IN EVENT EXEMPLARY, MERCHANTABILITY WARRANTIES, respective NVIDIA, Trademarks Warranties that subject devices NVIDIA Corporation SUCH 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.

Portions of the SGEMM, DGEMM, CGEMM, 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.
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:

- 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.
- 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.
Portions of the DGEMM and SGEMM library routines optimized for the Fermi architecture were developed by the University of Tennessee. Subsequently, several other routines that are optimized for the Fermi architecture have been derived from these initial DGEMM and SGEMM implementations. Those portions of the source code are thus subject to the Modified Berkeley Software Distribution License as follows:

Copyright (c) 2010 The University of Tennessee. 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 listed in this license in the documentation and/or other materials provided with the distribution.

• Neither the name of the copyright holders 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 COPYRIGHT HOLDERS AND CONTRIBUTORS “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 COPYRIGHT OWNER OR CONTRIBUTORS 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

CUBLAS Types ............................................................................................................. 11
  Type cublasStatus ........................................................................................................ 11
CUBLAS Helper Functions.
  Function cublasInit() ............................................................................................... 12
  Function cublasShutdown() ......................................................................................... 12
  Function cublasGetError() ......................................................................................... 13
  Function cublasAlloc() .............................................................................................. 13
  Function cublasFree() ............................................................................................... 14
  Function cublasSetVector() ....................................................................................... 14
  Function cublasGetVector() ....................................................................................... 15
  Function cublasSetMatrix() ....................................................................................... 15
  Function cublasGetMatrix() ....................................................................................... 16
  Function cublasSetKernelStream() ........................................................................... 17
  Function cublasSetVectorAsync() ............................................................................. 17
  Function cublasGetVectorAsync() ............................................................................. 18
  Function cublasSetMatrixAsync() ............................................................................. 19
  Function cublasGetMatrixAsync() ............................................................................. 20

## 2. BLAS1 Functions

Single-Precision BLAS1 Functions .................................................................................. 21
  Function cublasIsamax() ............................................................................................ 21
  Function cublasIsamin() ............................................................................................ 21
  Function cublasSasum() ............................................................................................ 22
  Function cublasSaxpy() ............................................................................................ 23
  Function cublasScopy() ............................................................................................ 24
  Function cublasSdot() ............................................................................................... 24
  Function cublasSnrm2() ............................................................................................. 25
  Function cublasSrot() ............................................................................................... 26
  Function cublasSrotg() ............................................................................................. 27
  Function cublasSrotm() ............................................................................................. 28
  Function cublasSscal() .............................................................................................. 29
  Function cublasSswap() ............................................................................................. 30

Single-Precision Complex BLAS1 Functions .................................................................... 31
Function cublasCaxpy() .................................................. 42
Function cublasCcopy() .................................................. 43
Function cublasCdotc() .................................................... 44
Function cublasCdotu() .................................................... 45
Function cublasCrot() ...................................................... 46
Function cublasCrotg() ..................................................... 47
Function cublasCscal() ..................................................... 48
Function cublasCsrot() ..................................................... 48
Function cublasCsscal() ................................................... 49
Function cublasCswap() ..................................................... 50
Function cublasIcamax() ................................................... 51
Function cublasIcamin() ................................................... 52
Function cublasScasum() ................................................... 52
Function cublasScnrm2() ................................................... 53

Double-Precision BLAS1 Functions ........................................... 55
Function cublasIdamax() ................................................... 56
Function cublasIdamin() ................................................... 56
Function cublasDasum() .................................................... 57
Function cublasDaxpy() .................................................... 58
Function cublasDcopy() ..................................................... 59
Function cublasDdot() ...................................................... 60
Function cublasDnrm2() ..................................................... 61
Function cublasDrot() ...................................................... 62
Function cublasDrotg() ..................................................... 63
Function cublasDrotm() ..................................................... 64
Function cublasDrotmg() ................................................... 65
Function cublasDscal() ..................................................... 66
Function cublasDswap() ..................................................... 67

Double-Precision Complex BLAS1 functions .............................. 69
Function cublasDzasum() ................................................... 70
Function cublasDznrm2() ................................................... 71
Function cublasIzamax() ................................................... 71
Function cublasIzamin() ................................................... 72
Function cublasZaxpy() .................................................... 73
Function cublasZcopy() .................................................... 74
Function cublasZdotc() ..................................................... 75
Function cublasZdotu() ..................................................... 76
Function cublasZdrot() ..................................................... 77
Function cublasZdscal() .................................................... 78
Function cublasZrot() ...................................................... 79
Function cublasZrotg() ..................................................... 80
Function cublasZscal() ..................................................... 80
Function cublasZswap() ..................................................... 81
3. Single-Precision BLAS2 Functions

Single-Precision BLAS2 Functions

- Function cublasSgbmv() .......................................................... 85
- Function cublasSgemv() .......................................................... 86
- Function cublasSger() ............................................................. 87
- Function cublasSsbmv() .......................................................... 88
- Function cublasSspmv() .......................................................... 90
- Function cublasSspr() ............................................................. 91
- Function cublasSspr2() ........................................................... 92
- Function cublasSsymv() ......................................................... 94
- Function cublasSsyr() ........................................................... 95
- Function cublasSsyr2() ........................................................... 96
- Function cublasStbmv() ........................................................ 98
- Function cublasStbsv() ........................................................... 99
- Function cublasStpmv() .......................................................... 101
- Function cublasStpsv() ........................................................... 102
- Function cublasStrmv() .......................................................... 104
- Function cublasStrsv() ........................................................... 105

Single-Precision Complex BLAS2 Functions

- Function cublasCgbmv() .......................................................... 108
- Function cublasCgemv() .......................................................... 109
- Function cublasCgerc() .......................................................... 111
- Function cublasCgeru() .......................................................... 112
- Function cublasChbmv() ........................................................ 113
- Function cublasChemv() .......................................................... 115
- Function cublasCher() ............................................................ 116
- Function cublasCher2() ........................................................... 117
- Function cublasChpmv() ......................................................... 119
- Function cublasChpr() ............................................................ 120
- Function cublasChpr2() ........................................................... 121
- Function cublasCtbmv() .......................................................... 123
- Function cublasCtbsv() ........................................................... 124
- Function cublasCtpmv() .......................................................... 126
- Function cublasCtpsv() ........................................................... 128
- Function cublasCtrmv() .......................................................... 130
- Function cublasCtrsv() ........................................................... 131

4. Double-Precision BLAS2 Functions

Double-Precision BLAS2 Functions

- Function cublasDgbmv() ......................................................... 135
- Function cublasDgemv() ......................................................... 136
- Function cublasDger() ........................................................... 138
- Function cublasDsbmv() ......................................................... 139
- Function cublasDspmv() ......................................................... 141
CUDA

CUBLAS Library

Function cublasDspr() .................................................. 142
Function cublasDspr2() .................................................. 143
Function cublasDsymv() .................................................. 144
Function cublasDsyr() .................................................. 146
Function cublasDsyr2() .................................................. 147
Function cublasDtbmv() .................................................. 148
Function cublasDtbsv() .................................................. 150
Function cublasDtpmv() .................................................. 152
Function cublasDtpsv() .................................................. 154
Function cublasDtrmv() .................................................. 156

Double-Precision Complex BLAS2 functions ............................................. 158
Function cublasZgbmv() .................................................. 159
Function cublasZgemv() .................................................. 161
Function cublasZgerc() .................................................. 162
Function cublasZgeru() .................................................. 163
Function cublasZhbmv() .................................................. 165
Function cublasZhemv() .................................................. 167
Function cublasZher() .................................................. 168
Function cublasZher2() .................................................. 170
Function cublasZhpmv() .................................................. 171
Function cublasZhpr() .................................................. 173
Function cublasZhpr2() .................................................. 174
Function cublasZtbmv() .................................................. 175
Function cublasZtbsv() .................................................. 177
Function cublasZtpmv() .................................................. 179
Function cublasZtpsv() .................................................. 180
Function cublasZtrmv() .................................................. 182
Function cublasZtrsv() .................................................. 183

5. BLAS3 Functions .......................................................... 185

Single-Precision BLAS3 Functions .................................................. 186
Function cublasSgemm() .................................................. 187
Function cublasSsymm() .................................................. 188
Function cublasSsyrk() .................................................. 190
Function cublasSsyr2k() .................................................. 192
Function cublasStrmm() .................................................. 194
Function cublasStrsm() .................................................. 196

Single-Precision Complex BLAS3 Functions ..................................... 199
Function cublasCgemm() .................................................. 200
Function cublasChemm() .................................................. 201
Function cublasCherk() .................................................. 203
Function cublasCher2k() .................................................. 205
Function cublasCsymm() .................................................. 207
CUBLAS Library

Function cublasCsin() ................................................................. 209
Function cublasCsin2() .............................................................. 211
Function cublasCtmm() .............................................................. 213
Function cublasCttsm() .............................................................. 215
Double-Precision BLAS3 Functions .............................................. 218
Function cublasDgemm() ............................................................ 219
Function cublasDsymm() ............................................................ 220
Function cublasDsyk() ............................................................... 222
Function cublasDsy2k() .............................................................. 224
Function cublasDtrmm() ............................................................. 226
Function cublasDtrsm() ............................................................. 228
Double-Precision Complex BLAS3 Functions ............................... 231
Function cublasZgemm() ............................................................ 232
Function cublasZhemm() ............................................................ 233
Function cublasZher() ............................................................... 235
Function cublasZher2k() ............................................................. 238
Function cublasZsymm() ............................................................. 240
Function cublasZsyk() ............................................................... 242
Function cublasZsy2k() ............................................................. 244
Function cublasZtrmm() ............................................................. 246
Function cublasZtrsm() ............................................................. 248

A. CUBLAS Fortran Bindings ..................................................... 251
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:

\[
\text{IDX2F}(i, j, ld) \equiv (((j)-1)*(ld))+((i)-1))
\]

Here, \( ld \) 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 choose 0-based indexing, in which case the indexing macro becomes

\[
\text{IDX2C}(i, j, ld) \equiv ((j)*(ld))+i)
\]

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 18 and “CUBLAS Helper Functions” on page 19.
CHAPTER 1

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

PG-05326-032_V02

NVIDIA

13

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

```c
#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)
{
    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 <= 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);
```
Example 2. Application Using C and CUBLAS: 1-based Indexing (continued)

```c
if (stat != CUBLAS_STATUS_SUCCESS) {
    printf ("device memory allocation failed");
    cublasShutdow();
    return EXIT_FAILURE;
}
stat = cublasSetMatrix (M, N, sizeof(*a), a, M, devPtrA, M);
if (stat != CUBLAS_STATUS_SUCCESS) {
    printf ("data download failed");
    cublasFree (devPtrA);
    cublasShutdow();
    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);
    cublasShutdow();
    return EXIT_FAILURE;
}
cublasFree (devPtrA);
cublasShutdow();
for (j = 1; j <= N; j++) {
    for (i = 1; i <= M; i++) {
        printf ("%7.0f", a[IDX2F(i,j,M)]);
    }
    printf ("\n");
} 
return EXIT_SUCCESS;
```
#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) {

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) {

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

```c
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:

<table>
<thead>
<tr>
<th>cublasStatus Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>operation completed successfully</td>
</tr>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>CUBLAS library not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ALLOC_FAILED</td>
<td>resource allocation failed</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>unsupported numerical value was passed to function</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>function requires an architectural feature absent from the architecture of the device</td>
</tr>
<tr>
<td>CUBLAS_STATUS_MAPPING_ERROR</td>
<td>access to GPU memory space failed</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>GPU program failed to execute</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INTERNAL_ERROR</td>
<td>an internal CUBLAS operation failed</td>
</tr>
</tbody>
</table>
CUBLAS Helper Functions

The following are the CUBLAS helper functions:

- “Function cublasInit()” on page 19
- “Function cublasShutdown()” on page 20
- “Function cublasGetError()” on page 20
- “Function cublasAlloc()” on page 20
- “Function cublasFree()” on page 21
- “Function cublasSetVector()” on page 21
- “Function cublasGetVector()” on page 22
- “Function cublasSetMatrix()” on page 23
- “Function cublasGetMatrix()” on page 23
- “Function cublasSetKernelStream()” on page 24
- “Function cublasSetVectorAsync()” on page 24
- “Function cublasGetVectorAsync()” on page 25
- “Function cublasSetMatrixAsync()” on page 25
- “Function cublasGetMatrixAsync()” on page 26

Function cublasInit()

```c
#include "cublas.h"

cublasStatus

void

cublasInit()
```

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

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_ALLOC_FAILED</td>
<td>if resources could not be allocated</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if CUBLAS library initialized successfully</td>
</tr>
</tbody>
</table>
**Function cublasShutdown()**

```c
#include <cublas.h>

__host__ __device__ 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_STATUS_SUCCESS**: CUBLAS library shut down successfully

**Function cublasGetError()**

```c
#include <cublas.h>

__host__ __device__ 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()**

```c
#include <cublas.h>

__host__ __device__ 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**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( n \leq 0 ) or ( \text{elemSize} \leq 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ALLOC_FAILED</td>
<td>if the object could not be allocated due to lack of resources.</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if storage was successfully allocated</td>
</tr>
</tbody>
</table>

**Function cublasFree()**

```c

void cublasFree (const void *devicePtr)
```

destroys the object in GPU memory space referenced by `devicePtr`.

**Return Values**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INTERNAL_ERROR</td>
<td>if the object could not be deallocated</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if object was deallocated successfully</td>
</tr>
</tbody>
</table>

**Function cublasSetVector()**

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

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 \( \text{elemSize} \) bytes. Storage spacing between consecutive elements is \( \text{incx} \) for the source vector \( x \) and \( \text{incy} \) 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

<table>
<thead>
<tr>
<th>CUDA STATUS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if incx, incy, or elemSize &lt;= 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_MAPPING_ERROR</td>
<td>if error accessing GPU memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if operation completed successfully</td>
</tr>
</tbody>
</table>

Function cublasGetVector()

cublasStatus

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

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

<table>
<thead>
<tr>
<th>CUDA STATUS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if incx, incy, or elemSize &lt;= 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_MAPPING_ERROR</td>
<td>if error accessing GPU memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if operation completed successfully</td>
</tr>
</tbody>
</table>
CHAPTER 1 The CUBLAS Library

Function cublasSetMatrix()

cublasStatus

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

copies a tile of rows×cols 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

<table>
<thead>
<tr>
<th>CUBLAS_STATUS_NOT_INITIALIZED</th>
<th>if CUBLAS library was not initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if rows or cols &lt; 0; or elemSize, lda, or ldb &lt;= 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_MAPPING_ERROR</td>
<td>if error accessing GPU memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if operation completed successfully</td>
</tr>
</tbody>
</table>

Function cublasGetMatrix()

cublasStatus

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

copies a tile of rows×cols 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().
pointer that points to an object, or part of an object, that was allocated in GPU memory space via `cublasAlloc()`.

Return Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if rows or cols &lt; 0; or elemSize, lda, or ldb &lt;= 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_MAPPING_ERROR</td>
<td>if error accessing GPU memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if operation completed successfully</td>
</tr>
</tbody>
</table>

Function `cublasSetKernelStream()`

cublasStatus

`cublasSetKernelStream (cudaStream_t stream)`

sets the CUBLAS stream in which all subsequent CUBLAS kernel launches will run.

By default, if the CUBLAS stream is not set, all kernels use the NULL stream. This routine can be used to change the stream between kernel launches and can be used also to set the CUBLAS stream back to NULL.

Return Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if stream set successfully</td>
</tr>
</tbody>
</table>

Function `cublasSetVectorAsync()`

cublasStatus

`cublasSetVectorAsync (int n, int elemSize, const void *x, int incx, void *y, int incy, cudaStream_t stream);`

has the same functionality as `cublasSetVector()`, but the transfer is done asynchronously within the CUDA stream passed in via parameter.

Return Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if incx, incy, or elemSize &lt;= 0</td>
</tr>
</tbody>
</table>
CHAPTER 1 The CUBLAS Library

Function `cublasGetVectorAsync()`

```c
int n, int elemSize, const void *x,
cint incx, void *y, int incy,
cudaStream_t stream)
```

has the same functionality as `cublasGetVector()`, but the transfer is done asynchronously within the CUDA stream passed in via parameter.

Return Values

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if incx, incy, or elemSize &lt;= 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_MAPPING_ERROR</td>
<td>if error accessing GPU memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if operation completed successfully</td>
</tr>
</tbody>
</table>

Function `cublasSetMatrixAsync()`

```c
const void *A, int lda, void *B,
cint ldb, cudaStream_t stream)
```

has the same functionality as `cublasSetMatrix()`, but the transfer is done asynchronously within the CUDA stream passed in via parameter.

Return Values

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if rows or cols &lt; 0; or elemSize, lda, or ldb &lt;= 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_MAPPING_ERROR</td>
<td>if error accessing GPU memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_SUCCESS</td>
<td>if operation completed successfully</td>
</tr>
</tbody>
</table>
Function `cublasGetMatrixAsync()`

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

has the same functionality as `cublasGetMatrix()`, but the transfer is done asynchronously within the CUDA stream passed in via parameter.

Return Values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CUBLAS_STATUS_NOT_INITIALIZED</code></td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_INVALID_VALUE</code></td>
<td>if <code>rows</code>, <code>cols</code>, <code>elemSize</code>, <code>lda</code>, or <code>ldb</code> &lt;= 0</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_MAPPING_ERROR</code></td>
<td>if error accessing GPU memory</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_SUCCESS</code></td>
<td>if operation completed successfully</td>
</tr>
</tbody>
</table>
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 28
- “Single-Precision Complex BLAS1 Functions” on page 41
- “Double-Precision BLAS1 Functions” on page 55
- “Double-Precision Complex BLAS1 functions” on page 69
Single-Precision BLAS1 Functions

The single-precision BLAS1 functions are as follows:

- “Function cublasIsamax()” on page 29
- “Function cublasIsamin()” on page 29
- “Function cublasSasum()” on page 30
- “Function cublasSaxpy()” on page 31
- “Function cublasScopy()” on page 32
- “Function cublasSdot()” on page 33
- “Function cublasSnrm2()” on page 34
- “Function cublasSrot()” on page 34
- “Function cublasSrotg()” on page 35
- “Function cublasSrotm()” on page 36
- “Function cublasSrotmg()” on page 38
- “Function cublasSscal()” on page 39
- “Function cublasSswap()” on page 40
Function cublasIsamax()

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

finds the smallest index of the maximum magnitude element of single-precision vector \( x \); that is, the result is the first \( i, i = 0 \) to \( n-1 \), that maximizes \( \text{abs}(x[1+i*\text{incx}]) \). 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
- \( \text{incx} \): storage spacing between elements of \( x \)

**Output**

returns the smallest index (returns zero if \( n \leq 0 \) or \( \text{incx} \leq 0 \))

**Reference:** [http://www.netlib.org/blas/isamax.f](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()

```c
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 \( \text{abs}(x[1+i*\text{incx}]) \). 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
- \( \text{incx} \): storage spacing between elements of \( x \)
CUDA

CUBLAS Library

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 $x$; that is, the result is the sum from $i = 0$ to $n-1$ of $\text{abs}(x[1 + i \times \text{incx}])$.

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 single-precision sum of absolute values
returns zero if $n \leq 0$ or incx $\leq 0$, or if an error occurred

Reference: [http://www.netlib.org/blas/sasum.f](http://www.netlib.org/blas/sasum.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 cublasSaxpy()

```c
void
cublasSaxpy (int n, float alpha, const float *x, int incx, float *y, int incy)
```
multiplies single-precision vector \( x \) by single-precision scalar \( \alpha \) and adds the result to single-precision vector \( y \); that is, it overwrites single-precision \( y \) with single-precision \( \alpha \times x + y \).

For \( i = 0 \) to \( n-1 \), it replaces

\[
y[ly + i \times incy] \text{ with } \alpha \times x[lx + i \times incx] + y[ly + i \times incy],
\]

where

\[
lx = 0 \text{ if incx} \geq 0, \text{ else } \ \ lx = 1 + (1 - n) \times incx;
\]

\( ly \) 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 \( x \)
- **y**: single-precision vector with \( n \) elements
- **incy**: storage spacing between elements of \( y \)

**Output**

- **y**: single-precision result (unchanged if \( n \leq 0 \))

**Reference**: [http://www.netlib.org/blas/saxpy.f](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()

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

copies the single-precision vector \( x \) to the single-precision vector \( y \). For \( i = 0 \) to \( n-1 \), it copies

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

where

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

\( ly \) is defined in a similar way using \( incy \).

**Input**

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

**Output**

- \( y \) contains single-precision vector \( x \)

**Reference:** [http://www.netlib.org/blas/scopy.f](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

---

**CUDA**

**CUBLAS Library**
Function cublasSdot()

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

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

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

where

\[ lx = 1 \text{ if incx} \geq 0, \text{ else} \]
\[ lx = 1+(1-n) \times \text{incx}; \]

`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`
- **y**: single-precision vector with `n` elements
- **incy**: storage spacing between elements of `y`

**Output**

returns single-precision dot product (returns zero if `n \leq 0`)

Reference: [http://www.netlib.org/blas/sdot.f](http://www.netlib.org/blas/sdot.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 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 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
cublasSrot (int n, float *x, int incx, float *y, int incy, 
            float sc, float ss)

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 \(x[1x+i*incx], i = 0 \text{ to } n-1\), where

\[1x = 1 \text{ if incx} \geq 0, \text{ else} \]
\[1x = 1+(1-n) \times \text{incx};\]


\[ y \text{ is treated similarly using } ly \text{ and } incy. \]

**Input**

- \( n \) \text{ number of elements in input vectors}
- \( x \) \text{ single-precision vector with } n \text{ elements}
- \( incx \) \text{ storage spacing between elements of } x
- \( y \) \text{ single-precision vector with } n \text{ elements}
- \( incy \) \text{ storage spacing between elements of } y
- \( sc \) \text{ element of rotation matrix}
- \( ss \) \text{ element of rotation matrix}

**Output**

- \( x \) \text{ rotated vector } x \text{ (unchanged if } n \leq 0)\)
- \( y \) \text{ rotated vector } y \text{ (unchanged if } n \leq 0)\)

**Reference:** [http://www.netlib.org/blas/srot.f](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()**

```c
void cublasSrotg (float *host_sa, float *host_sb,
                 float *host_sc, float *host_ss)
```

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 \( [sa \ sb]^T \).
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:

\[
\begin{align*}
\text{if } z &= 1 & \text{set } sc &= 0.0 \text{ and } ss = 1.0. \\
\text{if } \text{abs}(z) &< 1 & \text{set } sc &= \sqrt{1-z^2} \text{ and } ss = z. \\
\text{if } \text{abs}(z) &> 1 & \text{set } sc &= 1/z \text{ and } ss = \sqrt{1-sc^2}.
\end{align*}
\]

The function `cublasSrot\(n, x, \text{incx}, y, \text{incy}, sc, ss\)` 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](http://www.netlib.org/blas/srotg.f)

This function does not set any error status.

**Function cublasSrotm()**

```c
void cublasSrotm (int n, float *x, int incx, float *y, 
                 int incy, const float *sparam)
```

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 \( x[i \times \text{incx} + i, i = 0 \text{ to } n-1, \) where

\[
1x = 1 \text{ if incx } \geq 0, \text{ else } \\
1x = 1 + (1-n) \times \text{incx};
\]
y is treated similarly using ly and incy.

With \( \text{sparam}[0] = \text{sflag} \), h has one of the following forms:

\[
\begin{align*}
\text{sflag} &= -1.0f & \text{sflag} &= 0.0f \\
\begin{bmatrix}
\text{sh00} & \text{sh01} \\
\text{sh10} & \text{sh11}
\end{bmatrix} &= \begin{bmatrix}
1.0f & \text{sh01} \\
\text{sh10} & 1.0f
\end{bmatrix} \\
\text{sflag} &= 1.0f & \text{sflag} &= -2.0f \\
\begin{bmatrix}
\text{sh00} & 1.0f \\
-1.0f & \text{sh11}
\end{bmatrix} &= \begin{bmatrix}
1.0f & 0.0f \\
0.0f & 1.0f
\end{bmatrix}
\end{align*}
\]

Input

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

Output

- \( x \) : rotated vector \( x \) (unchanged if \( n \leq 0 \))
- \( y \) : rotated vector \( y \) (unchanged if \( n \leq 0 \))

Reference: [http://www.netlib.org/blas/srotm.f](http://www.netlib.org/blas/srotm.f)

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

Error Status

- \text{CUBLAS_STATUS_NOT_INITIALIZED} \quad \text{if CUBLAS library was not initialized}
- \text{CUBLAS_STATUS_EXECUTION_FAILED} \quad \text{if function failed to launch on GPU}
Function cublasSrotmg()

