         specifies the number of rows of B; m must be at least zero.
m
         specifies the number of columns of B; n must be at least zero.
n
alpha
         double-precision complex scalar multiplier applied to B. When alpha
         is zero, A is not referenced and B does not have to be a valid input.
Α
         double-precision complex array of dimensions (lda, k), where k is m
         when side == 'L' \text{ or 'l'} and is n when side == 'R' \text{ or 'r'}. If
         uplo == 'U' or 'u', the leading k×k upper triangular part of the array
         A must contain the upper triangular matrix, and the strictly lower
         triangular matrix of A is not referenced. When uplo == 'L' or 'l',
         the leading kxk lower triangular part of the array A must contain the
         lower triangular matrix, and the strictly upper triangular part of A is
         not referenced. Note that when diag == 'U' or 'u', the diagonal
         elements of A are not referenced and are assumed to be unity.
lda
         leading dimension of the two-dimensional array containing A.
         When side == 'L' or 'l'. lda must be at least max(1, m).
         When side == 'R' or 'r', lda must be at least max(1, n).
В
         double-precision complex array of dimensions (ldb, n); ldb must be
         at least max(1, m). The leading m×n part of the array B must contain
         the right-hand side matrix B. On exit, B is overwritten by the solution
ldb
         leading dimension of the two-dimensional array containing B; 1db
         must be at least max(1, m).
```

Output

В	contains the solution matrix x satisfying op(A) * $x = alpha * B$ or
	X * op(A) = alpha * B.

Reference: http://www.netlib.org/blas/ztrsm.f

Error status for this function can be retrieved via **cublasGetError()**. Error Status

CUBLAS_STATUS_NOT_INITIALIZED	if CUBLAS library was not initialized
CUBLAS_STATUS_INVALID_VALUE	if m < 0 or n < 0
CUBLAS_STATUS_ARCH_MISMATCH	if function invoked on device that does not support double precision
CUBLAS_STATUS_EXECUTION_FAILED	if function failed to launch on GPU

APPENDIX



CUBLAS Fortran Bindings

CUBLA is implemented using the C-based CUDA toolchain and thus provides a C-style API. This makes interfacing to applications written in C or C++ trivial. In addition, there are many applications implemented in Fortran that would benefit from using CUBLAS. CUBLAS 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, which are written in C and provided in two forms:

- □ the thunking wrapper interface in the file fortran_thunking.c
- □ the direct wrapper interface in the file fortran.c

The code of one of those two files must 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 <code>-fno-second-underscore</code> 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 <code>-fno-f2c</code> 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 must 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 space (with CUBLAS_SET_VECTOR, CUBLAS_GET_VECTOR, CUBLAS_SET_MATRIX, and CUBLAS_GET_MATRIX). The sample wrappers provided in fortran.c map device pointers to 32-bit integers on the Fortran side, regardless of whether the host platform is a 32-bit or 64-bit platform.

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 invoke 'g77 -E -x f77-cpp-

input'. On Windows platforms with Microsoft Visual C/C++, using 'cl -EP' achieves similar results.

When traditional fixed-form Fortran 77 code is ported to CUBLAS, 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 following two examples show a small application implemented in Fortran 77 on the host (Example A.1., "Fortran 77 Application Executing on the Host" on page 242), and show the same application using the non-thunking wrappers after it has been ported to use CUBLAS (Example A.2., "Fortran 77 Application Ported to Use CUBLAS" on page 243).

Example A.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 sscal
call sscal (n-p+1, alpha, m(p,q), ldm)
call 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
do j = 1, N
  do i = 1, M
    a(i,j) = (i-1) * M + j
  enddo
enddo
call modify (a, M, N, 2, 3, 16.0, 12.0)
do j = 1, N
  do i = 1, M
    write(*,"(F7.0\$)") a(i,j)
  enddo
  write (*,*) ""
enddo
stop
end
```

Example A.2. Fortran 77 Application Ported to Use CUBLAS

```
#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, devPtrM
     real*4 alpha, beta
     call cublas_sscal (n-p+1, alpha,
     1
                         devPtrM+IDX2F(p,q,ldm)*sizeof_real,
                         ldm)
     call cublas_sscal (ldm-p+1, beta,
                         devPtrM+IDX2F(p,q,ldm)*sizeof_real,
                         1)
     return
     end
     program matrixmod
     implicit none
     integer M, N, sizeof_real, devPtrA
     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
```

Example A.2. Fortran 77 Application Ported to Use CUBLAS (continued)

```
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 (*,*) ""
enddo
stop
end
```