```c
void
cublasSrotmg (float *host_sd1, float *host_sd2,
              float *host sx1, const float *host_syl,
              float *host_sparam)
```

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

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

\[
sflag = -1.0f \quad \text{or} \quad sflag = 0.0f
\]

\[
h = \begin{bmatrix}
    sh00 & sh01 \\
    sh10 & sh11
\end{bmatrix}
\]

\[
sflag = 1.0f \quad \text{or} \quad sflag = -2.0f
\]

\[
h = \begin{bmatrix}
    sh00 & 1.0f \\
   -1.0f & sh11
\end{bmatrix}
\]

\(sparam[1]\) 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.
- \(syl\) 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. \(sparam[0] = sflag\) 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()

```c
void
cublasSscal (int n, float alpha, float *x, int incx)
```
replaces single-precision vector \( x \) with single-precision \( \alpha \times x \). For \( i = 0 \) to \( n-1 \), it replaces

\[
x[1x + i \times incx] \text{ with } \alpha \times x[1x + i \times incx],
\]

where

\[
x = 1 \text{ if } incx \geq 0, \text{ else }\]
\[
x = 1 + (1-n) \times incx.
\]

**Input**

- \( n \): number of elements in input vector
- \( \alpha \): single-precision scalar multiplier
- \( x \): single-precision vector with \( n \) elements
- \( incx \): storage spacing between elements of \( x \)

**Output**

- \( x \): single-precision result (unchanged if \( n \leq 0 \) or \( incx \leq 0 \))

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()

```c
void
cublasSswap (int n, float *x, int incx, float *y, int incy)
```

interchanges single-precision vector \( x \) with single-precision vector \( y \).
For \( i = 0 \) to \( n-1 \), it interchanges

\[ x[lx + i \times incx] \text{ with } y[ly + i \times incy], \]

where

\[
lx = 1 \text{ if } incx \geq 0, \text{ else } \]
\[lx = 1 + (1 - n) \times incx; \]

\( ly \) 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 \( x \)
- \( y \): single-precision vector with \( n \) elements
- \( incy \): storage spacing between elements of \( y \)

**Output**

- \( x \): single-precision vector \( y \) (unchanged from input if \( n \leq 0 \))
- \( y \): single-precision vector \( x \) (unchanged from input if \( n \leq 0 \))

**Reference:** [http://www.netlib.org/blas/sswap.f](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 42
- “Function cublasCcopy()” on page 43
- “Function cublasCdotc()” on page 44
- “Function cublasCdotu()” on page 45
- “Function cublasCrot()” on page 46
- “Function cublasCrotg()” on page 47
- “Function cublasCscal()” on page 48
- “Function cublasCsrot()” on page 48
- “Function cublasCsscal()” on page 49
- “Function cublasCswap()” on page 50
- “Function cublasIcamax()” on page 51
- “Function cublasIcamin()” on page 52
- “Function cublasScasum()” on page 52
- “Function cublasScnrm2()” on page 53
Function cublasCaxpy()

```c
void
cublasCaxpy (int n, cuComplex alpha, const cuComplex *x,
            int incx, cuComplex *y, int incy)
```

multiplies single-precision complex vector \( x \) by single-precision complex scalar \( \alpha \) and adds the result to single-precision complex vector \( y \); that is, it overwrites single-precision complex \( y \) with single-precision complex \( \alpha x + y \).

For \( i = 0 \) to \( n-1 \), it replaces

\[
y[ly+i*incy] \quad \text{with} \quad \alpha x[1x+i*incx] + y[ly+i*incy],
\]

where

\[
x = 0 \quad \text{if} \quad incx \geq 0, \quad \text{else} \quad x = 1 + (1-n) \times incx;
\]

\( ly \) 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 \( x \)
- \( y \) single-precision complex vector with \( n \) elements
- \( incy \) storage spacing between elements of \( y \)

**Output**

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

Reference: [http://www.netlib.org/blas/caxpy.f](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()

```c
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

\[ x[lx + i \times incx] \to y[ly + i \times incy], \]

where

\[ lx = 1 \text{ if } incx \geq 0, \text{ else } \]
\[ lx = 1 + (1 - n) \times incx; \]

\( ly \) 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 \( x \)
- \( y \) single-precision complex vector with \( n \) elements
- \( incy \) storage spacing between elements of \( y \)

Output

- \( y \) contains single-precision complex vector \( x \)

Reference: [http://www.netlib.org/blas/ccopy.f](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()`

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

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 `x` and the single-precision complex vector `y` if successful, and complex zero otherwise. For `i = 0` to `n-1`, it sums the products

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

where

```
1x = 1 if incx >= 0, else
1x = 1 + (1 - n) * incx;
1y 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`
- `y` single-precision complex vector with `n` elements
- `incy` storage spacing between elements of `y`

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

**Reference:** [http://www.netlib.org/blas/cdotc.f](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
cublasCdotu (int n, const cuComplex *x, int incx,  
const cuComplex *y, int incy)

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

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

where

\[
\begin{align*}
lx &= 1 \text{ if } incx \geq 0, \text{ else } \\
ly &= 1+(1-n)*incx;
\end{align*}
\]

\(ly\) 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 \(x\)
- \(y\) single-precision complex vector with \(n\) elements
- \(incy\) storage spacing between elements of \(y\)

**Output**

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

**Reference:** [http://www.netlib.org/blas/cdotu.f](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()

```c
void
cublasCrot (int n, cuComplex *x, int incx, cuComplex *y, int incy, float sc, cuComplex cs)
```

multiplies a $2 \times 2$ matrix $\begin{bmatrix} sc & cs \\ -cs & sc \end{bmatrix}$ with the $2 \times n$ matrix $\begin{bmatrix} x \end{bmatrix}^T$.

The elements of $x$ are in $x[lx + i \times incx], i = 0 \text{ to } n-1$, where

\[
lx = \begin{cases} 1 & \text{if incx} \geq 0, \\ 1 + (1 - n) \times \text{incx}; & \text{else} \end{cases}
\]

$y$ 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 $x$
- $y$ single-precision complex vector with $n$ elements
- $incy$ storage spacing between elements of $y$
- $sc$ single-precision cosine component of rotation matrix
- $cs$ single-precision complex sine component of rotation matrix

Output

- $x$ rotated vector $x$ (unchanged if $n \leq 0$)
- $y$ rotated vector $y$ (unchanged if $n \leq 0$)


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()

```c
void
cublasCrotg (cuComplex *host_ca, cuComplex cb,
             float *host_sc, float *host_cs)
```

constructs the complex Givens transformation

\[
G = \begin{bmatrix}
    sc & cs \\
    -cs & sc
\end{bmatrix}, \quad sc*sc + cs*cs = 1
\]

which zeros the second entry of the complex 2-vector \([ca \ cb]^T\).

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

\[
|ca, cb| = \text{scale} \sqrt{|ca/\text{scale}|^2 + |cb/\text{scale}|^2}, \quad \text{where}
\]

\[
\text{scale} = |ca| + |cb|.
\]

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**

- `ca` single-precision complex scalar
- `cb` single-precision complex scalar

**Output**

- `ca` single-precision complex \(|ca|*|ca, cb||`  
- `sc` single-precision cosine component of rotation matrix  
- `cs` single-precision complex sine component of rotation matrix

**Reference:** [http://www.netlib.org/blas/crotg.f](http://www.netlib.org/blas/crotg.f)

This function does not set any error status.
Function cublasCscal()

```c
void
cublasCscal (int n, cuComplex alpha, cuComplex *x,
             int incx)
```

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

For \( i = 0 \) to \( n-1 \), it replaces

\[
x[1x+i*incx] \text{ with } \alpha \times x[1x+i*incx],
\]

where

\[
1x = 1 \text{ if } incx >= 0, \text{ else } 1x = 1 + (1-n)*incx.
\]

Input

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

Output

- \( \mathbf{x} \): single-precision complex result (unchanged if \( n \leq 0 \) or \( incx \leq 0 \))

Reference: [http://www.netlib.org/blas/cscal.f](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()

```c
void
cublasCsrot (int n, cuComplex *x, int incx, cuComplex *y,
             int incy, float sc, float ss)
```

multiplies a \( 2 \times 2 \) matrix \[
\begin{bmatrix}
sc & ss \\
-ss & sc
\end{bmatrix}
\]
with the \( 2 \times n \) matrix \[
\begin{bmatrix}
x^T \\
y^T
\end{bmatrix}
\].
The elements of \( x \) are in \( x[lx + i \cdot incx], i = 0 \) to \( n-1 \), where

\[
lx = 1 \text{ if } incx \geq 0, \text{ else } \nlx = 1 + (1 - n) \cdot incx;
\]

\( y \) 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 \( x \)
- \( y \) single-precision complex vector with \( n \) elements
- \( incy \) storage spacing between elements of \( y \)
- \( sc \) single-precision cosine component of rotation matrix
- \( ss \) single-precision sine component of rotation matrix

**Output**

- \( x \) rotated vector \( x \) (unchanged if \( n \leq 0 \))
- \( y \) rotated vector \( y \) (unchanged if \( n \leq 0 \))

Reference: [http://www.netlib.org/blas/csrot.f](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()**

```c
void
cublasCsscal (int n, float alpha, cuComplex *x, int incx)
```

replaces single-precision complex vector \( x \) with single-precision complex \( \alpha \cdot x \). For \( i = 0 \) to \( n-1 \), it replaces

\[
x[lx+i \cdot incx] \text{ with } \alpha \cdot x[lx+i \cdot incx],
\]

where

\[
lx = 1 \text{ if } incx \geq 0, \text{ else } \nlx = 1 + (1 - n) \cdot incx.
\]
Function cublasCswap()

void
cublasCswap (int n, const cuComplex *x, int incx, 
cuComplex *y, int incy)

interchanges the single-precision complex vector \(x\) with the single-precision complex vector \(y\). For \(i = 0\) to \(n-1\), it interchanges
\[x[lx+i*incx] \text{ with } y[ly+i*incy],\]

where
\[lx = 1 \text{ if } incx >= 0, \text{ else } lx = 1+(1-n)*incx;\]

\(ly\) is defined in a similar way using \(incy\).

Input

\(n\) number of elements in input vectors
\(x\) single-precision complex vector with \(n\) elements
\(incy\) storage spacing between elements of \(x\)
\(y\) single-precision complex vector with \(n\) elements
\(incy\) storage spacing between elements of \(y\)
CHAPTER 2

BLAS1 Functions

Output

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>single-precision complex vector y (unchanged from input if n &lt;= 0)</td>
</tr>
<tr>
<td>y</td>
<td>single-precision complex vector x (unchanged from input if n &lt;= 0)</td>
</tr>
</tbody>
</table>

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 `x`; that is, the result is the first `i`, `i = 0` to `n-1`, that maximizes `abs(x[1+i*incx])`. The result reflects 1-based indexing for compatibility with Fortran.

Input

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>number of elements in input vector</td>
</tr>
<tr>
<td>x</td>
<td>single-precision complex vector with n elements</td>
</tr>
<tr>
<td>incx</td>
<td>storage spacing between elements of x</td>
</tr>
</tbody>
</table>

Output

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

```c
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 \(\text{abs}(x[1+i*incx])\). 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 \(x\)

**Output**

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

**Reference:** Analogous to [http://www.netlib.org/blas/icamax.f](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

Function cublasScasum()

```c
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

\[
\text{abs}(\text{real}(x[lx+i*incx])) + \text{abs}(\text{imag}(x[lx+i*incx])),
\]

where

\[
lx = 1 \text{ if } incx <= 0, \text{ else } \]
\[
lx = 1+(1-n)*incx.
\]
CHAPTER 2

BLAS1 Functions

Input

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

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 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 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()`.

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CUBLAS_STATUS_NOT_INITIALIZED</code></td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ALLOC_FAILED</code></td>
<td>if function could not allocate reduction buffer</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_EXECUTION_FAILED</code></td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
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 56
- “Function cublasIdamin()” on page 56
- “Function cublasDasum()” on page 57
- “Function cublasDaxpy()” on page 58
- “Function cublasDcopy()” on page 59
- “Function cublasDdot()” on page 60
- “Function cublasDnrm2()” on page 61
- “Function cublasDrot()” on page 62
- “Function cublasDrotg()” on page 63
- “Function cublasDrotm()” on page 64
- “Function cublasDrotmg()” on page 65
- “Function cublasDscal()” on page 66
- “Function cublasDswap()” on page 67
Function cublasIdamax()

```c
int

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

finds the smallest index of the maximum magnitude element of double-precision vector x; that is, the result is the first i, i = 0 to n-1, that maximizes \( \text{abs}(x[1+i*\text{incx}]). \) 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
- \( \text{incx} \) storage spacing between elements of \( x \)

Output

returns the smallest index (returns zero if \( n <= 0 \) or \( \text{incx} <= 0 \))

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

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

Error Status

- \( \text{CUBLAS_STATUS_NOT_INITIALIZED} \) if CUBLAS library was not initialized
- \( \text{CUBLAS_STATUS_ALLOC_FAILED} \) if function could not allocate reduction buffer
- \( \text{CUBLAS_STATUS_ARCH_MISMATCH} \) if function invoked on device that does not support double precision
- \( \text{CUBLAS_STATUS_EXECUTION_FAILED} \) if function failed to launch on GPU

Function cublasIdamin()

```c
int

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

finds the smallest index of the minimum magnitude element of double-precision vector x; that is, the result is the first i, i = 0 to n-1, that minimizes \( \text{abs}(x[1+i*\text{incx}]). \) The result reflects 1-based indexing for compatibility with Fortran.
CHAPTER 2  BLAS1 Functions

Analogous to [http://www.netlib.org/blas/idamax.f](http://www.netlib.org/blas/idamax.f)

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

**Function cublasDasum()**

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

computes the sum of the absolute values of the elements of double-precision vector `x`; that is, the result is the sum from `i = 0` to `n-1` of 

```latex
abs(x[1+i*incx])
```

**Input**

- `n` number of elements in input vector
- `x` double-precision vector with `n` elements
- `incx` storage spacing between elements of `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](http://www.netlib.org/blas/dasum.f)
Error status for this function can be retrieved via `cublasGetError()`.  

**Error Status**  
<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CUBLAS_STATUS_NOT_INITIALIZED</code></td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ALLOC_FAILED</code></td>
<td>if function could not allocate reduction buffer</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ARCH_MISMATCH</code></td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_EXECUTION_FAILED</code></td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

**Function cublasDaxpy()**

```c
void cublasDaxpy (int n, double alpha, const double *x,  
                 int incx, double *y, int incy);
```

multiplies double-precision vector `x` by double-precision scalar `alpha` and adds the result to double-precision vector `y`; that is, it overwrites double-precision `y` with double-precision `alpha * x + y`. For `i = 0` to `n-1`, it replaces

\[
y[ly + i * incy] \text{ with } alpha \times x[1x + i * incx] + y[ly + i * incy],
\]

where

\[
1x = 0 \text{ if incx } \geq 0, \text{ else } 1x = 1 + (1 - n) \times incx;
\]

`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`
- `y`: double-precision vector with `n` elements
- `incy`: storage spacing between elements of `y`

**Output**

- `y`: double-precision result (unchanged if `n <= 0`)
Reference: [http://www.netlib.org/blas/daxpy.f](http://www.netlib.org/blas/daxpy.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 cublasDcopy()**

```c
void

// void cublasDcopy (int n, const double *x, int incx, double *y, int incy);

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

\[
x[1x + i \times incx] \rightarrow y[1y + i \times incy],
\]

where

\[
1x = 1 \text{ if incx } \geq 0, \text{ else } \quad 1x = 1 + (1-n) \times incx;
\]

\(1y\) 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 \(x\)
- \(y\) double-precision vector with \(n\) elements
- \(incy\) storage spacing between elements of \(y\)

**Output**

- \(y\) contains double-precision vector \(x\)

Reference: [http://www.netlib.org/blas/dcopy.f](http://www.netlib.org/blas/dcopy.f)
Error status for this function can be retrieved via `cublasGetError()`.

**Error Status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CUBLAS_STATUS_NOT_INITIALIZED</code></td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ARCH_MISMATCH</code></td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_EXECUTION_FAILED</code></td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

**Function cublasDdot()**

```c
double
cublasDdot (int n, const double *x, int incx, const double *y, int incy)
```

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

$$
lx = 1 \text{ if } incx \geq 0, \text{ else } lx = 1 + (1-n) \cdot incx;
$$

$ly$ 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 $x$
- $y$ double-precision vector with $n$ elements
- $incy$ storage spacing between elements of $y$

**Output**

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

Reference: [http://www.netlib.org/blas/ddot.f](http://www.netlib.org/blas/ddot.f)
CHAPTER 2  BLAS1 Functions

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

## Error Status

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ALLOC_FAILED</td>
<td>if function could not allocate reduction buffer</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

### Function `cublasDnrm2()`

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

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

#### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>number of elements in input vector</td>
</tr>
<tr>
<td>$x$</td>
<td>double-precision vector with $n$ elements</td>
</tr>
<tr>
<td>$\text{incx}$</td>
<td>storage spacing between elements of $x$</td>
</tr>
</tbody>
</table>

#### Output

returns the Euclidian norm  

(returns zero if $n \leq 0$, $\text{incx} \leq 0$, or if an error occurred)

Reference: [http://www.netlib.org/blas/dnrm2.f](http://www.netlib.org/blas/dnrm2.f)


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

## Error Status

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ALLOC_FAILED</td>
<td>if function could not allocate reduction buffer</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
Function cublasDrot()

```c
void
cublasDrot (int n, double *x, int incx, double *y, int incy, double dc, double ds)
```

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

The elements of $x$ are in $x[lx + i \times \text{incx}], i = 0 \text{ to } n-1$, where

\begin{align*}
lx &= 1 \text{ if incx} \geq 0, \text{ else} \\
lx &= 1 + (1-n) \times \text{incx};
\end{align*}

$y$ is treated similarly using $ly$ and $incy$.

**Input**

- **n**: number of elements in input vectors
- **x**: double-precision vector with $n$ elements
- **incx**: storage spacing between elements of $x$
- **y**: double-precision vector with $n$ elements
- **incy**: storage spacing between elements of $y$
- **dc**: element of rotation matrix
- **ds**: element of rotation matrix

**Output**

- **x**: rotated vector $x$ (unchanged if $n \leq 0$)
- **y**: rotated vector $y$ (unchanged if $n \leq 0$)

Reference: [http://www.netlib.org/blas/drot.f](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
CHAPTER 2

BLAS1 Functions

Function cublasDrotg()

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

constructs the Givens transformation

\[
G = \begin{bmatrix} dc & ds \\
    -ds & dc \end{bmatrix}, \quad dc^2 + ds^2 = 1
\]

which zeros the second entry of the 2-vector \([da\ db]^T\).

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:

\[
\begin{align*}
\text{if } z &= 1 & \text{set } dc &= 0.0 \text{ and } ds = 1.0. \\
\text{if } \lvert z \rvert &< 1 & \text{set } dc &= \sqrt{1-z^2} \text{ and } ds = z. \\
\text{if } \lvert z \rvert &> 1 & \text{set } dc &= 1/z \text{ and } ds = \sqrt{1-dc^2}.
\end{align*}
\]

The function `cublasDrot(n, x, incx, y, incy, dc, ds)` 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

- \(da\) double-precision scalar
- \(db\) double-precision scalar

Output

- \(da\) double-precision scalar overwrites \(da\)
- \(db\) double-precision scalar overwrites \(db\) by \(z\)
- \(dc\) double-precision result
- \(ds\) double-precision result

Reference: [http://www.netlib.org/blas/drotg.f](http://www.netlib.org/blas/drotg.f)

This function does not set any error status.
Function cublasDrotm()

```c
void
cublasDrotm (int n, double *x, int incx, double *y, 
              int incy, const double *dparam)
```

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 \( x[lx+i \times incx], i = 0 \) to \( n-1 \), where

\[
lx = 1 \text{ if } incx \geq 0, \text{ else } \]
\[
lx = 1 + (1 - n) \times incx;
\]

\( y \) is treated similarly using \( ly \) and \( incy \).

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

- \( dflag = -1.0 \)
  \[
h = \begin{bmatrix} dh00 & dh01 \\
dh10 & dh11 \end{bmatrix}
\]

- \( dflag = 0.0 \)
  \[
h = \begin{bmatrix} 1.0 & dh01 \\
dh10 & 1.0 \end{bmatrix}
\]

- \( dflag = 1.0 \)
  \[
h = \begin{bmatrix} dh00 & 1.0 \\
-1.0 & dh11 \end{bmatrix}
\]

- \( dflag = -2.0 \)
  \[
h = \begin{bmatrix} 1.0 & 0.0 \\
0.0 & 1.0 \end{bmatrix}
\]

**Input**

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

**Output**

- \( x \) rotated vector \( x \) (unchanged if \( n \leq 0 \))
- \( y \) rotated vector \( y \) (unchanged if \( n \leq 0 \))
Function cublasDrotmg()

```c
void
cublasDrotmg (double *host_dd1, double *host_dd2,
               double *host_dx1, const double *host_dy1,
               double *host_dparam)
```

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:

- $dflag = -1.0$
  - $h = \begin{bmatrix} dh00 & dh01 \\ dh10 & dh11 \end{bmatrix}$

- $dflag = 0.0$
  - $h = \begin{bmatrix} 1.0 & dh01 \\ dh10 & 1.0 \end{bmatrix}$

- $dflag = 1.0$
  - $h = \begin{bmatrix} dh00 & 1.0 \\ -1.0 & dh11 \end{bmatrix}$

- $dflag = -2.0$
  - $h = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$

$dparam[1]$ 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
Function cublasDscal()

```c
void
cublasDscal (int n, double alpha, double *x, int incx)
```

replaces double-precision vector \( x \) with double-precision \( \alpha \times x \). For \( i = 0 \) to \( n-1 \), it replaces

\[
x[lx+i*incx] \text{ with } \alpha \times x[lx+i*incx],
\]

where

\[
lx = 1 \text{ if } incx >= 0, \text{ else } \]
\[
lx = 1 + (1-n) * incx.
\]

Input

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

Output

- \( x \) double-precision result (unchanged if \( n <= 0 \) or \( incx <= 0 \))

Reference: [http://www.netlib.org/blas/dscal.f](http://www.netlib.org/blas/dscal.f)
Error status for this function can be retrieved via `cublasGetError()`. 

### Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

### Function `cublasDswap()`

```c
void cublasDswap (int n, double *x, int incx, double *y, int incy)
```

interchanges double-precision vector \(x\) with double-precision vector \(y\).

For \(i = 0\) to \(n-1\), it interchanges

\[
x[1x + i \cdot \text{incx}] \text{ with } y[1y + i \cdot \text{incy}],
\]

where

\[
x = 1 \text{ if } \text{incx} \geq 0, \text{ else } \text{incx};
\]

\(1y\) is defined in a similar manner using \(\text{incy}\).

### Input

- \(n\): number of elements in input vectors
- \(x\): double-precision vector with \(n\) elements
- \(\text{incx}\): storage spacing between elements of \(x\)
- \(y\): double-precision vector with \(n\) elements
- \(\text{incy}\): storage spacing between elements of \(y\)

### Output

- \(x\): double-precision vector \(y\) (unchanged from input if \(n \leq 0\))
- \(y\): double-precision vector \(x\) (unchanged from input if \(n \leq 0\))

### Reference

[http://www.netlib.org/blas/dswap.f](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 70
- “Function cublasDznrm2()” on page 71
- “Function cublasIzamax()” on page 71
- “Function cublasIzamin()” on page 72
- “Function cublasZaxpy()” on page 73
- “Function cublasZcopy()” on page 74
- “Function cublasZdotc()” on page 75
- “Function cublasZdotu()” on page 76
- “Function cublasZdrot()” on page 77
- “Function cublasZdscal()” on page 78
- “Function cublasZrot()” on page 79
- “Function cublasZrotg()” on page 80
- “Function cublasZscal()” on page 80
- “Function cublasZswap()” on page 81
Function `cublasDzasum()`

```c
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](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 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](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 x; that is, the result is the first i, i = 0 to n-1, that maximizes

\[
\text{abs(real}(x[i + i \times \text{incx}]) + \text{abs(imag}(x[i + i \times \text{incx}]))
\]

Input

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

```c
int
```
cublasIzamin (int n, const cuDoubleComplex *x, int incx)
```
finds the smallest index of the minimum magnitude element of
double-precision complex vector x; that is, the result is the first i, i = 0
to n−1, that minimizes
```
abs(real(x[i + i * incx])) + abs(imag(x[i + i * 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 smallest index (returns zero if n <= 0 or incx <= 0)

Reference: analogous to “Function cublasIzamax()” on page 71.

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 cublasZaxpy()

void
cublasZaxpy (int n, cuDoubleComplex alpha, 
    const cuDoubleComplex *x, int incx, 
    cuDoubleComplex *y, int incy)

multiplies double-precision complex vector \( x \) by double-precision complex scalar \( \alpha \) and adds the result to double-precision complex vector \( y \); that is, it overwrites double-precision complex \( y \) with double-precision complex \( \alpha \cdot x + y \).

For \( i = 0 \) to \( n-1 \), it replaces

\[
y[\text{ly} + i \cdot \text{incy}] \text{ with } \alpha \cdot x[\text{lx} + i \cdot \text{incx}] + y[\text{ly} + i \cdot \text{incy}],
\]

where

\[
\text{lx} = 0 \text{ if incx} \geq 0, \text{ else } \text{lx} = 1 + (1-n) \cdot \text{incx};
\]

\( \text{ly} \) is defined in a similar way using \( \text{incy} \).

Input

- \( n \) number of elements in input vectors
- \( \alpha \) double-precision complex scalar multiplier
- \( x \) double-precision complex vector with \( n \) elements
- \( \text{incx} \) storage spacing between elements of \( x \)
- \( y \) double-precision complex vector with \( n \) elements
- \( \text{incy} \) storage spacing between elements of \( y \)

Output

- \( y \) double-precision complex result (unchanged if \( n \leq 0 \))

Reference: [http://www.netlib.org/blas/zaxpy.f](http://www.netlib.org/blas/zaxpy.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 cublasZcopy()

```c
void
cublasZcopy (int n, const cuDoubleComplex *x, int incx,
cuDoubleComplex *y, int incy)
```
copies the double-precision complex vector \( x \) to the double-precision complex vector \( y \). For \( i = 0 \) to \( n-1 \), it copies

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

where

\[
lx = 1 \text{ if } incx >= 0, \text{ else } 1 + (1-n) * incx;
\]

\( ly \) 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 \( x \)
- \( y \) double-precision complex vector with \( n \) elements
- \( incy \) storage spacing between elements of \( y \)

**Output**

- \( y \) contains double-precision complex vector \( x \)

**Reference:** [http://www.netlib.org/blas/zcopy.f](http://www.netlib.org/blas/zcopy.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 cublasZdotc()**

```c
void cublasZdotc (int n, const cuDoubleComplex *x, int incx,
                 const cuDoubleComplex *y, int incy);
```

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

\[
\sum_{i=0}^{n-1} x[lx + i \times incx] \times y[ly + i \times incy],
\]

where

\[
lx = 1 \text{ if } incx \geq 0, \text{ else } lx = 1 + (1-n) \times incx;
\]

\( ly \) 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 \( x \)
- \( y \): double-precision complex vector with \( n \) elements
- \( incy \): storage spacing between elements of \( y \)

**Output**

returns double-precision complex dot product (zero if \( n \leq 0 \))

**Reference:** [http://www.netlib.org/blas/zdotc.f](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

cublasZdotu (int n, const cuDoubleComplex *x, int incx,
const cuDoubleComplex *y, int incy)

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

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

where

\[
lx = 1 \text{ if } incx \geq 0, \text{ else } \\
1x = 1 + (1-n) \cdot incx;
\]

\( ly \) 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 \( x \)
- \( y \) double-precision complex vector with \( n \) elements
- \( incy \) storage spacing between elements of \( y \)

Output

returns double-precision complex dot product (returns zero if \( n \leq 0 \))

Reference: [http://www.netlib.org/blas/zdotu.f](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
Function cublasZdrot()

```c
void
cublasZdrot (int n, cuDoubleComplex *x, int incx,
cuDoubleComplex *y, int incy,
double c, double s)
```

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 \( x[lx+i*\text{incx}], i=0 \) to \( n-1 \), where

\[
lx = 1 \text{ if } \text{incx} >= 0, \text{ else } \\
lx = 1+(1-n)*\text{incx};
\]

\( y \) is treated similarly using \( ly \) and \( \text{incy} \).

**Input**

- \( n \): number of elements in input vectors
- \( x \): double-precision complex vector with \( n \) elements
- \( \text{incx} \): storage spacing between elements of \( x \)
- \( y \): double-precision complex vector with \( n \) elements
- \( \text{incy} \): storage spacing between elements of \( y \)
- \( c \): double-precision cosine component of rotation matrix
- \( s \): double-precision sine component of rotation matrix

**Output**

- \( x \): rotated vector \( x \) (unchanged if \( n <= 0 \))
- \( y \): rotated vector \( y \) (unchanged if \( n <= 0 \))

**Reference:** [http://www.netlib.org/blas/zdrot.f](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()

```c
void cublasZdscal (int n, double alpha, cuDoubleComplex *x, int incx)
```

replaces double-precision complex vector \( x \) with double-precision complex \( \alpha \times x \).

For \( i = 0 \) to \( n-1 \), it replaces

\[ x[lx+i \times incx] \text{ with } \alpha \times x[lx+i \times incx], \]

where

\[
x_{l} = 1 \text{ if } incx \geq 0, \text{ else } \]
\[
x_{l} = 1+(1-n) \times incx.
\]

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 \( x \)

Output

- \( x \): double-precision complex result (unchanged if \( n \leq 0 \) or \( incx \leq 0 \))

Reference: [http://www.netlib.org/blas/zdscal.f](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
Function cublasZrot()

```c
void cublasZrot (int n, cuDoubleComplex *x, int incx,
    cuDoubleComplex *y, int incy, double sc,
    cuDoubleComplex cs)
```

multiplies a $2 \times 2$ matrix \[
\begin{bmatrix}
    sc & cs \\
    -cs & sc
\end{bmatrix}
\] with the $2 \times n$ matrix \[
\begin{bmatrix}
    x^T \\
    y^T
\end{bmatrix}
\].

The elements of $x$ are in $x[lx + i \times \text{incx}], i = 0$ to $n-1$, where

\[
lx = 1 \text{ if incx} \geq 0, \text{ else } \\
lx = 1 + (1-n) \times \text{incx};
\]

$y$ is treated similarly using $ly$ and $incy$.

**Input**

- $n$ number of elements in input vectors
- $x$ double-precision complex vector with $n$ elements
- $\text{incx}$ storage spacing between elements of $x$
- $y$ double-precision complex vector with $n$ elements
- $\text{incy}$ storage spacing between elements of $y$
- $sc$ double-precision cosine component of rotation matrix
- $cs$ double-precision complex sine component of rotation matrix

**Output**

- $x$ rotated double-precision complex vector $x$ (unchanged if $n \leq 0$)
- $y$ rotated double-precision complex vector $y$ (unchanged if $n \leq 0$)

**Reference:** [http://netlib.org/lapack/explore-html/zrot.f.html](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()

```c
void
cublasZrotg (cuDoubleComplex *host_ca,
             cuDoubleComplex *host_cb, double *host_sc,
             double *host_cs)
```

constructs the complex Givens transformation

\[
G = \begin{bmatrix}
    \text{sc} & \text{cs} \\
    -\text{cs} & \text{sc}
\end{bmatrix}, \quad \text{sc}^2 + |\text{cs}|^2 = 1
\]

which zeros the second entry of the complex 2-vector \([\text{ca} \ \text{cb}]^T\).

The quantity \(\text{ca}/|\text{ca}|\|\text{ca}, \ \text{cb}||\) overwrites \(\text{ca}\) in storage. The function `cublasCrot` \((n, x, \text{incx}, y, \text{incy}, \text{sc}, \text{cs})\) 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 \(\text{ca}/|\text{ca}|\|\text{ca}, \ \text{cb}||\)
- `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](http://www.netlib.org/blas/zrotg.f)

This function does not set any error status.

Function cublasZscal()

```c
void
cublasZscal (int n, cuDoubleComplex alpha,
             cuDoubleComplex *x, int incx)
```

replaces double-precision complex vector \(x\) with double-precision complex \(\text{alpha} \times x\).
For \( i = 0 \) to \( n-1 \), it replaces
\[
x[lx+i*incx] \textit{ with } \alpha x[lx+i*incx],
\]
where
\[
lx = 1 \text{ if } incx \geq 0, \text{ else } \\
lx = 1+(1-n)*incx.
\]

**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 \( x \)

**Output**
- \( x \): double-precision complex result (unchanged if \( n \leq 0 \) or \( incx \leq 0 \))

**Reference:** [http://www.netlib.org/blas/zscal.f](http://www.netlib.org/blas/zscal.f)

Error status for this function can be retrieved via \texttt{cublasGetError()}

**Error Status**
- \texttt{CUBLAS_STATUS_NOT_INITIALIZED} if CUBLAS library was not initialized
- \texttt{CUBLAS_STATUS_EXECUTION_FAILED} if function failed to launch on GPU
- \texttt{CUBLAS_STATUS_ARCH_MISMATCH} if function invoked on device that does not support double precision

**Function\ cublasZswap()**

```c
void
cublasZswap (int n, cuDoubleComplex *x, int incx,
                cuDoubleComplex *y, int incy)
```

interchanges double-precision complex vector \( x \) with double-precision complex vector \( y \). For \( i = 0 \) to \( n-1 \), it interchanges
\[
x[lx+i*incx] \textit{ with } y[ly+i*incy],
\]
where
\[
lx = 1 \text{ if } incx \geq 0, \text{ else } \\
lx = 1+(1-n)*incx;
\]
**ly is defined in a similar manner 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
- y: double-precision complex vector with n elements
- incy: storage spacing between elements of y

**Output**

- x: double-precision complex vector y (unchanged from input if n <= 0)
- y: double-precision complex vector x (unchanged from input if n <= 0)

**Reference:** [http://www.netlib.org/blas/zswap.f](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 84
- “Single-Precision Complex BLAS2 Functions” on page 107
Single-Precision BLAS2 Functions

The single-precision BLAS2 functions are as follows:

- “Function cublasSgbmv()” on page 85
- “Function cublasSgemv()” on page 86
- “Function cublasSger()” on page 87
- “Function cublasSsbmv()” on page 88
- “Function cublasSspmv()” on page 90
- “Function cublasSspr()” on page 91
- “Function cublasSspr2()” on page 92
- “Function cublasSsymv()” on page 94
- “Function cublasSsyr()” on page 95
- “Function cublasSsyr2()” on page 96
- “Function cublasStbmv()” on page 98
- “Function cublasStbsv()” on page 99
- “Function cublasStpmv()” on page 101
- “Function cublasStpsv()” on page 102
- “Function cublasStrmv()” on page 104
- “Function cublasStrsv()” on page 105
Function cublasSgbmv()

```c
void
cublasSgbmv (char trans, int m, int n, int kl, int ku,
float alpha, const float *A, int lda,
const float *x, int incx, float beta,
float *y, int incy);
```

performs one of the matrix-vector operations

\[ y = \alpha \cdot \text{op}(A) \cdot x + \beta \cdot y, \]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

\( \alpha \) 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**

- **trans** specifies \( \text{op}(A) \). If \( \text{trans} = 'N' \) or 'n', \( \text{op}(A) = A \).
  - If \( \text{trans} = 'T', 't', 'C', \) or 'c', \( \text{op}(A) = A^T \).
- **m** the number of rows of matrix \( A \); \( m \) must be at least zero.
- **n** the number of columns of matrix \( A \); \( n \) must be at least zero.
- **kl** the number of subdiagonals of matrix \( A \); \( kl \) must be at least zero.
- **ku** the number of superdiagonals of matrix \( A \); \( ku \) must be at least zero.
- **alpha** single-precision scalar multiplier applied to \( \text{op}(A) \).
- **A** single-precision array of dimensions \((\text{lda}, n)\). The leading \((kl + ku + 1) \times n\) 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 \( ku \times ku \) triangle) are not referenced.
- **lda** leading dimension of \( A \); \( lda \) must be at least \( kl + ku + 1 \).
- **x** single-precision array of length at least \((1 + (n - 1) \times \text{abs}(\text{incx}))\)
  - when \( \text{trans} = 'N' \) or 'n', and at least \((1 + (m - 1) \times \text{abs}(\text{incx}))\)
  - 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.
Function cublasSgemv()

```c
void
cublasSgemv (char trans, int m, int n, float alpha,
const float *A, int lda, const float *x,
int incx, float beta, float *y, int incy)
```

performs one of the matrix-vector operations

\[
y = \alpha \text{op}(A) \times \times + \beta y,
\]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

\( \alpha \) 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 \( \text{lda} \) is the leading dimension of the two-dimensional array in which \( A \) is stored.

Input

- **trans**: specifies \( \text{op}(A) \). If \( \text{trans} == \text{'N'} \) or \( \text{'n'} \), \( \text{op}(A) = A \).
  - If \( \text{trans} == \text{'T'}, \text{'t'}, \text{'C'}, \text{or 'c'}, \text{op}(A) = A^T. \)
- **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.
CHAPTER 3

Single-Precision BLAS2 Functions

Reference:
http://www.netlib.org/blas/sgemv.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_EXECUTION_FAILED** if function failed to launch on GPU

### Function `cublasSger()`

```c
void cublasSger (int m, int n, float alpha, const float *x,
                int incx, const float *y, int incy, float *A,
                int lda)
```

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 \( \text{lda} \) 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 \( x \) \(* y^T \).
- \( x \): single-precision array of length at least \((1+(m-1) \times \text{abs}(\text{incx}))\).
- \( \text{incx} \): the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- \( y \): single-precision array of length at least \((1+(n-1) \times \text{abs}(\text{incy}))\).
- \( \text{incy} \): the storage spacing between elements of \( y \); \( \text{incy} \) must not be zero.
- \( A \): single-precision array of dimensions \((\text{lda}, n)\).
- \( \text{lda} \): leading dimension of two-dimensional array used to store matrix \( A \).

**Output**
- \( A \): updated according to \( A = \alpha \cdot x \cdot y^T + A \).

**Reference:** [http://www.netlib.org/blas/sger.f](http://www.netlib.org/blas/sger.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 \( m < 0, n < 0, \text{incx} == 0 \), or \( \text{incy} == 0 \)
- `CUBLAS_STATUS_EXECUTION_FAILED`: if function failed to launch on GPU

**Function cublasSsbmv()**

```c
void
cublasSsbmv (char uplo, int n, int k, float alpha,
        const float *A, int lda, const float *x,
        int incx, float beta, float *y, int incy);
```

performs the matrix-vector operation

\[
y = \alpha \cdot A \cdot x + \beta \cdot y,
\]
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**

- **\( \text{uplo} \)** specifies whether the upper or lower triangular part of the symmetric band matrix \( A \) is being supplied. If \( \text{uplo} == 'U' \) or \( 'u' \), the upper triangular part is being supplied. If \( \text{uplo} == 'L' \) 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; \( k \) must be at least zero.
- **\( \alpha \)** single-precision scalar multiplier applied to \( A \cdot x \).
- **\( A \)** single-precision array of dimensions \((\text{lda}, n)\). When \( \text{uplo} == 'U' \) or \( 'u' \), the leading \((k+1) \times n\) 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 \( k \times k \) triangle of the array \( A \) is not referenced. When \( \text{uplo} == 'L' \) 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 \( k \times k \) triangle of the array \( A \) is not referenced.
- **\( \text{lda} \)** leading dimension of \( A \); \( \text{lda} \) must be at least \( k+1 \).
- **\( x \)** single-precision array of length at least \((1+(n-1) \times \text{abs(incx)})\).
- **\( \text{incx} \)** storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- **\( \beta \)** single-precision scalar multiplier applied to vector \( y \). If \( \beta \) is zero, \( y \) is not read.
- **\( Y \)** single-precision array of length at least \((1+(n-1) \times \text{abs(incy)})\). If \( \beta \) is zero, \( y \) is not read.
- **\( \text{incy} \)** storage spacing between elements of \( y \); \( \text{incy} \) must not be zero.

**Output**

- **\( y \)** updated according to \( y = \alpha \cdot A \cdot x + \beta \cdot y \).
Reference: http://www.netlib.org/blas/ssbmv.f

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

**Error Status**

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if k &lt; 0, n &lt; 0, incx == 0, or incy == 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

**Function cublasSspmv()**

```c
void
cublasSspmv (char uplo, int n, float alpha,
             const float *AP, const float *x, int incx,
             float beta, float *y, int incy)
```

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 \( AP \).
- **n** the number of rows and columns of matrix \( A \); \( n \) must be at least zero.
- **alpha** single-precision scalar multiplier applied to \( A \ast x \).
- **AP** single-precision array with at least \((n \ast (n + 1))/2\) 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 \( i \leq j, A[i, j] \) is stored in \( AP[i + (j \ast (j + 1)/2)] \). 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 \( i \geq j, A[i, j] \) is stored in \( AP[i + ((2 \ast n - j + 1) \ast j)/2] \).
- **x** single-precision array of length at least \((1 + (n - 1) \ast \text{abs}(\text{incx}))\).
- **incx** storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
CHAPTER 3 Single-Precision BLAS2 Functions

Reference: [http://www.netlib.org/blas/sspmv.f](http://www.netlib.org/blas/sspmv.f)

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

**Function cublasSspr()**

```c
void

cublasSspr (char uplo, int n, float alpha,
            const float *x, int incx, float *AP)
```

performs the symmetric rank 1 operation

\[
A = \alpha x^T A + \alpha x^T \alpha
\]

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**

- **uplo** specifies whether the matrix data is stored in the upper or the lower triangular part of array \(A\). If \(\text{uplo} == \text{'U'}\) or \(\text{'u'}\), the upper triangular part of \(A\) is supplied in \(A\). If \(\text{uplo} == \text{'L'}\) or \(\text{'l'}\), the lower triangular part of \(A\) is supplied in \(A\).
- **n** is the number of rows and columns of matrix \(A\); \(n\) must be at least zero.
- **alpha** is a single-precision scalar multiplier applied to \(x^T \alpha\).
- **x** is a single-precision array of length at least \((1 + (n - 1) \times \text{abs(incx)})\).
Input (continued)

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

Ap       single-precision array with at least \((n \times (n + 1))/2\) 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
          \(i \leq j\), \(A[i,j]\) is stored in \(AP[i + (j \times (j + 1))/2]\). 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
          \(i \geq j\), \(A[i,j]\) is stored in \(AP[i + (2 \times n - j + 1) \times j]/2]\).

Output

\(A\) updated according to \(A = \alpha \times x \times x^T + A\).

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

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

Error Status

\begin{tabular}{|l|}
\hline
\texttt{CUBLAS_STATUS_NOT_INITIALIZED} & if CUBLAS library was not initialized \\
\texttt{CUBLAS_STATUS_INVALID_VALUE} & if \(n < 0\) or \(\text{incx} == 0\) \\
\texttt{CUBLAS_STATUS_EXECUTION_FAILED} & if function failed to launch on GPU \\
\hline
\end{tabular}

Function \texttt{cublasSspr2()} 

\begin{verbatim}
void

cublasSspr2 (char uplo, int n, float alpha,
             const float *x, int incx, const float *y,
             int incy, float *AP)

performs the symmetric rank 2 operation

\[ A = \alpha \times x \times y^T + \alpha \times y \times x^T + A, \]

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.
\end{verbatim}
CHAPTER 3 Single-Precision BLAS2 Functions

Reference: [http://www.netlib.org/blas/sspr2.f](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()

```c
void
cublasSsymv (char uplo, int n, float alpha,
             const float *A, int lda, const float *x,
             int incx, float beta, float *y, int incy)
```

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**

- **uplo** specifies whether the upper or lower triangular part of the array \( A \) is referenced. If \( \text{uplo} == \text{'U'} \) or \( \text{'u'} \), 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 \( \text{uplo} == \text{'L'} \) or \( \text{'l'} \), 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 \( A \times x \).
- **A** single-precision array of dimensions \((\text{lda}, n)\). If \( \text{uplo} == \text{'U'} \) or \( \text{'u'} \), 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 \( \text{uplo} == \text{'L'} \) or \( \text{'l'} \), 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 \( A \); \( \text{lda} \) must be at least \( \max(1, n) \).
- **x** single-precision array of length at least \( (1 + (n - 1) \times \text{abs(incx)}) \).
- **incx** storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- **beta** single-precision scalar multiplier applied to vector \( y \).
CHAPTER 3  Single-Precision BLAS2 Functions

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

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

Error Status

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( n &lt; 0 ), ( \text{incx} == 0 ), or ( \text{incy} == 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function cublasSsyr()

```c
void
cublasSsyr (char uplo, int n, float alpha,
            const float *x, int incx, float *A, int lda)
```

performs the symmetric rank 1 operation

\[
A = \alpha x^T A + \alpha x^T x,
\]

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 \( \text{lda} \) is the leading dimension of the two-dimensional array containing \( A \).

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>specifies whether the matrix data is stored in the upper or the lower triangular part of array ( A ). If ( \text{uplo} == 'U' ) or ( 'u' ), only the upper triangular part of ( A ) is referenced. If ( \text{uplo} == 'L' ) or ( 'l' ), only the lower triangular part of ( A ) is referenced.</td>
</tr>
<tr>
<td>n</td>
<td>the number of rows and columns of matrix ( A ); ( n ) must be at least zero.</td>
</tr>
<tr>
<td>alpha</td>
<td>single-precision scalar multiplier applied to ( x^T x ).</td>
</tr>
<tr>
<td>x</td>
<td>single-precision array of length at least ( 1 + (n-1) \times \text{abs}(\text{incx}) ).</td>
</tr>
</tbody>
</table>
CUDA CUBLAS Library

Input (continued)

<table>
<thead>
<tr>
<th>incx</th>
<th>the storage spacing between elements of x; incx must not be zero.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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.</td>
</tr>
<tr>
<td>lda</td>
<td>leading dimension of the two-dimensional array containing A; lda must be at least max(1, n).</td>
</tr>
</tbody>
</table>

Output

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

Reference: [http://www.netlib.org/blas/ssyr.f](http://www.netlib.org/blas/ssyr.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 `cublasSsyr2()`

```c
void

void cublasSsyr2 (char uplo, int n, float alpha,  
const float *x, int incx, const float *y,  
int incy, float *A, int lda)

```

performs the symmetric rank 2 operation

\[
A = \alpha x x^T + \alpha y y^T + A,
\]

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.
### Input

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

### Output

\( A \) updated according to \( A = \alpha \ast x \ast y^T + \alpha \ast y \ast x^T + A \).

### Reference

http://www.netlib.org/blas/ssyr2.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 \( n < 0, \text{incx} == 0 \), or \( \text{incy} == 0 \)
- **CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU
Function cublasStbmv()

```c
void
cublasStbmv (char uplo, char trans, char diag, int n,
              int k, const float *A, int lda, float *x,
              int incx)
```

performs one of the matrix-vector operations

\[
x = \text{op}(A) \times x,
\]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \).

\( x \) 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 \( \text{op}(A) \). If `trans` == `'N'` or `'n'`, \( \text{op}(A) = A \). If `trans` == `'T'`, `'t'`, `'C'`, or `'c'`, \( \text{op}(A) = A^T \).
- `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.
- `A` single-precision array of dimension \((\text{lda}, n)\). If `uplo` == `'U'` or `'u'`, the leading \((k+1)\times n\) 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 \( k \times k \) 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 subdiagonal starting at position 1 in row 2, and so on. The bottom right \( k \times k \) triangle of the array is not referenced.
CHAPTER 3 Single-Precision BLAS2 Functions

Input (continued)

\[ \text{lda} \]

is the leading dimension of \( A \); \( \text{lda} \) must be at least \( k+1 \).

\[ x \]

single-precision array of length at least \( 1+(n-1) \times \text{abs}(\text{incx}) \).

On entry, \( x \) contains the source vector. On exit, \( x \) is overwritten with the result vector.

\[ \text{incx} \]

specifies the storage spacing for elements of \( x \); \( \text{incx} \) must not be zero.

Output

\[ x \]

updated according to \( x = \text{op}(A) \times x \).

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

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( n &lt; 0, k &lt; 0, ) or ( \text{incx} == 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ALLOC_FAILED</td>
<td>if function cannot allocate enough internal scratch vector memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function \text{cublasStbsv()}

\[
\text{void} \quad \text{cublasStbsv}(\text{char uplo}, \text{char trans}, \text{char diag}, \text{int n}, \\
\text{int k}, \text{const float *}A, \text{int lda}, \text{float *}X, \\
\text{int incx})
\]

solves one of the systems of equations

\[
\text{op}(A) \times x = b,
\]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

\( b \) 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

\textbf{uplo} specifies whether the matrix is an upper or lower triangular band matrix: If \texttt{uplo} == 'U' or 'u', \(A\) is an upper triangular band matrix. If \texttt{uplo} == 'L' or 'l', \(A\) is a lower triangular band matrix.

\textbf{trans} specifies \(\text{op}(A)\). If \texttt{trans} == 'N' or 'n', \(\text{op}(A) = A\).

If \texttt{trans} == 'T', 't', 'C', or 'c', \(\text{op}(A) = A^T\).

\textbf{diag} specifies whether \(A\) is unit triangular. If \texttt{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 \texttt{diag} == 'N' or 'n', \(A\) is not assumed to be unit triangular.

\textbf{n} the number of rows and columns of matrix \(A\); \(n\) must be at least zero.

\textbf{k} specifies the number of superdiagonals or subdiagonals.

If \texttt{uplo} == 'U' or 'u', \(k\) specifies the number of superdiagonals. If \texttt{uplo} == 'L' or 'l', \(k\) specifies the number of subdiagonals; \(k\) must be at least zero.

\(A\) single-precision array of dimension \((\text{lda}, n)\). If \texttt{uplo} == 'U' or 'u', the leading \((k+1)\times n\) 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 \(k\times k\) triangle of the array \(A\) is not referenced. If \texttt{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 referenced.

\textbf{x} single-precision array of length at least \((1+(n-1)\times \text{abs}(\text{incx}))\).

On entry, \(x\) contains the \(n\)-element right-hand side vector \(b\). On exit, it is overwritten with the solution vector \(x\).

\textbf{incx} storage spacing between elements of \(x\); \text{incx} must not be zero.

Output

\(x\) updated to contain the solution vector \(x\) that solves \(\text{op}(A) \times x = b\).

Reference: \url{http://www.netlib.org/blas/stbsv.f}
CHAPTER 3  Single-Precision BLAS2 Functions

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if incx == 0 or n &lt; 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function cublasStpmv()

```c
void
cublasStpmv (char uplo, char trans, char diag, int n,
             const float *AP, float *x, int incx)
```

performs one of the matrix-vector operations

\[ x = \text{op}(A) \times x, \]

where \( \text{op}(A) = A \text{ or } \text{op}(A) = A^T \),

\( x \) is an \( n \)-element single-precision vector, and \( A \) is an \( n \times n \), unit or non-unit, upper or lower, triangular matrix consisting of single-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 \( \text{op}(A) \).
  - If `trans` == 'N' or 'n', \( \text{op}(A) = A \).
  - If `trans` == 'T', 't', 'C', or 'c', \( \text{op}(A) = A^T \).

- **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.

- **AP** single-precision array with at least \( (n \times (n + 1))/2 \) 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 \( i \leq j \), \( A[i,j] \) is stored in \( AP[i + (j \times (j + 1)/2)] \). 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 \( i \geq j \), \( A[i,j] \) is stored in \( AP[i + ((2 \times n - j + 1) \times j)/2] \).
Function cublasStpsv()

```c
void

void
cublasStpsv (char uplo, char trans, char diag, int n,
const float *AP, float *X, int incx)

solves one of the systems of equations

\[ \text{op}(A) \times x = b, \]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

\( b \) 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.
CHAPTER 3 Single-Precision BLAS2 Functions

Input

**uplo** 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^T.

**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.

**AP** single-precision array with at least \((n \times (n + 1)) / 2\) elements. If
uplo == 'U' or 'u', array AP contains the upper triangular matrix A,
packed sequentially, column by column; that is, if \(i \leq j\), \(A[i, j]\) is
stored in \(AP[i + (j \times (j + 1)) / 2]\). If uplo == 'L' or 'l', array AP
contains the lower triangular matrix A, packed sequentially, column by
column; that is, if \(i \geq j\), \(A[i, j]\) is stored in
\(AP[i + (2 \times n - j + 1) \times j / 2]\). When diag == 'U' or 'u', the
diagonal elements of A are not referenced and are assumed to be unity.

**x** single-precision array of length at least \((1 + (n - 1) \times \text{abs(incx)})\).
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

x updated to contain the solution vector x that solves \(\text{op}(A) \times x = b\).

Reference: [http://www.netlib.org/blas/stpsv.f](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 or n < 0

**CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU
Function `cublasStrmv()`

```c
void cublasStrmv (char uplo, char trans, char diag, int n,
     const float *A, int lda, float *x, int incx)
```

performs one of the matrix-vector operations

\[ x = \text{op}(A) \times x, \]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

where \( \text{op}(A) \) is either \( A \) or its transpose \( A^T \).

\( x \) is an \( n \)-element single-precision vector, and \( A \) is an \( n \times n \), unit or non-unit, upper or lower, triangular matrix consisting of single-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 \( \text{op}(A) \). If `trans` == `'N'` or `'n'`, \( \text{op}(A) = A \).
  - If `trans` == `'T'`, `'t'`, `'C'`, or `'c'`, \( \text{op}(A) = A^T \).
- `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.
- `n` specifies the number of rows and columns of the matrix \( A \); \( n \) must be at least zero.
- `A` single-precision array of dimensions \((\text{lda}, n)\). If `uplo` == `'U'` or `'u'`, the leading \( n \times n \) 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 \( n \times n \) 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 \( A \); \( \text{lda} \) must be at least \( \max(1, n) \).
- `x` single-precision array of length at least \( 1 + (n-1) \times \text{abs}(\text{incx}) \).
  - On entry, \( x \) contains the source vector. On exit, \( x \) is overwritten with the result vector.
- `incx` the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
CHAPTER 3 Single-Precision BLAS2 Functions

Reference: [http://www.netlib.org/blas/strmv.f](http://www.netlib.org/blas/strmv.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 cublasStrsv()

```c
void

cublasStrsv (char uplo, char trans, char diag, int n,
             const float *A, int lda, float *x, int incx)
```

Solves a system of equations

\[ \text{op}(A) \cdot x = b, \]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

\( b \) 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

- **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 \( \text{op}(A) \). If \( trans == 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  - If \( trans == 'T', 't', 'C', \) or \( 'c' \), \( \text{op}(A) = A^T \).
**Input (continued)**

- **diag** specifies whether or not \( A \) is a unit triangular matrix.
  - If \( \text{diag} = 'U' \) or \( 'u' \), \( A \) is assumed to be unit triangular.
  - If \( \text{diag} = 'N' \) 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.
- **A** single-precision array of dimensions \((\text{lda}, n)\). If \( \text{uplo} = 'U' \) or \( 'u' \), \( A \) contains the upper triangular part of the symmetric matrix, and the strictly lower triangular part is not referenced. If \( \text{uplo} = 'L' \) 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 \( A \); \( \text{lda} \) must be at least \( \max(1, n) \).
- **x** single-precision array of length at least \( (1 + (n-1) \times \text{abs}(\text{incx})) \).
  - 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 \( x \); \( \text{incx} \) must not be zero.

**Output**

- \( x \) updated to contain the solution vector \( x \) that solves \( \text{op}(A) \times x = b \).

**Reference:** [http://www.netlib.org/blas/strsv.f](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 \( \text{incx} = 0 \) or \( n < 0 \)
- **CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU
CHAPTER 3
Single-Precision BLAS2 Functions

Single-Precision Complex BLAS2 Functions

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

- “Function cublasCgbmv()” on page 108
- “Function cublasCgemv()” on page 109
- “Function cublasCgerc()” on page 111
- “Function cublasCgeru()” on page 112
- “Function cublasChbmv()” on page 113
- “Function cublasChemv()” on page 115
- “Function cublasCher()” on page 116
- “Function cublasCher2()” on page 117
- “Function cublasChpmv()” on page 119
- “Function cublasChpr()” on page 120
- “Function cublasChpr2()” on page 121
- “Function cublasCtbbmv()” on page 123
- “Function cublasCtbsv()” on page 125
- “Function cublasCtpmv()” on page 126
- “Function cublasCtps()” on page 128
- “Function cublasCtprmv()” on page 129
- “Function cublasCtrops()” on page 131
Function cublasCgbmv()

```c
void

cublasCgbmv (char trans, int m, int n, int kl, int ku,
            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

\[ y = \alpha \cdot \text{op}(A) \cdot x + \beta \cdot y, \text{ where} \]

\[ \text{op}(A) = A, \text{op}(A) = A^T, \text{or} \ \text{op}(A) = A^H; \]

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

**Input**

- **trans** specifies \(\text{op}(A)\). If \(\text{trans} == \text{'N'}\) or \text{'n'}, \(\text{op}(A) = A\).
  - If \(\text{trans} == \text{'T'}\) or \text{'t'}, \(\text{op}(A) = A^T\).
  - If \(\text{trans} == \text{'C'}\), or \text{'c'}, \(\text{op}(A) = A^H\).

- **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.

- **kl** 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** single-precision complex scalar multiplier applied to \(\text{op}(A)\).

- **A** single-precision complex array of dimensions \((\text{lda}, n)\). The leading \((\text{kl}+\text{ku}+1)\times n\) part of the array \(A\) must contain the band matrix \(A\), supplied column by column, with the leading diagonal of the matrix in row \((\text{ku}+1)\) of the array, the first superdiagonal starting at position 2 in row \(\text{ku}\), the first subdiagonal starting at position 1 in row \((\text{ku}+2)\), and so on. Elements in the array \(A\) that do not correspond to elements in the band matrix (such as the top left \(\text{ku} \times \text{ku}\) triangle) are not referenced.

- **lda** leading dimension \(A\); \(lda\) must be at least \((\text{kl}+\text{ku}+1)\).
CHAPTER 3 Single-Precision BLAS2 Functions

Input (continued)

- x: single-precision complex array of length at least
  \((1 + (n - 1) \times \text{abs}(\text{incx})) \text{ if } \text{trans} \neq \text{'N'} \text{ or } \text{'n'}, \) else at least
  \((1 + (m - 1) \times \text{abs}(\text{incx})) \).
- incx: specifies the increment for the 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.
- y: single-precision complex array of length at least
  \((1 + (m - 1) \times \text{abs}(\text{incy})) \text{ if } \text{trans} \neq \text{'N'} \text{ or } \text{'n'}, \) else at least
  \((1 + (n - 1) \times \text{abs}(\text{incy})) \). 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 \( y = \alpha \times \text{op}(A) \times x + \beta \times y \).

Reference: [http://www.netlib.org/blas/cgbmv.f](http://www.netlib.org/blas/cgbmv.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 \( m < 0, n < 0, \text{incx} = 0, \) or \( \text{incy} = 0 \)
- **CUBLAS_STATUS_EXECUTION_FAILED**: if function failed to launch on GPU

Function `cublasCgemv()`

```c
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

- \( y = \alpha \times \text{op}(A) \times x + \beta \times y \),
- where \( \text{op}(A) = A, \text{op}(A) = A^T, \) or \( \text{op}(A) = A^H \);
**CUDA CUBLAS Library**

The `alpha` and `beta` are single-precision complex scalars; and `x` and `y` are single-precision complex vectors. `A` is an `m×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`
- `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.
- `alpha` is a single-precision complex scalar multiplier applied to `op(A)`.
- `A` is a single-precision complex 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` is the leading dimension of the two-dimensional array used to store matrix `A`.
- `x` is a single-precision complex array of length at least
  - `(1 + (n-1) * abs(in cx))` if `trans == 'N'` or `n'`, else at least
  - `(1 + (m-1) * abs(in cx))`
- `incx` specifies the storage spacing for elements of `x`; `incx` must not be zero.
- `beta` is a single-precision complex scalar multiplier applied to vector `y`. If `beta` is zero, `y` is not read.
- `y` is a 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))`
- `incy` specifies 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](http://www.netlib.org/blas/cgemv.f)
Error status for this function can be retrieved via `cublasGetError()`.

**Error Status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if m &lt; 0, n &lt; 0, incx == 0, or incy == 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

**Function cublasCgerc()**

```c
void
cublasCgerc (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

\[
A = \alpha \times x \times y^H + A,
\]

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

```c
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.
- \( n \): specifies the number of columns of matrix \( A \); \( n \) must be at least zero.
- \( \alpha \): single-precision complex scalar multiplier applied to \( x \) \( y^T \).
- \( x \): single-precision complex array of length at least \( (1 + (m-1) \times \text{abs}(\text{incx})). \)
- \( \text{incx} \): the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- \( y \): single-precision complex array of length at least \( (1 + (n-1) \times \text{abs}(\text{incy})). \)
CHAPTER 3       Single-Precision BLAS2 Functions

CHAPTER 3       Single-Precision BLAS2 Functions

Input (continued)

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incy</td>
<td>the storage spacing between elements of y; incy must not be zero.</td>
</tr>
<tr>
<td>A</td>
<td>single-precision complex array of dimensions (lda,n).</td>
</tr>
<tr>
<td>lda</td>
<td>leading dimension of two-dimensional array used to store matrix A.</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>updated according to $A = \alpha \times x y^* + A$.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if $m &lt; 0$, $n &lt; 0$, incx == 0, or incy == 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function `cublasChbmv()`

```c
void
cublasChbmv (char uplo, int n, int k, cuComplex alpha,
             const cuComplex *A, int lda,
             const cuComplex *x, int incx,
             cuComplex beta, cuComplex *y, int incy)
```

performs the matrix-vector operation

$$y = \alpha \times A \times x + \beta \times y,$$

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>specifies whether the upper or lower triangular part of the Hermitian band matrix $A$ is being supplied. If $\text{uplo} = 'U'$ or 'u', the upper triangular part is being supplied. If $\text{uplo} = 'L'$ or 'l', the lower triangular part is being supplied.</td>
</tr>
<tr>
<td>n</td>
<td>specifies the number of rows and the number of columns of the symmetric matrix $A$; $n$ must be at least zero.</td>
</tr>
</tbody>
</table>
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 \( A \times x \).

**A** single-precision complex array of dimensions \((\text{lda}, n)\). If \( \text{uplo} = 'U' \) or \( 'u' \), the leading \((k + 1) \times n\) 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 \( k \times k \) triangle of array \( A \) is not referenced. When \( \text{uplo} = 'L' \) 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 \( k \times k \) 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 \( A \); \( \text{lda} \) must be at least \( k + 1 \).

**x** single-precision complex array of length at least \((1 + (n-1) \times \text{abs} (\text{incx}))\).

**incx** storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.

**beta** single-precision complex scalar multiplier applied to vector \( y \).

**y** single-precision complex array of length at least \((1 + (n-1) \times \text{abs} (\text{incy}))\). If \( \text{beta} \) is zero, \( y \) is not read.

**incy** storage spacing between elements of \( y \); \( \text{incy} \) must not be zero.

Output

**y** updated according to \( y = \alpha A \times x + \beta y \).

Reference: [http://www.netlib.org/blas/chbmv.f](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 \( k < 0, n < 0, \text{incx} = 0 \), or \( \text{incy} = 0 \)
- **CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU
Function cublasChemv()

void

cublasChemv (char uplo, int n, cuComplex alpha,
const cuComplex *A, int lda,
const cuComplex *x, int incx,
cuComplex beta, cuComplex *y, int incy)

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 \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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.

- **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 complex scalar multiplier applied to \( A \times x \).

- **A** single-precision complex array of dimensions \(( \text{lda}, n) \). If \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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 \( A \); \( \text{lda} \) must be at least \( \max(1, n) \).

- **x** single-precision complex array of length at least \((1 + (n-1) \times \text{abs(incx)}) \).

- **incx** storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
CUDA CUBLAS Library

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

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

Function cublasCher()

```c
void
cublasCher (char uplo, int n, float alpha,
            const cuComplex *x, int incx, cuComplex *A,
            int lda)
```

performs the Hermitian rank 1 operation

\[
A = \alpha x \cdot x^H + A,
\]

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 \( \text{lda} \) is the leading dimension of the two-dimensional array containing \( A \).

Input

- \( \text{uplo} \) specifies whether the matrix data is stored in the upper or the lower triangular part of array \( A \). If \( \text{uplo} \) is equals to 'U' or 'u', only the upper triangular part of \( A \) is referenced. If \( \text{uplo} \) is equals to 'L' or 'l', only the lower triangular part of \( A \) is referenced.
- \( n \) is the number of rows and columns of matrix \( A \); \( n \) must be at least zero.
CHAPTER 3  Single-Precision BLAS2 Functions

Input (continued)

**alpha**  single-precision scalar multiplier applied to
\[ x \cdot x^H. \]

**x**  single-precision complex array of length at least
\[(1 + (n - 1) \cdot \text{abs}(\text{incx})).\]

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

**A**  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
\[ A = \alpha \cdot x \cdot x^H + A. \]

Reference: [http://www.netlib.org/blas/cher.f](http://www.netlib.org/blas/cher.f)

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( n &lt; 0 ) or ( \text{incx} == 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function `cublasCher2()`

```c
void 

__cublasCher2__(
    char uplo, int n, cuComplex alpha,
    const cuComplex *x, int incx,
    const cuComplex *y, int incy,
    cuComplex *A, int lda)

) 

performs the Hermitian rank 2 operation
\[ A = \alpha \cdot x \cdot y^H + \alpha \cdot y \cdot x^H + A, \]```
where \( \alpha \) is a single-precision complex scalar, \( x \) and \( y \) are \( n \)-element single-precision complex vectors, and \( A \) is an \( n \times n \) Hermitian matrix consisting of single-precision complex elements.

**Input**

- \( \text{uplo} \): specifies whether the matrix data is stored in the upper or the lower triangular part of array \( A \). If \( \text{uplo} = 'U' \) or \( 'u' \), only the upper triangular part of \( A \) may be referenced and the lower triangular part of \( A \) is inferred. If \( \text{uplo} = 'L' \) 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 \( x \) and whose conjugate is applied to \( y \).
- \( x \): single-precision array of length at least \( 1 + (n-1) \times \text{abs} \( \text{incx} \) \).
- \( \text{incx} \): the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- \( y \): single-precision array of length at least \( 1 + (n-1) \times \text{abs} \( \text{incy} \) \).
- \( \text{incy} \): the storage spacing between elements of \( y \); \( \text{incy} \) must not be zero.
- \( A \): single-precision complex array of dimensions \((\text{lda}, n)\). If \( \text{uplo} = 'U' \) or \( 'u' \), \( A \) contains the upper triangular part of the Hermitian matrix, and the strictly lower triangular part is not referenced. If \( \text{uplo} = 'L' \) 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.
- \( \text{lda} \): leading dimension of \( A \); \( \text{lda} \) must be at least \( \max(1, n) \).

**Output**

- \( A \): updated according to

\[
A = \alpha \times x \times y^H + \alpha \times y \times x^H + A
\]

**Reference:** [http://www.netlib.org/blas/cher2.f](http://www.netlib.org/blas/cher2.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 \( n < 0, \text{incx} = 0 \), or \( \text{incy} = 0 \)
- `CUBLAS_STATUS_EXECUTION_FAILED`: if function failed to launch on GPU
Function cublasChpmv()

void
cublasChpmv (char uplo, int n, cuComplex alpha,
        const cuComplex *AP, const cuComplex *x,
        int incx, cuComplex beta, cuComplex *y,
        int incy)

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>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 ).</td>
</tr>
<tr>
<td>( n )</td>
<td>the number of rows and columns of matrix ( A ); ( n ) must be at least zero.</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>single-precision complex scalar multiplier applied to ( A x ).</td>
</tr>
<tr>
<td>( AP )</td>
<td>single-precision complex array with at least ( (n \times (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 &lt;= j ), ( A[i,j] ) is stored in ( AP[i + (j \times (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 &gt;= j ), ( A[i,j] ) is stored in ( AP[i + ((2 \times n - j + 1) \times j)/2] ). The imaginary parts of the diagonal elements need not be set; they are assumed to be zero.</td>
</tr>
<tr>
<td>( x )</td>
<td>single-precision complex array of length at least ( (1 + (n - 1) \times \text{abs}(\text{incx})) ).</td>
</tr>
<tr>
<td>incx</td>
<td>storage spacing between elements of ( x ); ( \text{incx} ) must not be zero.</td>
</tr>
<tr>
<td>( \beta )</td>
<td>single-precision scalar multiplier applied to vector ( y ).</td>
</tr>
<tr>
<td>( y )</td>
<td>single-precision array of length at least ( (1 + (n - 1) \times \text{abs}(\text{incy})) ). If ( \beta ) is zero, ( y ) is not read.</td>
</tr>
<tr>
<td>incy</td>
<td>storage spacing between elements of ( y ); ( \text{incy} ) must not be zero.</td>
</tr>
</tbody>
</table>
CUDA

CUBLAS Library

Output

\[ y \quad \text{updated according to} \quad y = \alpha A x + \beta y. \]

Reference: [http://www.netlib.org/blas/chpmv.f](http://www.netlib.org/blas/chpmv.f)

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

Error Status

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( n &lt; 0, \text{incx} == 0, \text{incy} == 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function \texttt{cublasChpr()}

```c
void
cublasChpr (char uplo, int n, float alpha,
            const cuComplex *x, int incx, cuComplex *AP)
```

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 that is supplied in packed form.

Input

- \texttt{uplo} specifies whether the matrix data is stored in the upper or the lower triangular part of array \( AP \). If \( \text{uplo} == 'U' \) or \( 'u' \), the upper triangular part of \( A \) is supplied in \( AP \). If \( \text{uplo} == 'L' \) or \( 'l' \), the lower triangular part of \( A \) is supplied in \( AP \).
- \texttt{n} is the number of rows and columns of matrix \( A \); \( n \) must be at least zero.
- \texttt{alpha} is a single-precision scalar multiplier applied to \( x \) and \( x^H \).
- \texttt{x} is a single-precision complex array of length at least \( (1 + (n - 1) \times \text{abs(incx)}) \).
CHAPTER 3 Single-Precision BLAS2 Functions

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
cublasChpr2 (char uplo, int n, cuComplex alpha,
             const cuComplex *x, int incx,
             const cuComplex *y, int incy, cuComplex *AP)

performs the Hermitian rank 2 operation

\[ A = \alpha x^* y^H + \alpha y^* x^H + A, \]

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

Input

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>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.</td>
</tr>
<tr>
<td>n</td>
<td>the number of rows and columns of matrix A; n must be at least zero.</td>
</tr>
<tr>
<td>alpha</td>
<td>single-precision complex scalar multiplier applied to ( x \cdot y^* ) and whose conjugate is applied to ( y \cdot x^* ).</td>
</tr>
<tr>
<td>x</td>
<td>single-precision complex array of length at least ( (1 + (n-1) \cdot \text{abs}(\text{incx})) ).</td>
</tr>
<tr>
<td>incx</td>
<td>the storage spacing between elements of x; incx must not be zero.</td>
</tr>
<tr>
<td>y</td>
<td>single-precision complex array of length at least ( (1 + (n-1) \cdot \text{abs}(\text{incy})) ).</td>
</tr>
<tr>
<td>incy</td>
<td>the storage spacing between elements of y; incy must not be zero.</td>
</tr>
<tr>
<td>AP</td>
<td>single-precision complex array with at least ( (n \cdot (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 \leq j ), ( A[i,j] ) is stored in ( \text{AP}[i + (j \cdot (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 \geq j ), ( A[i,j] ) is stored in ( \text{AP}[i + ((2 \cdot n-j+1) \cdot 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.</td>
</tr>
</tbody>
</table>

Output

A updated according to \( A = \alpha \cdot x \cdot y^* + \alpha \cdot y \cdot x^* + A \)

Reference: [http://www.netlib.org/blas/chpr2.f](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 \( n < 0, \text{incx} = 0, \) or \( \text{incy} = 0 \)
- `CUBLAS_STATUS_EXECUTION_FAILED` if function failed to launch on GPU
Function cublasCtbmv()

void cublasCtbmv (char uplo, char trans, char diag, int n,
        int k, const cuComplex *A, int lda,
        cuComplex *x, int incx)

performs one of the matrix-vector operations

\[ x = \text{op}(A) \times x, \]

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

\( x \) 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

- **uplo** specifies whether the matrix \( A \) is an upper or lower triangular band matrix. If \( \text{uplo} == \text{'}U\text{' or } \text{'}u\text{'} \), \( A \) is an upper triangular band matrix. If \( \text{uplo} == \text{'}L\text{' or } \text{'}l\text{'} \), \( A \) is a lower triangular band matrix.

- **trans** specifies \( \text{op}(A) \). If \( \text{trans} == \text{'}N\text{' or } \text{'}n\text{'} \), \( \text{op}(A) = A \).
  If \( \text{trans} == \text{'}T\text{' or } \text{'}t\text{'} \), \( \text{op}(A) = A^T \).
  If \( \text{trans} == \text{'}C\text{' or } \text{'}c\text{'} \), \( \text{op}(A) = A^H \).

- **diag** specifies whether or not matrix \( A \) is unit triangular. If \( \text{diag} == \text{'}U\text{' or } \text{'}u\text{'} \), \( A \) is assumed to be unit triangular. If \( \text{diag} == \text{'}N\text{' or } \text{'}n\text{'} \), \( A \) 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 \( \text{uplo} == \text{'}U\text{' or } \text{'}u\text{'} \), \( k \) specifies the number of superdiagonals. If \( \text{uplo} == \text{'}L\text{' or } \text{'}l\text{'} \), \( k \) specifies the number of subdiagonals; \( k \) must at least be zero.
Input (continued)

\( A \) single-precision complex array of dimension \((\text{lda}, n)\). If \( \text{uplo} = 'U' \) or \( 'u' \), the leading \((k+1) \times n\) 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 \( k \times k \) triangle of the array \( A \) is not referenced. If \( \text{uplo} = 'L' \) 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 \( k \times k \) triangle of the array is not referenced.

\( \text{lda} \) is the leading dimension of \( A \); \( \text{lda} \) must be at least \( k+1 \).

\( x \) single-precision complex array of length at least \((1 + (n - 1) \times \text{abs}(\text{incx}))\).

On entry, \( x \) contains the source vector. On exit, \( x \) is overwritten with the result vector.

\( \text{incx} \) specifies the storage spacing for elements of \( x \); \( \text{incx} \) must not be zero.

Output

\( x \) updated according to \( x = \text{op}(A) \times x \).

Reference: [http://www.netlib.org/blas/ctbmv.f](http://www.netlib.org/blas/ctbmv.f)

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

Error Status

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{CUBLAS_STATUS_NOT_INITIALIZED}</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>\texttt{CUBLAS_STATUS_INVALID_VALUE}</td>
<td>if ( \text{incx} = 0 ), ( k &lt; 0 ), or ( n &lt; 0 )</td>
</tr>
<tr>
<td>\texttt{CUBLAS_STATUS_ALLOC_FAILED}</td>
<td>if function cannot allocate enough internal scratch vector memory</td>
</tr>
<tr>
<td>\texttt{CUBLAS_STATUS_EXECUTION_FAILED}</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
Function cublasCtbsv()

```c
void
cublasCtbsv (char uplo, char trans, char diag, int n,
             int k, const cuComplex *A, int lda,
             cuComplex *X, int incx)
```

solves one of the systems of equations

\[
op(A) \times x = b,
\]

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.

- **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 \( A \) 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.

- **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.
CUDA

CUBLAS Library

Function cublasCtpmv()

```c
void
void
```

performs one of the matrix-vector operations

```
void cublasCtpmv (char uplo, char trans, char diag, int n,
const cuComplex *AP, cuComplex *x, int incx)
```

where \( \text{op}(A) = A, \text{op}(A) = A^T, \text{or } \text{op}(A) = A^H \);
x is an n-element single-precision complex vector, and A is an n×n, unit or non-unit, upper or lower, triangular matrix consisting of single-precision complex elements.

Input

<table>
<thead>
<tr>
<th>uplo</th>
<th>specifies whether the matrix A is an upper or lower triangular matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If uplo == 'U' or 'u', A is an upper triangular matrix.</td>
</tr>
<tr>
<td></td>
<td>If uplo == 'L' or 'l', A is a lower triangular matrix.</td>
</tr>
<tr>
<td>trans</td>
<td>specifies ( \text{op}(A) ). If trans == 'N' or 'n', ( \text{op}(A) = A ).</td>
</tr>
<tr>
<td></td>
<td>If trans == 'T' or 't', ( \text{op}(A) = A^T ).</td>
</tr>
<tr>
<td></td>
<td>If trans == 'C', or 'c', ( \text{op}(A) = A^H ).</td>
</tr>
<tr>
<td>diag</td>
<td>specifies whether or not matrix A is unit triangular.</td>
</tr>
<tr>
<td></td>
<td>If diag == 'U' or 'u', A is assumed to be unit triangular.</td>
</tr>
<tr>
<td></td>
<td>If diag == 'N' or 'n', A is not assumed to be unit triangular.</td>
</tr>
<tr>
<td>n</td>
<td>specifies the number of rows and columns of the matrix A; n must be at least zero.</td>
</tr>
<tr>
<td>AP</td>
<td>single-precision complex array with at least ( (n * (n + 1))/2 ) elements.</td>
</tr>
<tr>
<td></td>
<td>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 \leq j ), ( A[i,j] ) is stored in ( \text{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 \geq j ), ( A[i,j] ) is stored in ( \text{AP}[i + ((2 * n - j + 1) * j)/2] ).</td>
</tr>
<tr>
<td>x</td>
<td>single-precision complex array of length at least ( 1 + (n - 1) * \text{abs}(\text{incx}) ). On entry, x contains the source vector. On exit, x is overwritten with the result vector.</td>
</tr>
<tr>
<td>incx</td>
<td>specifies the storage spacing for elements of x; incx must not be zero.</td>
</tr>
</tbody>
</table>

Output

| x     | updated according to \( x = \text{op}(A) * x \). |

Reference: http://www.netlib.org/blas/ctpmv.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 \( \text{incx} = 0 \) or \( n < 0 \)
Function cublasCtpsv()

void
cublasCtpsv (char uplo, char trans, char diag, int n,
        const cuComplex *AP, cuComplex *X, int incx)
solves one of the systems of equations

\[ \text{op}(A) \times x = b, \]

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

\( b \) 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

- **uplo** specifies whether the matrix is an upper or lower triangular matrix. If
  \( \text{uplo} = 'U' \) or \( 'u' \), \( A \) is an upper triangular matrix. If \( \text{uplo} = 'L' \)
  or \( 'l' \), \( A \) is a lower triangular matrix.

- **trans** specifies \( \text{op}(A) \). If \( \text{trans} = 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  If \( \text{trans} = 'T' \) or \( 't' \), \( \text{op}(A) = A^T \).
  If \( \text{trans} = 'C' \), or \( 'c' \), \( \text{op}(A) = A^H \).

- **diag** specifies whether \( A \) is unit triangular. If \( \text{diag} = 'U' \) or \( 'u' \), \( A \) is
  assumed to be unit triangular; that is, diagonal elements are not read
  and are assumed to be unity. If \( \text{diag} = 'N' \) 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.
CHAPTER 3 Single-Precision BLAS2 Functions

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

Function cublasCtrmv()

```c
void

void cublasCtrmv (char uplo, char trans, char diag, int n, 
                 const cuComplex *A, int lda, cuComplex *x, 
                 int incx)
```

performs one of the matrix-vector operations

\[ x = \text{op}(A) * x, \]

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

\( x \) 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.
NVIDIA
CUDA CUBLAS Library

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>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.</td>
</tr>
<tr>
<td>trans</td>
<td>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) = Aᴴ.</td>
</tr>
<tr>
<td>diag</td>
<td>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.</td>
</tr>
<tr>
<td>n</td>
<td>specifies the number of rows and columns of the matrix A; n must be at least zero.</td>
</tr>
<tr>
<td>A</td>
<td>single-precision complex 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.</td>
</tr>
<tr>
<td>lda</td>
<td>leading dimension of A; lda must be at least max(1,n).</td>
</tr>
<tr>
<td>x</td>
<td>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.</td>
</tr>
<tr>
<td>incx</td>
<td>the storage spacing between elements of x; incx must not be zero.</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

Reference: [http://www.netlib.org/blas/ctrmv.f](http://www.netlib.org/blas/ctrmv.f)

Error status for this function can be retrieved via [cublasGetError()](http://www.netlib.org/blas/ctrmv.f).

Error Status

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if incx == 0 or n &lt; 0</td>
</tr>
</tbody>
</table>
CHAPTER 3  Single-Precision BLAS2 Functions

Function cublasCtrsv()

void cublasCtrsv (char uplo, char trans, char diag, int n,
        const cuComplex *A, int lda, cuComplex *x,
        int incx)

solves a system of equations

\[ \text{op}(A) \times x = b, \]
where \( \text{op}(A) = A, \text{op}(A) = A^T, \text{or } \text{op}(A) = A^H; \)

\( b \) 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 \( \text{lda} \) 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

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

\[ A \] single-precision complex array of dimensions \((\text{lda}, n)\). If \(\text{uplo} == 'U' \) or \('u'\), \(A\) contains the upper triangular part of the symmetric matrix, and the strictly lower triangular part is not referenced. If \(\text{uplo} == 'L' \) or \('l'\), \(A\) contains the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.

\(\text{lda}\) leading dimension of the two-dimensional array containing \(A\); \(\text{lda}\) must be at least \(\max(1, n)\).

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

\(\text{incx}\) the storage spacing between elements of \(x\); \(\text{incx}\) must not be zero.

Output

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

Reference: [http://www.netlib.org/blas/ctrsv.f](http://www.netlib.org/blas/ctrsv.f)

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

Error Status

- \text{CUBLAS_STATUS_NOT_INITIALIZED} if CUBLAS library was not initialized
- \text{CUBLAS_STATUS_INVALID_VALUE} if \(\text{incx} == 0\) or \(n < 0\)
- \text{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 134
- “Double-Precision Complex BLAS2 functions” on page 158
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 135
- “Function cublasDgemv()” on page 136
- “Function cublasDger()” on page 138
- “Function cublasDsbmv()” on page 139
- “Function cublasDspmv()” on page 141
- “Function cublasDspr()” on page 142
- “Function cublasDspr2()” on page 143
- “Function cublasDsymv()” on page 144
- “Function cublasDsyr()” on page 146
- “Function cublasDsyr2()” on page 147
- “Function cublasDtbbmv()” on page 148
- “Function cublasDtbsv()” on page 150
- “Function cublasDtpmv()” on page 152
- “Function cublasDtpsv()” on page 153
- “Function cublasDtrmv()” on page 154
- “Function cublasDtrsv()” on page 156
Function cublasDgbmv()

```c
void
cublasDgbmv (char trans, int m, int n, int kl, int ku,
  double alpha, const double *A, int lda,
  const double *x, int incx, double beta,
  double *y, int incy)
```

performs one of the matrix-vector operations

\[ y = \alpha \text{op}(A) \cdot x + \beta y, \]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

\( \alpha \) and \( \beta \) are double-precision scalars, and \( x \) and \( y \) are double-precision vectors. \( A \) is an \( m \times n \) band matrix consisting of double-precision elements with \( k_l \) subdiagonals and \( k_u \) superdiagonals.

**Input**

- `trans` specifies \( \text{op}(A) \). If `trans == 'N'` or `'n'`, \( \text{op}(A) = A \).
  - If `trans == 'T'`, `'t'`, `'C'`, or `'c'`, \( \text{op}(A) = A^T \).
- `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.
- `kl` specifies the number of subdiagonals of matrix \( A \); \( k_l \) must be at least zero.
- `ku` specifies the number of superdiagonals of matrix \( A \); \( k_u \) must be at least zero.
- `alpha` double-precision scalar multiplier applied to \( \text{op}(A) \).
- `A` double-precision array of dimensions \((\text{lda}, n)\). The leading \((k_l+k_u+1)\times n\) part of the array \( A \) must contain the band matrix \( A \), supplied column by column, with the leading diagonal of the matrix in row \((k_u+1)\) of the array, the first superdiagonal starting at position 2 in row \( k_u \), the first subdiagonal starting at position 1 in row \((k_u+2)\), and so on. Elements in the array \( A \) that do not correspond to elements in the band matrix (such as the top left \( k_u \times k_u \) triangle) are not referenced.
- `lda` leading dimension \( A \); \( \text{lda} \) must be at least \((k_l+k_u+1)\).
- `x` double-precision array of length at least \((1+(n-1) \times \text{abs}(\text{incx}))\) if `trans == 'N'` or `'n'`, else at least \((1+(m-1) \times \text{abs}(\text{incx}))\).
- `incx` specifies the increment for the elements of \( x \); `incx` must not be zero.
CUDA CUBLAS Library

Function cublasDgemv()

```c
void

    cublasDgemv (char trans, int m, int n, double alpha,
               const double *A, int lda, const double *x,
               int incx, double beta, double *y, int incy)
```

performs one of the matrix-vector operations

```c
    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 \( A \) is stored in column-major format, and \( \text{lda} \) is the leading dimension of the two-dimensional array in which \( A \) is stored.

**Input**

- \texttt{trans} specifies \( \text{op}(A) \). If \( \text{trans} == 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  
  If \( \text{trans} == 'T', 't', 'C', \) or \( 'c' \), \( \text{op}(A) = A^T \).

- \texttt{m} specifies the number of rows of matrix \( A \); \texttt{m} must be at least zero.

- \texttt{n} specifies the number of columns of matrix \( A \); \texttt{n} must be at least zero.

- \texttt{alpha} double-precision scalar multiplier applied to \( \text{op}(A) \).

- \texttt{A} double-precision array of dimensions \( (\text{lda}, n) \) if \( \text{trans} == 'N' \) or \( 'n' \), of dimensions \( (\text{lda}, m) \) otherwise; \texttt{lda} must be at least \( \max(1, m) \) if \( \text{trans} == 'N' \) or \( 'n' \) and at least \( \max(1, n) \) otherwise.

- \texttt{lda} leading dimension of two-dimensional array used to store matrix \( A \).

- \texttt{x} double-precision array of length at least \( 1 + (n - 1) \times \text{abs} (\text{incx}) \) if \( \text{trans} == 'N' \) or \( 'n' \), else at least \( 1 + (m - 1) \times \text{abs} (\text{incx}) \).

- \texttt{incx} specifies the storage spacing for elements of \( x \); \texttt{incx} must not be zero.

- \texttt{beta} double-precision scalar multiplier applied to vector \( y \). If \texttt{beta} is zero, \( y \) is not read.

- \texttt{Y} double-precision array of length at least \( 1 + (m - 1) \times \text{abs} (\text{incy}) \) if \( \text{trans} == 'N' \) or \( 'n' \), else at least \( 1 + (n - 1) \times \text{abs} (\text{incy}) \).

- \texttt{incy} the storage spacing between elements of \( y \); \texttt{incy} must not be zero.

**Output**

- \( Y \) updated according to \( y = \alpha \times \text{op}(A) \times x + \beta \times y \).

**Reference:** [http://www.netlib.org/blas/dgemv.f](http://www.netlib.org/blas/dgemv.f)

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

**Error Status**

- \texttt{CUBLAS_STATUS_NOT_INITIALIZED} if CUBLAS library was not initialized

- \texttt{CUBLAS_STATUS_INVALID_VALUE} if \( m < 0, n < 0, \text{incx} = 0 \), or \( \text{incy} = 0 \)

- \texttt{CUBLAS_STATUS_ARCH_MISMATCH} if function invoked on device that does not support double precision

- \texttt{CUBLAS_STATUS_EXECUTION_FAILED} if function failed to launch on GPU
Function cublasDger()

```c
void cublasDger (int m, int n, double alpha, const double *x,
                int incx, const double *y, int incy,
                double *A, int lda)
```

performs the symmetric rank 1 operation

\[
A = \alpha \cdot x \cdot y^T + A,
\]

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 \(m \times n\) 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 \(x \cdot y^T\).
- **\(x\)** double-precision array of length at least \((1 + (m - 1) \cdot \text{abs}(\text{incx}))\).
- **\(\text{incx}\)** the storage spacing between elements of \(x\); \(\text{incx}\) must not be zero.
- **\(y\)** double-precision array of length at least \((1 + (n - 1) \cdot \text{abs}(\text{incy}))\).
- **\(\text{incy}\)** the storage spacing between elements of \(y\); \(\text{incy}\) must not be zero.
- **\(A\)** double-precision array of dimensions \((\text{lda}, n)\).
- **\(\text{lda}\)** leading dimension of two-dimensional array used to store matrix \(A\).

Output

- **\(A\)** updated according to \(A = \alpha \cdot x \cdot y^T + A\).

Reference: [http://www.netlib.org/blas/dger.f](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 \(m < 0, n < 0, \text{incx} = 0\), or \(\text{incy} = 0\)
CHAPTER 4  Double-Precision BLAS2 Functions

Error Status (continued)

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function cublasDsbmv()

```c
void
cublasDsbmv (char uplo, int n, int k, double alpha,
    const double *A, int lda, const double *x,
    int incx, double beta, double *y, int incy)
```

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

- **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; \(k\) must be at least zero.

- **alpha** double-precision scalar multiplier applied to \(A x\).
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 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; lda 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.

Y  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/dsbmv.f

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if k &lt; 0, n &lt; 0, incx == 0, or incy == 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
Function `cublasDspmv()`

```c
void
cublasDspmv (char uplo, int n, double alpha,
const double *AP, const double *x, int incx,
double beta, double *y, int incy)
```

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 \( A \) * \( x \).
- `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 \( i \leq j, A[i,j] \) 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 \( i \geq j, A[i,j] \) is stored in `AP[i + ((2 * n - j + 1) * j)/2]`.
- `x` double-precision array of length at least \( 1 + (n - 1) * \text{abs}(\text{incx}) \).
- `incx` storage spacing between elements of \( x \); `incx` must not be zero.
- `beta` double-precision scalar multiplier applied to vector \( y \).
- `Y` double-precision array of length at least \( 1 + (n - 1) * \text{abs}(\text{incy}) \).
- `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/dspmv.f](http://www.netlib.org/blas/dspmv.f)
Error status for this function can be retrieved via `cublasGetError()`.

**Error Status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if n &lt; 0, incx == 0, or incy == 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

**Function cublasDspr()**

```c
void
cublasDspr (char uplo, int n, double alpha, const double *x, int incx, double *AP)
```

performs the symmetric rank 1 operation

\[
A = \alpha x \cdot x^T + A,
\]

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** is the number of rows and columns of matrix \(A\); \(n\) must be at least zero.
- **alpha** is the double-precision scalar multiplier applied to \(x \cdot x^T\).
- **x** is a double-precision array of length at least \((1 + (n - 1) \cdot \text{abs}(\text{incx}))\).
- **incx** storage spacing between elements of \(x\); \(\text{incx}\) must not be zero.
- **AP** is a double-precision array with at least \((n \cdot (n + 1)) / 2\) 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 \(AP[i + (j \cdot (j + 1) / 2)]\). 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 \(AP[i + ((2 \cdot n - j - 1) \cdot j) / 2]\).
CHAPTER 4 Double-Precision BLAS2 Functions

Output

\[ A \text{ updated according to } A = \alpha \cdot x \cdot x^T + A. \]

Reference: http://www.netlib.org/blas/dspr.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 \( \text{incy} == 0 \)
- `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 `cublasDspr2()`

```c
void

\text{cublasDspr2} \ (\text{char uplo, int n, double alpha, 
const double *x, int incx, const double *y, 
int incy, double *AP})

```

performs the symmetric rank 2 operation

\[ A = \alpha \cdot x \cdot y^T + \alpha \cdot y \cdot x^T + A, \]

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

- \( \text{uplo} \) specifies whether the matrix data is stored in the upper or the lower triangular part of array \( A \). If \( \text{uplo} == 'U' \) or \( 'u' \), only the upper triangular part of \( A \) may be referenced and the lower triangular part of \( A \) is inferred. If \( \text{uplo} == 'L' \) 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 scalar multiplier applied to \( x \cdot y^T \) and \( y \cdot x^T \).
- \( x \) double-precision array of length at least \((1 + (n - 1) \cdot \text{abs(incy)}) \).
- \( \text{incy} \) storage spacing between elements of \( x \); \( \text{incy} \) must not be zero.
- \( y \) double-precision array of length at least \((1 + (n - 1) \cdot \text{abs(incy)}) \).
Function cublasDsymv()

```c
void
cublasDsymv (char uplo, int n, double alpha,
            const double *A, int lda, const double *x,
            int incx, double beta, double *y, int incy)
```

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 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 * x.

- **A** 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 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; lda must be at least max(1, n).

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

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

- **beta** double-precision scalar multiplier applied to vector y.

- **y** 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](http://www.netlib.org/blas/dsymv.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
**Function cublasDsyr()**

```c
void cublasDsyr (char uplo, int n, double alpha,
                const double *x, int incx, double *A,
                int lda)
```

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 \( \text{lda} \) 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 \( \text{uplo} == 'U' \) or \( 'u' \), only the upper triangular part of \( A \) is referenced. If \( \text{uplo} == 'L' \) or \( 'l' \), only the lower triangular part of \( A \) is referenced.
- **n** is the number of rows and columns of matrix \( A \); \( n \) must be at least zero.
- **alpha** is double-precision scalar multiplier applied to \( x \times x^T \).
- **x** is double-precision array of length at least \((1+(n-1)\times\text{abs(incx)})\).
- **incx** is the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- **A** is double-precision array of dimensions \((\text{lda}, n)\). If \( \text{uplo} == 'U' \) or \( 'u' \), \( A \) contains the upper triangular part of the symmetric matrix, and the strictly lower triangular part is not referenced. If \( \text{uplo} == 'L' \) or \( 'l' \), \( A \) contains the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.
- **lda** is leading dimension of the two-dimensional array containing \( A \); \( \text{lda} \) must be at least \( \text{max}(1, n) \).

**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
CHAPTER 4 Double-Precision BLAS2 Functions

Output

\[ A \text{ updated according to } A = \alpha x x^T + A. \]

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

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

Error Status

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( n &lt; 0 ) or ( \text{incx} = 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function \texttt{cublasDsy2()}

\begin{verbatim}
void 
cublasDsy2 (char uplo, int n, double alpha, 
             const double *x, int incx, const double *y, 
             int incy, double *A, int lda)
\end{verbatim}

performs the symmetric rank 2 operation

\[
A = \alpha x x^T + \alpha y y^T + A,
\]

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.

Input

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>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.</td>
</tr>
<tr>
<td>n</td>
<td>the number of rows and columns of matrix ( A ); ( n ) must be at least zero.</td>
</tr>
<tr>
<td>alpha</td>
<td>double-precision scalar multiplier applied to ( x y^T ) and ( y x^T ).</td>
</tr>
<tr>
<td>x</td>
<td>double-precision array of length at least ( (1 + (n - 1) \times \text{abs(incx)}) ).</td>
</tr>
<tr>
<td>incx</td>
<td>storage spacing between elements of ( x ); ( \text{incx} ) must not be zero.</td>
</tr>
<tr>
<td>y</td>
<td>double-precision array of length at least ( (1 + (n - 1) \times \text{abs(incy)}) ).</td>
</tr>
</tbody>
</table>
Input (continued)

<table>
<thead>
<tr>
<th>incy</th>
<th>storage spacing between elements of y; incy must not be zero.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>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.</td>
</tr>
<tr>
<td>lda</td>
<td>leading dimension of A; lda must be at least max(1, n).</td>
</tr>
</tbody>
</table>

Output

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

Reference: [http://www.netlib.org/blas/dsyr2.f](http://www.netlib.org/blas/dsyr2.f)

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

Error Status

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if n &lt; 0, incx == 0, or incy == 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function `cublasDtbmv()`

```c
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 = \text{op}(A) * x,

where \text{op}(A) = A or \text{op}(A) = A^T,

x is an n-element double-precision vector, and A is an n×n, unit or non-unit, upper or lower, triangular band matrix consisting of double-precision elements.
CHAPTER 4  Double-Precision BLAS2 Functions

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 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.

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; lda 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/dtbbmv.f
Error status for this function can be retrieved via `cublasGetError()`.  

### Error Status

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CUBLAS_STATUS_NOT_INITIALIZED</code></td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_INVALID_VALUE</code></td>
<td>if <code>incx == 0</code>, <code>k &lt; 0</code>, or <code>n &lt; 0</code></td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ARCH_MISMATCH</code></td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ALLOC_FAILED</code></td>
<td>if function cannot allocate enough internal scratch vector memory</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_EXECUTION_FAILED</code></td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

### Function cublasDtbsv()

```c
void cublasDtbsv (char uplo, char trans, char diag, int n,
                 int k, const double *A, int lda, double *X, int incx)
```

solves one of the systems of equations

\[ \text{op}(A) \times x = b, \]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

\( b \) 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 \( \text{op}(A) \). If `trans == 'N'` or `n'`, \( \text{op}(A) = A \).
  - If `trans == 'T'`, `'t'`, `'C'`, or `'c'`, \( \text{op}(A) = A^T \).
- **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.
CHAPTER 4  Double-Precision BLAS2 Functions

Input (continued)

\textbf{k} 

specifies the number of superdiagonals or subdiagonals.

If \texttt{uplo} == \textquoteleft U' or \textquoteleft u', \texttt{k} specifies the number of superdiagonals. If \texttt{uplo} == \textquoteleft L' or \textquoteleft l', \texttt{k} specifies the number of subdiagonals; \texttt{k} must be at least zero.

\textbf{A} 

double-precision array of dimension (lda, n). If \texttt{uplo} == \textquoteleft U' or \textquoteleft u', the leading \((k+1)\times n\) part of the array \texttt{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 \(k\times k\) triangle of the array \texttt{A} is not referenced. If \texttt{uplo} == \textquoteleft L' or \textquoteleft l', the leading \((k+1)\times n\) part of the array \texttt{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 referenced.

\textbf{x} 

double-precision array of length at least \((1+(n-1)\times \text{abs(incx)})\).

\textbf{incx} 

storage spacing between elements of \texttt{x}; \texttt{incx} must not be zero.

Output

\textbf{x} 

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

Reference: \url{http://www.netlib.org/blas/dtbsv.f}

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{CUBLAS_STATUS_NOT_INITIALIZED}</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>\texttt{CUBLAS_STATUS_INVALID_VALUE}</td>
<td>if incx == 0 or n &lt; 0</td>
</tr>
<tr>
<td>\texttt{CUBLAS_STATUS_ARCH_MISMATCH}</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>\texttt{CUBLAS_STATUS_EXECUTION_FAILED}</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
Function cublasDtpmv()

```c
void
cublasDtpmv (char uplo, char trans, char diag, int n,
            const double *AP, double *x, int incx)
```

performs one of the matrix-vector operations

\[ x = \text{op}(A) \times x, \]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

where \( \text{op}(A) \) is one of \( A \) or \( A^T \).

\( x \) 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**

- \( \text{uplo} \) specifies whether the matrix \( A \) is an upper or lower triangular matrix.
  - If \( \text{uplo} == 'U' \) or \( 'u' \), \( A \) is an upper triangular matrix.
  - If \( \text{uplo} == 'L' \) or \( 'l' \), \( A \) is a lower triangular matrix.

- \( \text{trans} \) specifies \( \text{op}(A) \).
  - If \( \text{trans} == 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  - If \( \text{trans} == 'T' \), \( 't' \), \( 'C' \), or \( 'c' \), \( \text{op}(A) = A^T \).

- \( \text{diag} \) specifies whether or not matrix \( A \) is unit triangular.
  - If \( \text{diag} == 'U' \) or \( 'u' \), \( A \) is assumed to be unit triangular.
  - If \( \text{diag} == 'N' \) 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.

- \( \text{AP} \) double-precision array with at least \( (n + (n + 1))/2 \) elements. If
  - \( \text{uplo} == 'U' \) or \( 'u' \), the array \( \text{AP} \) 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 \( \text{AP}[i + (j \times (j + 1)/2)] \).
  - If \( \text{uplo} == 'L' \) or \( 'l' \), array \( \text{AP} \) 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 \( \text{AP}[i + ((2 \times n - j + 1) \times j)/2] \).

- \( x \) double-precision array of length at least \( (1 + (n - 1) \times \text{abs}(\text{incx})) \).
  - On entry, \( x \) contains the source vector. On exit, \( x \) is overwritten with
    the result vector.

- \( \text{incx} \) specifies the storage spacing for elements of \( x \); \( \text{incx} \) must not be zero.

**Output**

- \( x \) updated according to \( x = \text{op}(A) \times x \).
Reference: http://www.netlib.org/blas/dtpmv.f

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

Error Status

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if incx == 0 or n &lt; 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ALLOC_FAILED</td>
<td>if function cannot allocate enough internal scratch vector memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function `cublasDtpsv()`

```c
void

cublasDtpsv (char uplo, char trans, char diag, int n,
            const double *AP, double *X, int incx)
```

solves one of the systems of equations

\[
\text{op}(A) \times x = b,
\]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

\( b \) 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.

Input

<table>
<thead>
<tr>
<th>uplo</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>trans</td>
<td>specifies ( \text{op}(A) ). If trans == 'N' or 'n', ( \text{op}(A) = A ). If trans == 'T', 't', 'C', or 'c', ( \text{op}(A) = A^T ).</td>
</tr>
<tr>
<td>diag</td>
<td>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.</td>
</tr>
</tbody>
</table>

```
Function cublasDtrmv()

```c
void

void cublasDtrmv (char uplo, char trans, char diag, int n,
                  const double *A, int lda, double *x,
                  int incx);
```

performs one of the matrix-vector operations

\[ x = \text{op}(A) \ast x, \]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),
\( x \) 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 \( \text{uplo} = 'U' \) or \( 'u' \), \( A \) is an upper triangular matrix. 
  - If \( \text{uplo} = 'L' \) or \( 'l' \), \( A \) is a lower triangular matrix. 

- **trans** specifies \( \text{op}(A) \). If \( \text{trans} = 'N' \) or \( 'n' \), \( \text{op}(A) = A \). 
  - If \( \text{trans} = 'T', 't', 'C', \) or \( 'c' \), \( \text{op}(A) = A^T \). 

- **diag** specifies whether \( A \) is a unit triangular matrix. If \( \text{diag} = 'U' \) or \( 'u' \), \( A \) is assumed to be unit triangular. If \( \text{diag} = 'N' \) 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. 

- **A** double-precision array of dimensions \((\text{lda}, n)\). If \( \text{uplo} = 'U' \) or \( 'u' \), the leading \( n \times n \) 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 \( \text{uplo} = 'L' \) or \( 'l' \), the leading \( n \times n \) 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 \( \text{diag} = 'U' \) or \( 'u' \), the diagonal elements of \( A \) are not referenced either, but are assumed to be unity. 

- **lda** leading dimension of \( A \); \( \text{lda} \) must be at least \( \max(1, n) \). 

- **x** double-precision array of length at least \( 1 + (n-1) \times \text{abs}(\text{incx}) \). 
  - On entry, \( x \) contains the source vector. 
  - On exit, \( x \) is overwritten with the result vector. 

- **incx** the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero. 

**Output**

- \( x \) updated according to 
  \[
  x = \text{op}(A) \times x 
  \]

**Reference:** [http://www.netlib.org/blas/dtrmv.f](http://www.netlib.org/blas/dtrmv.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 \( \text{incx} = 0 \) or \( n < 0 \).
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

\[ \text{op}(A) \cdot x = b, \]

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),

\( b \) 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 double-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

- **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 \( \text{op}(A) \). If \( trans == 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  
  If \( trans == 'T' \), \( 't' \), \( 'C' \), or \( 'c' \), \( \text{op}(A) = A^T \).

- **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.

- **n** specifies the number of rows and columns of the matrix \( A \); \( n \) must be 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).

x  double-precision array of length at least \(1 + (n - 1) \times \text{abs}(\text{incx})\).
   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

x  updated to contain the solution vector x that solves \(\text{op}(A) \times x = b\).

Reference: http://www.netlib.org/blas/dtrsv.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
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 159
- “Function cublasZgemv()” on page 161
- “Function cublasZgerc()” on page 162
- “Function cublasZgeru()” on page 163
- “Function cublasZher()” on page 168
- “Function cublasZher2()” on page 170
- “Function cublasZhpmv()” on page 171
- “Function cublasZhpr()” on page 173
- “Function cublasZhpr2()” on page 174
- “Function cublasZtbmv()” on page 175
- “Function cublasZtbsv()” on page 177
- “Function cublasZtpmv()” on page 179
- “Function cublasZtpsv()” on page 180
- “Function cublasZtrmv()” on page 182
- “Function cublasZtrsv()” on page 183
Function cublasZgbmv()

```c
void
cublasZgbmv (char trans, int m, int n, int kl, int ku, 
cuDoubleComplex alpha, 
const cuDoubleComplex *A, int lda, 
const cuDoubleComplex *x, int incx, 
cuDoubleComplex beta, cuDoubleComplex *y, 
int incy)
```

performs one of the matrix-vector operations

\[
y = \alpha \cdot \text{op}(A) \cdot x + \beta \cdot y, \quad \text{where}
\]

\[
\text{op}(A) = A, \quad \text{op}(A) = A^T, \quad \text{or} \quad \text{op}(A) = A^H;
\]

\(\alpha\) 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 \(kl\) subdiagonals and \(ku\) superdiagonals.

**Input**

- `trans` specifies \(\text{op}(A)\). If `trans` is `'N'` or `'n'`, \(\text{op}(A) = A\).
  - If `trans` is `'T'` or `'t'`, \(\text{op}(A) = A^T\).
  - If `trans` is `'C'`, or `'c'`, \(\text{op}(A) = A^H\).

- `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.

- `kl` 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 complex scalar multiplier applied to \(\text{op}(A)\).

- `A` double-precision complex array of dimensions \((\text{lda}, n)\). The leading \((kl+ku+1)\times n\) 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 \(ku\times ku\) triangle) are not referenced.

- `lda` leading dimension \(A\); `lda` must be at least \((kl+ku+1)\).
Input (continued)

- \(x\) double-precision complex array of length at least
  
  \((1 + (n-1) \cdot \text{abs}(\text{incx}))\) if \(\text{trans} == 'N'\) or \('n'\), else at least
  
  \((1 + (m-1) \cdot \text{abs}(\text{incx}))\).

- \(\text{incx}\) specifies the increment for the elements of \(x\); \(\text{incx}\) must not be zero.

- \(\beta\) double-precision complex scalar multiplier applied to vector \(y\). If \(\beta\) is zero, \(y\) is not read.

- \(y\) double-precision complex array of length at least
  
  \((1 + (m-1) \cdot \text{abs}(\text{incy}))\) if \(\text{trans} == 'N'\) or \('n'\), else at least
  
  \((1 + (n-1) \cdot \text{abs}(\text{incy}))\). If \(\beta\) is zero, \(y\) is not read.

- \(\text{incy}\) on entry, \(\text{incy}\) specifies the increment for the elements of \(y\); \(\text{incy}\) must not be zero.

Output

\(y\) updated according to

\[y = \alpha \cdot \text{op}(A) \cdot x + \beta \cdot y.\]

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

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

- \texttt{CUBLAS\_STATUS\_NOT\_INITIALIZED}\quad \text{if CUBLAS library was not initialized}

- \texttt{CUBLAS\_STATUS\_INVALID\_VALUE}\quad \text{if} \ m < 0, n < 0, \text{incx} == 0, \text{incy} == 0

- \texttt{CUBLAS\_STATUS\_ARCH\_MISMATCH}\quad \text{if function invoked on device that does not support double precision}

- \texttt{CUBLAS\_STATUS\_EXECUTION\_FAILED}\quad \text{if function failed to launch on GPU}
Function cublasZgemv()

```c
void
  cublasZgemv (char trans, int m, int n,
               cuDoubleComplex alpha,
               const cuDoubleComplex *A, int lda,
               const cuDoubleComplex *x, int incx,
               cuDoubleComplex beta, cuDoubleComplex *y,
               int incy)
```

performs one of the matrix-vector operations

\[
y = \alpha \cdot \text{op}(A) \cdot x + \beta \cdot y,
\]

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

\( \alpha \) 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.

**Input**

- **trans** specifies \( \text{op}(A) \). If \( trans = \'N' \) or \( \'n' \), \( \text{op}(A) = A \).
  - If \( trans = \'T' \) or \( \'t' \), \( \text{op}(A) = A^T \).
  - If \( trans = \'C' \) or \( \'c' \), \( \text{op}(A) = A^H \).
- **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.
- **alpha** double-precision complex scalar multiplier applied to \( \text{op}(A) \).
- **A** double-precision complex array of dimensions \((\text{lda}, n)\) if \( trans = \'N' \) or \( \'n' \), of dimensions \((\text{lda}, m)\) otherwise; \( \text{lda} \) 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** double-precision complex array of length at least
  \((1 + (n-1) \cdot \text{abs}(\text{incx})) \) if \( trans = \'N' \) or \( \'n' \), else at least \((1 + (m-1) \cdot \text{abs}(\text{incx})) \).
- **incx** specifies the storage spacing for elements of \( x \); \( \text{incx} \) must not be zero.
- **beta** double-precision complex scalar multiplier applied to vector \( y \). If \( \beta \) is zero, \( y \) is not read.
Input (continued)

| Y | double-precision complex array of length at least 
|   | \( (1 + (m-1) \times \text{abs}(\text{incy})) \) if \( \text{trans} \) == 'N' or 'n', else at least \( (1 + (n-1) \times \text{abs}(\text{incy})) \). |
| incy | the storage spacing between elements of \( y \); incy must not be zero. |

Output

\( Y \) updated according to \( y = \alpha \cdot \text{op}(A) \cdot x + \beta \cdot y \).

Reference: [http://www.netlib.org/blas/zgemv.f](http://www.netlib.org/blas/zgemv.f)

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

Error Status

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( m &lt; 0 ), ( n &lt; 0 ), ( \text{incx} = 0 ), or ( \text{incy} = 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function `cublasZgerc()`

```c
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

\[ A = \alpha \cdot x \cdot y^H + A, \]

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 \( m \times n \) matrix consisting of double-precision complex elements. Matrix \( A \) is stored in column-major format, and \( \text{lda} \) is the leading dimension of the two-dimensional array used to store \( A \).
CHAPTER 4 Double-Precision BLAS2 Functions

Reference: [http://www.netlib.org/blas/zgerc.f](http://www.netlib.org/blas/zgerc.f)

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

### Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CUBLAS_STATUS_NOT_INITIALIZED</code></td>
<td>If CUBLAS library was not initialized</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_INVALID_VALUE</code></td>
<td>If ( m &lt; 0, n &lt; 0, \text{incx} = 0 ), or ( \text{incy} = 0 )</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ARCH_MISMATCH</code></td>
<td>If function invoked on device that does not support double precision</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_EXECUTION_FAILED</code></td>
<td>If function failed to launch on GPU</td>
</tr>
</tbody>
</table>

### Function `cublasZgeru()`

```c
void

void cublasZgeru (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

\[
A = \alpha * x * y^\text{T} + A,
\]
where \(\text{alpha}\) 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 \(m\times n\) matrix consisting of double-precision complex elements. Matrix \(A\) is stored in column-major format, and \(\text{lda}\) 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.
- \(\text{alpha}\) double-precision complex scalar multiplier applied to \(x \ast y^\ast\).
- \(x\) double-precision complex array of length at least 
  \((1 + (m - 1) \times \text{abs}(\text{incx}))\).
- \(\text{incx}\) the storage spacing between elements of \(x\); \(\text{incx}\) must not be zero.
- \(y\) double-precision complex array of length at least 
  \((1 + (n - 1) \times \text{abs}(\text{incy}))\).
- \(\text{incy}\) the storage spacing between elements of \(y\); \(\text{incy}\) must not be zero.
- \(A\) double-precision complex array of dimensions \((\text{lda}, n)\).
- \(\text{lda}\) leading dimension of two-dimensional array used to store matrix \(A\).

**Output**

- \(A\) updated according to \(A = \text{alpha} \ast x \ast y^\ast + A\).

**Reference:** [http://www.netlib.org/blas/zgeru.f](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 \(m < 0, n < 0, \text{incx} = 0,\) or \(\text{incy} = 0\)
- `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()

```c
void
cublasZhbmv (char uplo, int n, int k,
c   cuDoubleComplex alpha,
   const cuDoubleComplex *A, int lda,
   const cuDoubleComplex *x, int incx,
   cuDoubleComplex beta, cuDoubleComplex *y,
   int incy)
```

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.

**Input**

- `uplo` 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 \( A x \).
**Input (continued)**

- $A$ double-precision complex array of dimensions $(\text{lda}, n)$. If $\text{uplo} == 'U'$ or 'u', the leading $(k + 1) \times n$ 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 $k \times k$ triangle of array $A$ is not referenced. When $\text{uplo} == 'L'$ 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 $k \times k$ triangle of array $A$ is not referenced. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero.

- $\text{lda}$ leading dimension of $A$; $\text{lda}$ must be at least $k + 1$.

- $x$ double-precision complex array of length at least $(1 + (n - 1) \times \text{abs}(\text{incx}))$.

- $\text{incx}$ storage spacing between elements of $x$; $\text{incx}$ must not be zero.

- $\text{beta}$ double-precision complex scalar multiplier applied to vector $y$.

- $y$ double-precision complex array of length at least $(1 + (n - 1) \times \text{abs}(\text{incy}))$. If $\text{beta}$ is zero, $y$ is not read.

- $\text{incy}$ storage spacing between elements of $y$; $\text{incy}$ must not be zero.

**Output**

- $y$ updated according to $y = \text{alpha} \times A \times x + \text{beta} \times y$.

**Reference:** [http://www.netlib.org/blas/zhbmv.f](http://www.netlib.org/blas/zhbmv.f)

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

**Error Status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if $k &lt; 0, n &lt; 0, \text{incx} == 0$, or $\text{incy} == 0$</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
Function cublasZhemv()

```c
void
cublasZhemv (char uplo, int n, cuDoubleComplex alpha,
             const cuDoubleComplex *A, int lda,
             const cuDoubleComplex *x, int incx,
             cuDoubleComplex beta, cuDoubleComplex *y,
             int incy)
```

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.

**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.
- `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 \( A x \).
- `A` 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 \( \max(1, n) \).
- `x` double-precision complex array of length at least \((1 + (n - 1) * \text{abs}(\text{incx}))\).
- `incx` storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
CUDA CUBLAS Library

References:
- [http://www.netlib.org/blas/zhemv.f](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 \( n < 0, \text{incx} == 0, \text{incy} == 0 \)
- **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()`

```c
void

cublasZher (char uplo, int n, double alpha,
             const cuDoubleComplex *x, int incx,
             cuDoubleComplex *A, int lda)
```

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 \( A \).
CHAPTER 4
Double-Precision BLAS2 Functions

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

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

Error Status

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( n ) &lt; 0 or ( \text{incx} = 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Input

- **uplo** specifies whether the matrix data is stored in the upper or the lower triangular part of array \( A \). If \( \text{uplo} = 'U' \) or \( 'u' \), only the upper triangular part of \( A \) is referenced. If \( \text{uplo} = 'L' \) or \( 'l' \), only the lower triangular part of \( A \) is referenced.
- **n** is the number of rows and columns of matrix \( A \); \( n \) must be at least zero.
- **alpha** is a double-precision scalar multiplier applied to \( x \ast x^H \).
- **x** is a double-precision complex array of length at least \((1 + (n - 1) \ast \text{abs(incx)})\).
- **incx** is the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- **A** is a double-precision complex array of dimensions \((\text{lda}, n)\). If \( \text{uplo} = 'U' \) or \( 'u' \), \( A \) contains the upper triangular part of the Hermitian matrix, and the strictly lower triangular part is not referenced. If \( \text{uplo} = 'L' \) 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** is the leading dimension of the two-dimensional array containing \( A \); \( \text{lda} \) must be at least \( \text{max}(1, n) \).

Output

- \( A \) is updated according to \( A = \alpha \ast x \ast x^H + A \).
Function cublasZher2()

```c
void
cublasZher2 (char uplo, int n, cuDoubleComplex alpha,
         const cuDoubleComplex *x, int incx,
         const cuDoubleComplex *y, int incy,
         cuDoubleComplex *A, int lda)
```

performs the Hermitian rank 2 operation

\[ A = \alpha x y^H + \alpha y x^H + A, \]

where \( \alpha \) is a double-precision complex scalar, \( x \) and \( y \) are \( n \)-element double-precision complex vectors, and \( A \) is an \( n \times n \) Hermitian matrix consisting of double-precision complex elements.

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uplo</code></td>
<td>Specifies whether the matrix data is stored in the upper or the lower triangular part of array ( A ). If <code>uplo</code> == 'U' or 'u', only the upper triangular part of ( A ) may be referenced and the lower triangular part of ( A ) is inferred. If <code>uplo</code> == 'L' or 'l', only the lower triangular part of ( A ) may be referenced and the upper triangular part of ( A ) is inferred.</td>
</tr>
<tr>
<td><code>n</code></td>
<td>The number of rows and columns of matrix ( A ); ( n ) must be at least zero.</td>
</tr>
<tr>
<td><code>alpha</code></td>
<td>Double-precision complex scalar multiplier applied to ( x ) and whose conjugate is applied to ( y ).</td>
</tr>
<tr>
<td><code>x</code></td>
<td>Double-precision array of length at least ( (1+(n-1) \times \text{abs}(\text{incx})) ).</td>
</tr>
<tr>
<td><code>incx</code></td>
<td>The storage spacing between elements of ( x ); \text{incx} must not be zero.</td>
</tr>
<tr>
<td><code>y</code></td>
<td>Double-precision array of length at least ( (1+(n-1) \times \text{abs}(\text{incy})) ).</td>
</tr>
<tr>
<td><code>incy</code></td>
<td>The storage spacing between elements of ( y ); \text{incy} must not be zero.</td>
</tr>
<tr>
<td><code>A</code></td>
<td>Double-precision complex array of dimensions ( (lda,n) ). If <code>uplo</code> == 'U' or 'u', ( A ) contains the upper triangular part of the Hermitian matrix, and the strictly lower triangular part is not referenced. If <code>uplo</code> == '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.</td>
</tr>
<tr>
<td><code>lda</code></td>
<td>Leading dimension of ( A ); \text{lda} must be at least ( \max(1,n) ).</td>
</tr>
</tbody>
</table>
CHAPTER 4 Double-Precision BLAS2 Functions

Reference: http://www.netlib.org/blas/zher2.f

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

Error Status

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if n &lt; 0, incx == 0, or incy == 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function cublasZhpmv()

```c
void cublasZhpmv (char uplo, int n, cuDoubleComplex alpha,
                 const cuDoubleComplex *AP,
                 const cuDoubleComplex *x, int incx,
                 cuDoubleComplex beta,
                 cuDoubleComplex *y, int incy)
```

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.

Input

- \( \text{uplo} \) specifies whether the matrix data is stored in the upper or the lower triangular part of array \( \text{AP} \). If \( \text{uplo} == \text{'U'} \) or \( \text{'u'} \), the upper triangular part of \( A \) is supplied in \( \text{AP} \). If \( \text{uplo} == \text{'L'} \) or \( \text{'l'} \), the lower triangular part of \( A \) is supplied in \( \text{AP} \).
- \( n \) 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)

\( \text{AP} \)  
double-precision complex array with at least \((n \times (n+1))/2\) elements.  
If uplo == 'U' or 'u', array \( \text{AP} \) contains the upper triangular part of the 
Hermitian matrix \( A \), packed sequentially, column by column; that is, if \( i \leq j \), \( A[i,j] \) is stored in \( \text{AP}[i+(j \times (j+1))/2] \). If uplo == 'L' or 'l', the array \( \text{AP} \) contains the lower triangular part of the 
Hermitian matrix \( A \), packed sequentially, column by column; that is, if \( i \geq j \), \( A[i,j] \) is stored in \( \text{AP}[i+((2 \times n-j+1) \times j)/2] \). 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 \( (1 + (n-1) \times \text{abs}(\text{incx})). \)

\( \text{incx} \)  
storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.

\( \text{beta} \)  
double-precision scalar multiplier applied to vector \( y \).

\( y \)  
double-precision array of length at least \( (1 + (n-1) \times \text{abs}(\text{incy})) \).  
If \( \text{beta} \) is zero, \( y \) is not read.

\( \text{incy} \)  
storage spacing between elements of \( y \); \( \text{incy} \) must not be zero.

Output

\( y \)  
updated according to \( y = \text{alpha} \times A \times x + \text{beta} \times y \).

Reference: [http://www.netlib.org/blas/zhpmv.f](http://www.netlib.org/blas/zhpmv.f)

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

Error Status

- **CUBLAS_STATUS_NOT_INITIALIZED**  
  if CUBLAS library was not initialized

- **CUBLAS_STATUS_INVALID_VALUE**  
  if \( n < 0 \), \( \text{incx} == 0 \), or \( \text{incy} == 0 \)

- **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()

```c
void
cublasZhpr (char uplo, int n, double alpha,
            const cuDoubleComplex *x, int incx,
            cuDoubleComplex *AP)
```

performs the Hermitian rank 1 operation

\[
A = \alpha \times x \times x^H + A,
\]

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` is the number of rows and columns of matrix \( A \); \( n \) must be at least zero.
- `alpha` is a double-precision scalar multiplier applied to \( x \times x^H \).
- `x` is a double-precision complex array of length at least \( (1 + (n - 1) \times \text{abs}(\text{incx})) \).
- `incx` is the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- `AP` is a double-precision complex array with at least \( (n \times (n + 1)) / 2 \) 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 \( i \leq j \), \( A[i, j] \) is stored in \( AP[i + (j \times (j + 1) / 2)] \). 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 \( i \geq j \), \( A[i, j] \) is stored in \( AP[i + ((2 \times n - j + 1) \times j) / 2] \). 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` is updated according to \( A = \alpha \times x \times x^H + A \).

**Reference:** [http://www.netlib.org/blas/zhpr.f](http://www.netlib.org/blas/zhpr.f)
Error status for this function can be retrieved via `cublasGetError()`.  

**Error Status**  

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if n &lt; 0 or incx == 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

**Function cublasZhpr2()**  

```c
void cublasZhpr2 (char uplo, int n, cuDoubleComplex alpha,
          const cuDoubleComplex *x, int incx,
          const cuDoubleComplex *y, int incy,
          cuDoubleComplex *AP)
```

performs the Hermitian rank 2 operation  

$$
A = \alpha x \cdot y^H + \alpha y \cdot x^H + A,
$$

where $\alpha$ is a double-precision complex scalar, $x$ and $y$ are $n$-element 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**  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uplo</td>
<td>specifies whether the matrix data is stored in the upper or the lower triangular part of array $A$. If <code>uplo</code> == 'U' or 'u', only the upper triangular part of $A$ may be referenced and the lower triangular part of $A$ is inferred. If <code>uplo</code> == 'L' or 'l', only the lower triangular part of $A$ may be referenced and the upper triangular part of $A$ is inferred.</td>
</tr>
<tr>
<td>$n$</td>
<td>the number of rows and columns of matrix $A$; $n$ must be at least zero.</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>double-precision complex scalar multiplier applied to $x \cdot y^H$ and whose conjugate is applied to $y \cdot x^H$.</td>
</tr>
<tr>
<td>$x$</td>
<td>double-precision complex array of length at least $(1 + (n-1) \cdot \text{abs}(\text{incx}))$.</td>
</tr>
<tr>
<td>incx</td>
<td>the storage spacing between elements of $x$; incx must not be zero.</td>
</tr>
<tr>
<td>$y$</td>
<td>double-precision complex array of length at least $(1 + (n-1) \cdot \text{abs}(\text{incy}))$.</td>
</tr>
</tbody>
</table>
CHAPTER 4

Double-Precision BLAS2 Functions

CHAPTER 4

Double-Precision BLAS2 Functions

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()

```c
void

cublasZtbmv (char uplo, char trans, char diag, int n,
int k, const cuDoubleComplex *A, int lda,
cuDoubleComplex *x, int incx)
```

performs one of the matrix-vector operations

\[
x = \text{op}(A) \cdot x,
\]

where \( \text{op}(A) = A, \text{op}(A) = A^T, \text{or } \text{op}(A) = A^H; \)
x is an n-element double-precision complex vector, and A is an n×n, unit or non-unit, upper or lower, triangular 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) = Aᵀ. If trans == 'C', or 'c', op(A) = Aᴴ.
- **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.
- **A** 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 matrix 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; lda must be at least k+1.
- **x** double-precision complex array of length at least 
  \((1 + (n-1) \times \text{abs}(incx)).\) 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 \(x = \text{op}(A) \times x\).
CHAPTER 4  Double-Precision BLAS2 Functions

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

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>If CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>If incx == 0, k &lt; 0, or n &lt; 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ALLOC_FAILED</td>
<td>If function cannot allocate enough</td>
</tr>
<tr>
<td></td>
<td>internal scratch vector memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>If function invoked on device that</td>
</tr>
<tr>
<td></td>
<td>does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>If function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function cublasZtbsv()

void
cublasZtbsv (char uplo, char trans, char diag, int n, int k, const cuDoubleComplex *A, int lda, cuDoubleComplex *X, int incx)
solves one of the systems of equations

\[
\text{op}(A) * x = b,
\]

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

\( b \) 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.

trans specifies \( \text{op}(A) \). If trans == 'N' or 'n', \( \text{op}(A) = A \).
If trans == 'T' or 't', \( \text{op}(A) = A^T\).
If trans == 'C', or 'c', \( \text{op}(A) = A^H \).
Input (continued)

\text{diag} \quad \text{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.}

\text{n} \quad \text{the number of rows and columns of matrix A; n must be at least zero.}

\text{k} \quad \text{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.}

\text{A} \quad \text{double-precision complex array of dimension (lda, n). If uplo == 'U' or 'u', the leading (k+1)xn 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)xn 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 kxk triangle of the array is not referenced.}

\text{x} \quad \text{double-precision complex array of length at least (1 + (n-1) * \text{abs} (\text{incx})).}

\text{incx} \quad \text{storage spacing between elements of x; incx must not be zero.}

Output

\text{x} \quad \text{updated to contain the solution vector x that solves } \text{op}(A) \times x = b.

Reference: \text{http://www.netlib.org/blas/ztbsv.f}

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

\text{CUBLAS_STATUS_NOT_INITIALIZED} \quad \text{if CUBLAS library was not initialized}

\text{CUBLAS_STATUS_INVALID_VALUE} \quad \text{if incx == 0 or } n < 0

\text{CUBLAS_STATUS_ARCH_MISMATCH} \quad \text{if function invoked on device that does not support double precision}

\text{CUBLAS_STATUS_EXECUTION_FAILED} \quad \text{if function failed to launch on GPU}
CHAPTER 4 Double-Precision BLAS2 Functions

Function cublasZtpmv()

```c
void
cublasZtpmv (char uplo, char trans, char diag, int n,
const cuDoubleComplex *AP,
cuDoubleComplex *x, int incx)
```

performs one of the matrix-vector operations

\[
x = \text{op}(A) \times x,
\]

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

\( x \) 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.

**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 \( \text{op}(A) \). If `trans == 'N'` or `'n'`, \( \text{op}(A) = A \).
  - If `trans == 'T'` or `'t'`, \( \text{op}(A) = A^T \).
  - If `trans == 'C'`, or `'c'`, \( \text{op}(A) = A^H \).

- **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 \( A \); \( n \) must be at least zero.

- **AP** 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 \( i \leq j \), \( A[i, j] \) 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 \( i \geq j \), \( A[i, j] \) is stored in \( AP[i + ((2 * n - j + 1) * j) / 2] \).

- **x** double-precision complex array of length at least \( (1 + (n - 1) * \text{abs}(incx)) \). 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.
CUDA

CUBLAS Library

Output

\[ x \] updated according to \[ x = \text{op}(A) \times x. \]

Reference: [http://www.netlib.org/blas/ztpmv.f](http://www.netlib.org/blas/ztpmv.f)

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if <code>incx == 0</code> or <code>n &lt; 0</code></td>
</tr>
<tr>
<td>CUBLAS_STATUS_ALLOC_FAILED</td>
<td>if function cannot allocate enough internal scratch vector memory</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function `cublasZtpsv()`

```c
void
cublasZtpsv (char uplo, char trans, char diag, int n,
             const cuDoubleComplex *AP,
             cuDoubleComplex *X, int incx)
```

solves one of the systems of equations

\[ \text{op}(A) \times x = b, \]

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

\( b \) 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

- `uplo` 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 \( \text{op}(A) \). If `trans == 'N'` or `n'`, \( \text{op}(A) = A \).
  If `trans == 'T'` or `t'`, \( \text{op}(A) = A^T \).
  If `trans == 'C'`, or `c'`, \( \text{op}(A) = A^H \).
CHAPTER 4  Double-Precision BLAS2 Functions

Input (continued)

- **diag** specifies whether \( A \) is unit triangular. If \( \text{diag} = \text{‘U’} \) or \( \text{‘u’} \), \( A \) is assumed to be unit triangular; that is, diagonal elements are not read and are assumed to be unity. If \( \text{diag} = \text{‘N’} \) or \( \text{‘n’} \), \( A \) 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.

- **AP** double-precision complex array with at least \((n * (n + 1)) / 2 \) elements. If \( \text{uplo} = \text{‘U’} \) or \( \text{‘u’} \), 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 \( \text{uplo} = \text{‘L’} \) or \( \text{‘l’} \), 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 \( \text{diag} = \text{‘U’} \) or \( \text{‘u’} \), the diagonal elements of \( A \) are not referenced and are assumed to be unity.

- **x** double-precision complex array of length at least \((1 + (n - 1) * \text{abs} (\text{incx})) \).

- **incx** storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.

Output

- **x** updated to contain the solution vector \( x \) that solves \( \text{op}(A) \cdot x = b \).

Reference: [http://www.netlib.org/blas/ztpsv.f](http://www.netlib.org/blas/ztpsv.f)

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

**Error Status**

- **CUBLAS_STATUS_NOT_INITIALIZED** if CUBLAS library was not initialized
- **CUBLAS_STATUS_INVALID_VALUE** if \( \text{incx} = 0 \) 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
Function cublasZtrmv()

```c
void
cublasZtrmv (char uplo, char trans, char diag, int n,
            const cuDoubleComplex *A, int lda,
            cuDoubleComplex *x, int incx)
```

performs one of the matrix-vector operations

$$x = \text{op}(A) \times x,$$

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

$x$ 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.

**Input**

- **uplo** specifies whether the matrix $A$ is an upper or lower triangular matrix. If $\text{uplo} == 'U' \text{ or } 'u'$, $A$ is an upper triangular matrix. If $\text{uplo} == 'L' \text{ or } 'l'$, $A$ is an lower triangular matrix.
- **trans** specifies $\text{op}(A)$. If $\text{trans} == 'N' \text{ or } 'n'$, $\text{op}(A) = A$. If $\text{trans} == 'T' \text{ or } 't'$, $\text{op}(A) = A^T$. If $\text{trans} == 'C' \text{ or } 'c'$, $\text{op}(A) = A^H$.
- **diag** specifies whether or not $A$ is a unit triangular matrix. If $\text{diag} == 'U' \text{ or } 'u'$, $A$ is assumed to be unit triangular. If $\text{diag} == 'N' \text{ or } 'n'$, $A$ 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.
- **A** double-precision complex array of dimensions $(\text{lda}, n)$. If $\text{uplo} == 'U' \text{ or } 'u'$, the leading $n \times n$ 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 $\text{uplo} == 'L' \text{ or } 'l'$, the leading $n \times n$ 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 $\text{diag} == 'U' \text{ or } 'u'$, the diagonal elements of $A$ are not referenced either, but are assumed to be unity.
- **lda** leading dimension of $A$; $\text{lda}$ must be at least $\max(1, n)$. 
CHAPTER 4  Double-Precision BLAS2 Functions

Input (continued)

\( x \)  
- double-precision complex array of length at least 
  \((1 + (n-1) \times \text{abs(incx)})\). On entry, \( x \) contains the source vector.
  On exit, \( x \) is overwritten with the result vector.

\( \text{incx} \)  
- the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.

Output

\( x \)  
- updated according to \( x = \text{op}(A) \times x \).

Reference:  \text{http://www.netlib.org/blas/ztrmv.f}

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

Error Status

\begin{itemize}
  \item \text{CUBLAS_STATUS_NOT_INITIALIZED} if CUBLAS library was not initialized
  \item \text{CUBLAS_STATUS_INVALID_VALUE} if \( \text{incx} = 0 \) or \( n < 0 \)
  \item \text{CUBLAS_STATUS_EXECUTION_FAILED} if function failed to launch on GPU
  \item \text{CUBLAS_STATUS_ALLOC_FAILED} if function cannot allocate enough
    internal scratch vector memory
\end{itemize}

Function \text{cublasZtrsv}()

\begin{verbatim}
void
    cublasZtrsv (char uplo, char trans, char diag, int n,
              const cuDoubleComplex *A, int lda,
              cuDoubleComplex *x, int incx)
\end{verbatim}

solves a system of equations

\[
\text{op}(A) \times x = b,
\]

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

\( b \) 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 \( \text{lda} \) 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

**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 == 'L' or 'l', 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.

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 \( (1 + (n - 1) \times \text{abs}(\text{incx})) \). 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

x updated to contain the solution vector x that solves \( \text{op}(A) \times x = b \).

Reference: [http://www.netlib.org/blas/ztrsv.f](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
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 186
- “Single-Precision Complex BLAS3 Functions” on page 199
- “Double-Precision BLAS3 Functions” on page 218
- “Double-Precision Complex BLAS3 Functions” on page 231
Single-Precision BLAS3 Functions

The single-precision BLAS3 functions are listed below:

- “Function cublasSgemm()” on page 187
- “Function cublasSsymm()” on page 188
- “Function cublasSsyrk()” on page 190
- “Function cublasSsyr2k()” on page 192
- “Function cublasStrmm()” on page 194
- “Function cublasStrsm()” on page 196
Function cublasSgemm()

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

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:

\[ C = \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C, \]
\[ \text{where } \text{op}(X) = X \text{ or } \text{op}(X) = X^T, \]

and alpha and beta are single-precision scalars. A, B, and C are matrices consisting of single-precision elements, with \( \text{op}(A) \) an \( m \times k \) matrix, \( \text{op}(B) \) a \( k \times n \) matrix, and \( C \) an \( m \times n \) matrix. Matrices A, B, and C are stored in column-major format, and \( \text{lda}, \text{ldb}, \text{ldc} \) are the leading dimensions of the two-dimensional arrays containing A, B, and C.

**Input**

- `transa` specifies \( \text{op}(A) \). If \( \text{transa} = 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  - If \( \text{transa} = 'T', 't', 'C', \) or \( 'c' \), \( \text{op}(A) = A^T \).
- `transb` specifies \( \text{op}(B) \). If \( \text{transb} = 'N' \) or \( 'n' \), \( \text{op}(B) = B \).
  - If \( \text{transb} = 'T', 't', 'C', \) or \( 'c' \), \( \text{op}(B) = B^T \).
- `m` number of rows of matrix \( \text{op}(A) \) and rows of matrix C; \( m \) must be at least zero.
- `n` number of columns of matrix \( \text{op}(B) \) and number of columns of C;
  \( n \) must be at least zero.
- `k` number of columns of matrix \( \text{op}(A) \) and number of rows of \( \text{op}(B) \);
  \( k \) must be at least zero.
- `alpha` single-precision scalar multiplier applied to \( \text{op}(A) \cdot \text{op}(B) \).
- `A` single-precision array of dimensions \( (\text{lda}, k) \) if \( \text{transa} = 'N' \) or \( 'n' \), and of dimensions \( (\text{lda}, m) \) otherwise. If \( \text{transa} = 'N' \) or \( 'n' \), \( \text{lda} \) must be at least \( \text{max}(1, m) \); otherwise, \( \text{lda} \) must be at least \( \text{max}(1, k) \).
- `lda` leading dimension of two-dimensional array used to store matrix A.
CUDA

CUBLAS Library

Input (continued)

- **B**
  - single-precision array of dimensions \((\text{ldb}, n)\) if \(\text{transb} = 'N'\) or \('n'\), and of dimensions \((\text{ldb}, k)\) otherwise. If \(\text{transb} = 'N'\) or \('n'\), \(\text{ldb}\) must be at least \(\max(1, k)\); otherwise, \(\text{ldb}\) 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.

- **C**
  - single-precision array of dimensions \((\text{ldc}, n)\); \(\text{ldc}\) must be at least \(\max(1, m)\).

- **ldc**
  - leading dimension of two-dimensional array used to store matrix \(C\).

Output

- **C**
  - updated based on \(C = \alpha \times \text{op}(A) \times \text{op}(B) + \beta \times C\).

Reference: [http://www.netlib.org/blas/sgemm.f](http://www.netlib.org/blas/sgemm.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_EXECUTION_FAILED** if function failed to launch on GPU

**Function cublasSsymm()**

```c
void

cublasSsymm (char side, char uplo, int m, int n,
             float alpha, const float *A, int lda,
             const float *B, int ldb, float beta,
             float *C, int ldc)
```

performs one of the matrix-matrix operations

\[
C = \alpha \times A \times B + \beta \times C \quad \text{or} \quad C = \alpha \times B \times A + \beta \times C,
\]

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 $m \times n$ matrices consisting of single-precision elements.

**Input**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| **side** | specifies whether the symmetric matrix $A$ appears on the left-hand side or right-hand side of matrix $B$.  
If $\text{side} = \text{'L' or 'l'}$, $C = \alpha A \times B + \beta C$.  
If $\text{side} = \text{'R' or 'r'}$, $C = \alpha B \times A + \beta C$. |
| **uplo** | specifies whether the symmetric matrix $A$ is stored in upper or lower storage mode. If $\text{uplo} = \text{'U' or 'u'}$, 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 $\text{uplo} = \text{'L' or 'l'}$, 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. |
| **m** | specifies the number of rows of matrix $C$, and the number of rows of matrix $B$. It also specifies the dimensions of symmetric matrix $A$ when $\text{side} = \text{'L' or 'l'}$; $m$ 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 $\text{side} = \text{'R' or 'r'}$; $n$ must be at least zero. |
| **alpha** | single-precision scalar multiplier applied to $A \times B$ or $B \times A$. |
| **A** | single-precision array of dimensions ($\text{lda}, \text{ka}$), where $\text{ka}$ is $m$ when $\text{side} = \text{'L' or 'l'}$ and is $n$ otherwise. If $\text{side} = \text{'L' or 'l'}$, the leading $m \times m$ part of array $A$ must contain the symmetric matrix such that when $\text{uplo} = \text{'U' or 'u'}$, the leading $m \times m$ part stores the upper triangular part of the symmetric matrix, and the strictly lower triangular part of $A$ is not referenced; and when $\text{uplo} = \text{'L' or 'l'}$, the leading $m \times m$ part stores the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.  
If $\text{side} = \text{'R' or 'r'}$, the leading $n \times n$ part of array $A$ must contain the symmetric matrix such that when $\text{uplo} = \text{'U' or 'u'}$, 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 $\text{uplo} = \text{'L' or 'l'}$, 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 $\text{side} = \text{'L' or 'l'}$, it must be at least $\max(1, m)$ and at least $\max(1, n)$ otherwise. |
Input (continued)

| B     | 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 \(B\); \(ldb\) 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. |
| C     | single-precision array of dimensions \((ldc, n)\). |
| ldc   | leading dimension of \(C\); \(ldc\) must be at least \(\max(1, m)\). |

Output

\(C\) updated according to \(C = alpha \times A \times B + beta \times C\) or \(C = alpha \times B \times A + beta \times C\).

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

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if (m &lt; 0) or (n &lt; 0)</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function `cublasSsyrk()`

```c
void

cublasSsyrk (char uplo, char trans, int n, int k,
            float alpha, const float *A, int lda,
            float beta, float *C, int ldc)
```

performs one of the symmetric rank \(k\) operations

\[ C = alpha \times A \times A^T + beta \times C \quad \text{or} \quad C = alpha \times A^T \times A + beta \times C, \]

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** specifies whether the symmetric matrix $C$ is stored in upper or lower storage mode. If $\text{uplo} = 'U'$ 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 $\text{uplo} = 'L'$ 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 $\text{trans} = 'N'$ or 'n', $C = \alpha A A^T + \beta C$. If $\text{trans} = 'T'$, 't', 'C', or 'c', $C = \alpha A^T A + \beta C$.

**n** specifies the number of rows and the number columns of matrix $C$. If $\text{trans} = 'N'$ or 'n', $n$ specifies the number of rows of matrix $A$. If $\text{trans} = 'T'$, 't', 'C', or 'c', $n$ specifies the number of columns of matrix $A$; $n$ must be at least zero.

**k** If $\text{trans} = 'N'$ or 'n', $k$ specifies the number of columns of matrix $A$. If $\text{trans} = 'T'$, '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 $A + A^T$ or $A^T + A$.

**A** single-precision array of dimensions $(\text{lda}, \text{ka})$, where $\text{ka}$ is $k$ when $\text{trans} = 'N'$ or 'n' and is $n$ otherwise. When $\text{trans} = 'N'$ 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 $\text{trans} = 'N'$ or 'n', $\text{lda}$ must be at least $\max(1, n)$. Otherwise $\text{lda}$ 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)

| C     | 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

| C     | 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](http://www.netlib.org/blas/ssyrk.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_EXECUTION_FAILED | if function failed to launch on GPU |

Function cublasSsyr2k()

```c
void

void cublasSsyr2k (char uplo, char trans, int n, int k,
    float alpha, const float *A, int lda,
    const float *B, int ldb, float beta,
    float *C, int ldc)
```

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×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 \( n \times k \) 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 \( \text{uplo} = \text{'U'} \) or \( \text{'u'} \), 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 \( \text{uplo} = \text{'L'} \) or \( \text{'l'} \), 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 \( \text{trans} = \text{'N'} \) or \( \text{'n'} \), \( C = \text{alpha} \times A \times B^T + \text{alpha} \times B \times A^T + \text{beta} \times C \). If \( \text{trans} = \text{'T'} \), \( \text{'t'} \), \( \text{'C'} \), or \( \text{'c'} \), \( C = \text{alpha} \times A^T \times B + \text{alpha} \times B^T \times A + \text{beta} \times C \).

- **n** specifies the number of rows and the number columns of matrix \( C \). If \( \text{trans} = \text{'N'} \) or \( \text{'n'} \), \( n \) specifies the number of rows of matrix \( A \). If \( \text{trans} = \text{'T'} \), \( \text{'t'} \), \( \text{'C'} \), or \( \text{'c'} \), \( n \) specifies the number of columns of matrix \( A \); \( n \) must be at least zero.

- **k** If \( \text{trans} = \text{'N'} \) or \( \text{'n'} \), \( k \) specifies the number of columns of matrix \( A \). If \( \text{trans} = \text{'T'} \), \( \text{'t'} \), \( \text{'C'} \), or \( \text{'c'} \), \( k \) specifies the number of rows of matrix \( A \); \( k \) must be at least zero.

- **alpha** single-precision scalar multiplier.

- **A** single-precision array of dimensions \((\text{lda}, \text{ka})\), where \( \text{ka} \) is \( k \) when \( \text{trans} = \text{'N'} \) or \( \text{'n'} \) and is \( n \) otherwise. When \( \text{trans} = \text{'N'} \) or \( \text{'n'} \), 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 \( \text{trans} = \text{'N'} \) or \( \text{'n'} \), \( \text{lda} \) must be at least \( \max(1, n) \). Otherwise \( \text{lda} \) must be at least \( \max(1, k) \).

- **B** single-precision array of dimensions \((\text{ldb}, \text{kb})\), where \( \text{kb} = k \) when \( \text{trans} = \text{'N'} \) or \( \text{'n'} \), and \( k = n \) otherwise. When \( \text{trans} = \text{'N'} \) or \( \text{'n'} \), 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 \( \text{trans} = \text{'N'} \) or \( \text{'n'} \), \( \text{ldb} \) must be at least \( \max(1, n) \). Otherwise \( \text{ldb} \) must be at least \( \max(1, k) \).

- **beta** single-precision scalar multiplier applied to \( C \). If \( \text{beta} \) is zero, \( C \) does not have to be a valid input.
Function cublasStrmm()

```
void
cublasStrmm (char side, char uplo, char transa,
        char diag, int m, int n, float alpha,
        const float *A, int lda, const float *B,
        int ldb)
```

performs one of the matrix-matrix operations

\[
B = \alpha \cdot \text{op}(A) \cdot B \quad \text{or} \quad B = \alpha \cdot B \cdot \text{op}(A),
\]

where \( \text{op}(A) = A \quad \text{or} \quad \text{op}(A) = A^T \),
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 lda and ldb are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

- \( \text{side} \): specifies whether \( \text{op}(A) \) multiplies B from the left or right.
  - If \( \text{side} == 'L' \) or \( 'l' \), \( B = \text{alpha} \times \text{op}(A) \times B \).
  - If \( \text{side} == 'R' \) or \( 'r' \), \( B = \text{alpha} \times B \times \text{op}(A) \).
- \( \text{uplo} \): specifies whether the matrix A is an upper or lower triangular matrix.
  - If \( \text{uplo} == 'U' \) or \( 'u' \), A is an upper triangular matrix.
  - If \( \text{uplo} == 'L' \) or \( 'l' \), A is a lower triangular matrix.
- \( \text{transa} \): specifies the form of \( \text{op}(A) \) to be used in the matrix multiplication.
  - If \( \text{transa} == 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  - If \( \text{transa} == 'T' \), \( 't' \), \( 'C' \), or \( 'c' \), \( \text{op}(A) = A^T \).
- \( \text{diag} \): specifies whether or not A is a unit triangular matrix. If \( \text{diag} == 'U' \) or \( 'u' \), A is assumed to be unit triangular. If \( \text{diag} == 'N' \) or \( 'n' \), A is not assumed to be unit triangular.
- \( m \): the number of rows of matrix B; \( m \) must be at least zero.
- \( n \): the number of columns of matrix B; \( n \) must be at least zero.
- \( \text{alpha} \): single-precision scalar multiplier applied to \( \text{op}(A) \times B \) or \( B \times \text{op}(A) \), respectively. If \( \text{alpha} \) is zero, no accesses are made to matrix A, and no read accesses are made to matrix B.
- A: single-precision array of dimensions \((\text{lda}, k)\). If \( \text{side} == 'L' \) or \( 'l' \), \( k = m \). If \( \text{side} == 'R' \) or \( 'r' \), \( k = n \). If \( \text{uplo} == 'U' \) or \( 'u' \), the leading \( k \times k \) 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 \( \text{uplo} == 'L' \) or \( 'l' \), the leading \( k \times k \) 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 \( \text{diag} == 'U' \) or \( 'u' \), the diagonal elements of A are not referenced and are assumed to be unity.
- \( \text{lda} \): leading dimension of A. When \( \text{side} == 'L' \) or \( 'l' \), it must be at least \( \max(1, m) \) and at least \( \max(1, n) \) otherwise.
Function cublasStrsm()

void

cublasStrsm (char side, char uplo, char transa,
    char diag, int m, int n, float alpha,
    const float *A, int lda, float *B, int ldb)

solves one of the matrix equations

\[ \text{op}(A) \cdot X = \alpha \cdot B \quad \text{or} \quad X \cdot \text{op}(A) = \alpha \cdot B, \]

where \( \text{op}(A) = A \quad \text{or} \quad \text{op}(A) = A^T, \)

\( \alpha \) 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

side specifies whether \( \text{op}(A) \) appears on the left or right of \( X \):
side == 'L' or 'l' indicates solve \( \text{op}(A) \) \( X = \alpha \) \( B \);
side == 'R' or 'r' indicates solve \( X \) \( \text{op}(A) = \alpha B \).

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 \( \text{op}(A) \) to be used in matrix multiplication.
If transa == 'N' or 'n', \( \text{op}(A) = A \).
If transa == 'T', 't', 'C', or 'c', \( \text{op}(A) = A^T \).

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.

m specifies the number of rows of \( B \); \( m \) must be at least zero.

n specifies the number of columns of \( B \); \( n \) must be at least zero.

alpha single-precision scalar multiplier applied to \( B \). When alpha is zero, \( A \) is not referenced and \( B \) does not have to be a valid input.

A single-precision array of dimensions \((lda, k)\), where \( k \) is \( m \) when
side == 'L' or 'l' and is \( n \) when side == 'R' or 'r'.
If uplo == 'U' or 'u', the leading \( k \times k \) 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
\( k \times k \) 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 'l', \( lda \) must be at least maxi(1, m).
When side == 'R' or 'r', \( lda \) must be at least maxi(1, n).

B single-precision array of dimensions \((ldb, n)\); \( ldb \) must be at least
max(1, m).
The leading \( m \times n \) 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 \( B \); \( ldb \)
must be at least max(1, m).
Output

\[ B \] contains the solution matrix \( X \) satisfying \( \text{op}(A) \ast X = \alpha \ast B \) or
\[ X \ast \text{op}(A) = \alpha \ast B. \]

Reference: [http://www.netlib.org/blas/strsm.f](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 200
- “Function cublasChemm()” on page 201
- “Function cublasCherk()” on page 203
- “Function cublasCher2k()” on page 205
- “Function cublasCsymm()” on page 207
- “Function cublasCsyrk()” on page 209
- “Function cublasCsy2k()” on page 211
- “Function cublasCtmm()” on page 213
- “Function cublasCtrsm()” on page 215
Function cublasCgemm()

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

performs one of the matrix-matrix operations

\[
C = \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C,
\]

where \( \text{op}(X) = X, \text{op}(X) = X^T, \) or \( \text{op}(X) = X^\dagger; \)

and \( \alpha \) and \( \beta \) are single-precision complex scalars. \( A, B, \) and \( C \) are matrices consisting of single-precision complex elements, with \( \text{op}(A) \) a \( m \times k \) matrix, \( \text{op}(B) \) a \( k \times n \) matrix, and \( C \) an \( m \times n \) matrix.

**Input**

- **transa** specifies \( \text{op}(A) \). If \( \text{transa} == 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  - If \( \text{transa} == 'T' \) or \( 't' \), \( \text{op}(A) = A^T \).
  - If \( \text{transa} == 'C' \) or \( 'c' \), \( \text{op}(A) = A^H \).

- **transb** specifies \( \text{op}(B) \). If \( \text{transb} == 'N' \) or \( 'n' \), \( \text{op}(B) = B \).
  - If \( \text{transb} == 'T' \) or \( 't' \), \( \text{op}(B) = B^T \).
  - If \( \text{transb} == 'C' \) or \( 'c' \), \( \text{op}(B) = B^H \).

- **m** number of rows of matrix \( \text{op}(A) \) and rows of matrix \( C \); \( m \) must be at least zero.

- **n** number of columns of matrix \( \text{op}(B) \) and number of columns of \( C \); \( n \) must be at least zero.

- **k** number of columns of matrix \( \text{op}(A) \) and number of rows of \( \text{op}(B) \); \( k \) must be at least zero.

- **alpha** single-precision complex scalar multiplier applied to \( \text{op}(A) \cdot \text{op}(B) \).

- **A** single-precision complex array of dimension \((\text{lda},k)\) if \( \text{transa} == 'N' \) or \( 'n' \), and of dimension \((\text{lda},m)\) otherwise.

- **lda** leading dimension of \( A \). When \( \text{transa} == 'N' \) or \( 'n' \), it must be at least \( \max(1,m) \) and at least \( \max(1,k) \) otherwise.

- **B** single-precision complex array of dimension \((\text{ldb},n)\) if \( \text{transb} == 'N' \) or \( 'n' \), and of dimension \((\text{ldb},k)\) otherwise.

- **ldb** leading dimension of \( B \). When \( \text{transb} == 'N' \) or \( 'n' \), it must be at least \( \max(1,k) \) and at least \( \max(1,n) \) otherwise.
CHAPTER 5

BLAS3 Functions

Reference:
http://www.netlib.org/blas/cgemm.f
Error status for this function can be retrieved via cublasGetError().

Function cublasChemm()

void
cublasChemm (char side, char uplo, int m, int n,
cuComplex alpha, const cuComplex *A,
    int lda, const cuComplex *B, int ldb,
cuComplex beta, cuComplex *C, int ldc)

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×n
matrices consisting of single-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 \( \text{side} == 'L' \) or \( 'l' \), \( C = \alpha A B + \beta C \).
- If \( \text{side} == 'R' \) or \( 'r' \), \( C = \alpha B A + \beta C \).

uplo specifies whether the Hermitian matrix \( A \) is stored in upper or lower storage mode. If \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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.

\( m \) specifies the number of rows of matrix \( C \), and the number of rows of matrix \( B \). It also specifies the dimensions of Hermitian matrix \( A \) when \( \text{side} == 'L' \) 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 Hermitian matrix \( A \) when \( \text{side} == 'R' \) or \( 'r' \); \( n \) must be at least zero.

\( \alpha \) single-precision complex scalar multiplier applied to \( A B \) or \( B A \).

\( A \) single-precision complex array of dimensions \((\text{lda}, \text{ka})\), where \( \text{ka} \) is \( m \) when \( \text{side} == 'L' \) or \( 'l' \) and is \( n \) otherwise. If \( \text{side} == 'L' \) or \( 'l' \), the leading \( m \times m \) part of array \( A \) must contain the Hermitian matrix such that when \( \text{uplo} == 'U' \) or \( 'u' \), the leading \( m \times m \) part stores the upper triangular part of the Hermitian matrix, and the strictly lower triangular part of \( A \) is not referenced; and when \( \text{uplo} == 'L' \) or \( 'l' \), the leading \( m \times m \) part stores the lower triangular part of the Hermitian matrix, and the strictly upper triangular part is not referenced. If \( \text{side} == 'R' \) or \( 'r' \), the leading \( n \times n \) part of array \( A \) must contain the Hermitian matrix such that when \( \text{uplo} == 'U' \) or \( 'u' \), the leading \( n \times n \) part stores the upper triangular part of the Hermitian matrix, and the strictly lower triangular part of \( A \) is not referenced; and when \( \text{uplo} == 'L' \) 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.

\( \text{lda} \) leading dimension of \( A \). When \( \text{side} == 'L' \) or \( 'l' \), it must be at least \( \max(1, m) \) and at least \( \max(1, n) \) otherwise.
Input (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>single-precision complex array of dimensions $(\text{ldb}, n)$. On entry, the leading $m \times n$ part of the array contains the matrix $B$.</td>
</tr>
<tr>
<td>ldb</td>
<td>leading dimension of $B$; $\text{ldb}$ must be at least $\max(1, m)$.</td>
</tr>
<tr>
<td>beta</td>
<td>single-precision complex scalar multiplier applied to $C$. If $\beta$ is zero, $C$ does not have to be a valid input.</td>
</tr>
<tr>
<td>C</td>
<td>single-precision complex array of dimensions $(\text{ldc}, n)$.</td>
</tr>
<tr>
<td>ldc</td>
<td>leading dimension of $C$; $\text{ldc}$ must be at least $\max(1, m)$.</td>
</tr>
</tbody>
</table>

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](http://www.netlib.org/blas/chemm.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 cublasCherk()**

```c
void

cublasCherk (char uplo, char trans, int n, int k,
             float alpha, const cuComplex *A,
             int lda, float beta, cuComplex *C,
             int ldc)
```

performs one of the Hermitian rank $k$ operations

$$
C = \alpha A A^H + \beta C \quad \text{or} \quad C = \alpha A^H A + \beta C,
$$

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 \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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 \( \text{trans} == 'N' \) or \( 'n' \), \( C = \alpha \cdot A \cdot A^H + \beta \cdot C \). If \( \text{trans} == 'T', 't', 'C', \) or \( 'c' \), \( C = \alpha \cdot A^H \cdot A + \beta \cdot C \).

- **n** specifies the number of rows and the number columns of matrix \( C \). If \( \text{trans} == 'N' \) or \( 'n' \), \( n \) specifies the number of rows of matrix \( A \). If \( \text{trans} == 'T', 't', 'C', \) or \( 'c' \), \( n \) specifies the number of columns of matrix \( A \); \( n \) must be at least zero.

- **k** If \( \text{trans} == 'N' \) or \( 'n' \), \( k \) specifies the number of columns of matrix \( A \). If \( \text{trans} == 'T', 't', 'C', \) 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 \((\text{lda}, \text{ka})\), where \( \text{ka} \) is \( k \) when \( \text{trans} == 'N' \) or \( 'n' \) and is \( n \) otherwise. When \( \text{trans} == 'N' \) 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 \( \text{trans} == 'N' \) or \( 'n' \), \( \text{lda} \) must be at least \( \max(1, n) \). Otherwise \( \text{lda} \) must be at least \( \max(1, k) \).

- **beta** single-precision real scalar multiplier applied to \( C \). If \( \beta \) is zero, \( C \) does not have to be a valid input.
CHAPTER 5 BLAS3 Functions

Reference: http://www.netlib.org/blas/cherk.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_EXECUTION_FAILED` if function failed to launch on GPU

**Function cublas Cher2k()**

```c
void

void cublasCher2k (char uplo, char trans, int n, int k,
        cuComplex alpha, const cuComplex *A,
        int lda, const cuComplex *B, int ldb,
        float beta, cuComplex *C, int ldc)
```

performs one of the Hermitian rank \( 2k \) operations

\[
C = \alpha A^H B + \alpha B^H A + \beta C \quad \text{or} \\
C = \alpha A^H B + \alpha B^H A + \beta C
\]

\( \alpha, \beta \in \mathbb{C} \) are complex scalars, \( A, B \in \mathbb{C}^{n \times n} \) are \( \alpha \times \beta \) complex matrices, and \( C \in \mathbb{C}^{n \times n} \) is the output complex matrix.
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**

- \( \text{uplo} \) specifies whether the Hermitian matrix \( C \) is stored in upper or lower storage mode. If \( \text{uplo} == \'U' \) 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 \( \text{uplo} == \'L' \) 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.

- \( \text{trans} \) specifies the operation to be performed. If \( \text{trans} == \'N' \) or \'n'\, \( C = \alpha \cdot A \cdot B^H + \alpha \cdot B \cdot A^H + \beta \cdot C \). If \( \text{trans} == \'T' \), \'t'\, \'C'\, or \'c'\, \( C = \alpha \cdot A^H \cdot B + \alpha \cdot B^H \cdot A + \beta \cdot C \).

- \( n \) specifies the number of rows and the number columns of matrix \( C \). If \( \text{trans} == \'N' \) or \'n'\, \( n \) specifies the number of rows of matrices \( A \) and \( B \). If \( \text{trans} == \'T' \), \'t'\, \'C'\, or \'c'\, \( n \) specifies the number of columns of matrices \( A \) and \( B \); \( n \) must be at least zero.

- \( k \) If \( \text{trans} == \'N' \) or \'n'\, \( k \) specifies the number of columns of matrices \( A \) and \( B \). If \( \text{trans} == \'T' \), \'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.

- \( A \) single-precision complex array of dimensions \( (\text{lda}, \text{ka}) \), where \( \text{ka} \) is \( k \) when \( \text{trans} == \'N' \) or \'n'\ and is \( n \) otherwise. When \( \text{trans} == \'N' \) 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 \( A \).

- \( \text{lda} \) leading dimension of \( A \). When \( \text{trans} == \'N' \) or \'n'\, \( \text{lda} \) must be at least \( \max (1, n) \). Otherwise \( \text{lda} \) must be at least \( \max (1, k) \).

- \( B \) single-precision complex array of dimensions \( (\text{ldb}, \text{kb}) \), where \( \text{kb} \) is \( k \) when \( \text{trans} == \'N' \) or \'n'\ and is \( n \) otherwise. When \( \text{trans} == \'N' \) 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 \( B \).

- \( \text{ldb} \) leading dimension of \( B \). When \( \text{trans} == \'N' \) or \'n'\, \( \text{ldb} \) must be at least \( \max (1, n) \). Otherwise \( \text{ldb} \) must be at least \( \max (1, k) \).
CHAPTER 5 BLAS3 Functions

Input (continued)

beta  single-precision real 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). 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

C  updated according to C = alpha * A * B_H + alpha * B * A_H + beta * C
or C = alpha * A_H * B + 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

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if n &lt; 0 or k &lt; 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function cublasCsymm()

```c
void
cublasCsymm (char side, char uplo, int m, int n,
cuComplex alpha, const cuComplex *A,
int lda, const cuComplex *B, int ldb,
cuComplex beta, cuComplex *C, int ldc)
```

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 $\text{side} == 'L'$ or 'l', $C = \alpha A * B + \beta C$.
  
  If $\text{side} == 'R'$ or 'r', $C = \alpha B * A + \beta C$.

- **uplo** specifies whether the symmetric matrix $A$ is stored in upper or lower storage mode. If $\text{uplo} == 'U'$ 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 $\text{uplo} == 'L'$ 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.

- **m** specifies the number of rows of matrix $C$, and the number of rows of matrix $B$. It also specifies the dimensions of symmetric matrix $A$ when $\text{side} == 'L'$ 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 $\text{side} == 'R'$ or 'r'; $n$ must be at least zero.

- **alpha** single-precision complex scalar multiplier applied to $A * B$ or $B * A$.

- **A** single-precision complex array of dimensions $(l da, k a)$, where $k a$ is $m$ when $\text{side} == 'L'$ or 'l' and is $n$ otherwise. If $\text{side} == 'L'$ or 'l', the leading $m \times m$ part of array $A$ must contain the symmetric matrix such that when $\text{uplo} == 'U'$ or 'u', the leading $m \times m$ part stores the upper triangular part of the symmetric matrix, and the strictly lower triangular part of $A$ is not referenced; and when $\text{uplo} == 'L'$ or 'l', the leading $m \times m$ part stores the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced. If $\text{side} == 'R'$ or 'r', the leading $n \times n$ part of array $A$ must contain the symmetric matrix such that when $\text{uplo} == 'U'$ 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 $\text{uplo} == 'L'$ 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.
CHAPTER 5

BLAS3 Functions

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.

B single-precision complex array of dimensions (ldb, n). On entry, the leading m×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 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.

Error Status

<table>
<thead>
<tr>
<th>CUBLAS_STATUS_NOT_INITIALIZED</th>
<th>if CUBLAS library was not initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if m &lt; 0 or n &lt; 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

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

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

Function cublasCsyrk()

void
cublasCsyrk (char uplo, char trans, int n, int k, cuComplex alpha, const cuComplex *A, int lda, cuComplex beta, cuComplex *C, int ldc)

performs one of the symmetric rank k operations

\[ C = alpha \times A \times A^T + beta \times C \text{ or } C = alpha \times A^T \times A + beta \times C, \]

where alpha and beta are single-precision complex scalars. C is an n×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×k in the first case and k×n in the second case.
**Input**

*uplo* specifies whether the symmetric matrix $C$ is stored in upper or lower storage mode. If $\text{uplo} == \text{'U'}$ or \text{'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 $\text{uplo} == \text{'L'}$ or \text{'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 $\text{trans} == \text{'N'}$ or \text{'n'}, $C = \text{alpha} * A * A^T + \text{beta} * C$. If $\text{trans} == \text{'T'}$, \text{'t'}, \text{'C'}, \text{or 'c'}, C = \text{alpha} * A^T * A + \text{beta} * C$.

*n* specifies the number of rows and the number columns of matrix $C$. If $\text{trans} == \text{'N'}$ or \text{'n'}, $n$ specifies the number of rows of matrix $A$. If $\text{trans} == \text{'T'}$, \text{'t'}, \text{'C'}, \text{or 'c'}, n$ specifies the number of columns of matrix $A$; $n$ must be at least zero.

*k* If $\text{trans} == \text{'N'}$ or \text{'n'}, $k$ specifies the number of columns of matrix $A$. If $\text{trans} == \text{'T'}$, \text{'t'}, \text{'C'}, \text{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 $(\text{lda}, \text{ka})$, where $\text{ka}$ is $k$ when $\text{trans} == \text{'N'}$ or \text{'n'} and is $n$ otherwise. When $\text{trans} == \text{'N'}$ or \text{'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 $\text{trans} == \text{'N'}$ or \text{'n'}, $\text{lda}$ must be at least $\text{max}(1,n)$. Otherwise $\text{lda}$ must be at least $\text{max}(1,k)$.

*beta* single-precision complex scalar multiplier applied to $C$. If $\text{beta}$ is zero, $C$ is not read.
CHAPTER 5  BLAS3 Functions

Input (continued)

C is 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 must be at least max(1, n).

Output

C updated according to $C = \alpha A^T A + \beta C$ or $C = \alpha A^T A + \beta C$.

Reference: http://www.netlib.org/blas/csymr.k

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_EXECUTION_FAILED if function failed to launch on GPU

Function cublasCsr2k()

```c
void

cublasCsr2k (char uplo, char trans, int n, int k,
    cuComplex alpha, const cuComplex *A,
    int lda, const cuComplex *B, int ldb,
    cuComplex beta, cuComplex *C, int ldc)
```

performs one of the symmetric rank $2k$ operations

$C = \alpha A^T A + \alpha B^T B + \beta C$ or $C = \alpha A^T A + \beta C$,

where alpha and beta are single-precision complex scalars. C is an n×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 $\text{uplo} == 'U' \text{ or } 'u'$, 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 $\text{uplo} == 'L' \text{ or } 'l'$, 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 $\text{trans} == 'N' \text{ or } 'n'$,
  
  $C = \alpha A B^T + \alpha B A^T + \beta C$.
  
  If $\text{trans} == 'T'$, $\text{trans} == 't' \text{, 'C'}, \text{ or 'c'}, C = \alpha A^T B + \alpha B^T A + \beta C$.
- **n** specifies the number of rows and the number columns of matrix $C$. If $\text{trans} == 'N' \text{ or } 'n'$, $n$ specifies the number of rows of matrix $A$. If $\text{trans} == 'T' \text{, 't' \text{, 'C'}, \text{ or 'c'}, n$ specifies the number of columns of matrix $A$; $n$ must be at least zero.
- **k** If $\text{trans} == 'N' \text{ or } 'n'$, $k$ specifies the number of columns of matrix $A$. If $\text{trans} == 'T' \text{, 't' \text{, 'C'}, \text{ or 'c'}, k$ 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 $(\text{lda}, \text{ka})$, where $\text{ka}$ is $k$ when $\text{trans} == 'N' \text{ or } 'n'$ and is $n$ otherwise. When $\text{trans} == 'N' \text{ or } 'n'$, 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 $\text{trans} == 'N' \text{ or } 'n'$, $\text{lda}$ must be at least $\max(1, n)$. Otherwise $\text{lda}$ must be at least $\max(1, k)$.
- **B** single-precision complex array of dimensions $(\text{ldb}, \text{kb})$, where $\text{kb}$ is $k$ when $\text{trans} == 'N' \text{ or } 'n'$ and is $n$ otherwise. When $\text{trans} == 'N' \text{ or } 'n'$, 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 $\text{trans} == 'N' \text{ or } 'n'$, $\text{ldb}$ must be at least $\max(1, n)$. Otherwise $\text{ldb}$ must be at least $\max(1, k)$.
- **beta** single-precision complex scalar multiplier applied to $C$. If $\text{beta}$ is zero, $C$ does not have to be a valid input.
CHAPTER 5

BLAS3 Functions

Input (continued)

C 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

C 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/csy2r.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_EXECUTION_FAILED if function failed to launch on GPU

Function cublasCtrmm()

void cublasCtrmm (char side, char uplo, char transa,
char diag, int m, int n, cuComplex alpha,
const cuComplex *A, int lda,
const cuComplex *B, int ldb)

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 $lda$ and $ldb$ are the leading dimensions of the two-dimensional arrays that contain $A$ and $B$, respectively.

**Input**

- **side** specifies whether $\text{op}(A)$ multiplies $B$ from the left or right.
  - If $\text{side} = 'L' \text{ or } 'l'$, $B = \alpha \cdot \text{op}(A) \cdot B$.
  - If $\text{side} = 'R' \text{ or } 'r'$, $B = \alpha \cdot B \cdot \text{op}(A)$.

- **uplo** specifies whether the matrix $A$ is an upper or lower triangular matrix.
  - If $\text{uplo} = 'U' \text{ or } 'u'$, $A$ is an upper triangular matrix.
  - If $\text{uplo} = 'L' \text{ or } 'l'$, $A$ is a lower triangular matrix.

- **transa** specifies $\text{op}(A)$. If $\text{transa} = 'N' \text{ or } 'n'$, $\text{op}(A) = A$.
  - If $\text{transa} = 'T' \text{ or } 't'$, $\text{op}(A) = A^\top$.
  - If $\text{transa} = 'C' \text{ or } 'c'$, $\text{op}(A) = A^H$.

- **diag** specifies whether or not $A$ is a unit triangular matrix. If $\text{diag} = 'U' \text{ or } 'u'$, $A$ is assumed to be unit triangular. If $\text{diag} = 'N' \text{ or } 'n'$, $A$ is not assumed to be unit triangular.

- **m** the number of rows of matrix $B$; $m$ must be at least zero.

- **n** the number of columns of matrix $B$; $n$ must be at least zero.

- **alpha** single-precision complex scalar multiplier applied to $\text{op}(A) \cdot B$ or $B \cdot \text{op}(A)$, respectively. If $\alpha$ is zero, no accesses are made to matrix $A$, and no read accesses are made to matrix $B$.

- **A** single-precision complex array of dimensions $(lda, k)$. If $\text{side} = 'L' \text{ or } 'l'$, $k = m$. If $\text{side} = 'R' \text{ or } 'r'$, $k = n$. If $\text{uplo} = 'U' \text{ or } 'u'$, the leading $k \times k$ 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 $\text{uplo} = 'L' \text{ or } 'l'$, the leading $k \times k$ 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 $\text{diag} = 'U' \text{ or } 'u'$, the diagonal elements of $A$ are not referenced and are assumed to be unity.

- **lda** leading dimension of $A$. When $\text{side} = 'L' \text{ or } 'l'$, it must be at least $\max(1, m)$ and at least $\max(1, n)$ otherwise.
CHAPTER 5  BLAS3 Functions

Input (continued)

B
single-precision complex 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/ctrmm.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 cublasCtrsm()

void
cublasCtrsm (char side, char uplo, char transa,
char diag, int m, int n, cuComplex alpha,
const cuComplex *A, int lda, cuComplex *B,
int ldb)

solves one of the matrix equations

op(A) * X = alpha * B or X * op(A) = alpha * B,
where op(A) = A, op(A) = Aᵀ, or op(A) = Aᴴ;

alpha is a single-precision complex scalar, and X and B are m×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**

- **side** specifies whether \( \text{op}(A) \) appears on the left or right of \( X \):
  - \( \text{side} = 'L' \) or \( 'l' \) indicates solve \( \text{op}(A) \times X = \text{alpha} \times B \);  
  - \( \text{side} = 'R' \) or \( 'r' \) indicates solve \( X \times \text{op}(A) = \text{alpha} \times B \).

- **uplo** specifies whether the matrix A is an upper or lower triangular matrix:
  - \( \text{uplo} = 'U' \) or \( 'u' \) indicates A is an upper triangular matrix;
  - \( \text{uplo} = 'L' \) or \( 'l' \) indicates A is a lower triangular matrix.

- **transa** specifies \( \text{op}(A) \).
  - If \( \text{transa} = 'N' \) or \( 'n' \), \( \text{transa} = 'T' \) or \( 't' \), \( \text{transa} = 'C' \) or \( 'c' \), 
  - \( \text{transa} = 'N' \) or \( 'n' \), \( \text{transa} = 'T' \) or \( 't' \), \( \text{transa} = 'C' \) or \( 'c' \),

- **diag** specifies whether or not A is a unit triangular matrix.
  - If \( \text{diag} = 'U' \) or \( 'u' \), A is assumed to be unit triangular.
  - If \( \text{diag} = 'N' \) or \( 'n' \), A is not assumed to be unit triangular.

- **m** specifies the number of rows of B; m must be at least zero.

- **n** specifies the number of columns of B; n must be at least zero.

- **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.

- **A** single-precision complex array of dimensions \((\text{lda}, k)\), where \( k \) is m when \( \text{side} = 'L' \) or \( 'l' \) and is n when \( \text{side} = 'R' \) or \( 'r' \). If \( \text{uplo} = 'U' \) or \( 'u' \), the leading \( k \times k \) 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 \( \text{uplo} = 'L' \) or \( 'l' \), the leading \( k \times k \) 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 \( \text{diag} = 'U' \) 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 \( \text{side} = 'L' \) or \( 'l' \), lda must be at least \( \max(1, m) \).
  - When \( \text{side} = 'R' \) or \( 'r' \), lda must be at least \( \max(1, n) \).

- **B** single-precision complex array of dimensions \((\text{ldb}, n)\); ldb must be at least \( \max(1, m) \). The leading \( m \times n \) 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; ldb must be at least \( \max(1, m) \).
Output

\[ B \text{ contains the solution matrix } X \text{ satisfying } \operatorname{op}(A) \cdot X = \alpha \cdot B \text{ or } X \cdot \operatorname{op}(A) = \alpha \cdot B. \]

Reference: [http://www.netlib.org/blas/ctrsm.f](http://www.netlib.org/blas/ctrsm.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
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 219
- “Function cublasDsymm()” on page 220
- “Function cublasDsyrk()” on page 222
- “Function cublasDsyrk()” on page 224
- “Function cublasDtrmm()” on page 226
- “Function cublasDtrsm()” on page 228
Function cublasDgemm()

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

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:

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

where $\text{op}(X) = X$ or $\text{op}(X) = X^T$,

and $\alpha$ and $\beta$ are double-precision scalars. $A$, $B$, and $C$ are matrices consisting of double-precision elements, with $\text{op}(A)$ an $m \times k$ matrix, $\text{op}(B)$ a $k \times n$ matrix, and $C$ an $m \times n$ matrix. Matrices $A$, $B$, and $C$ are stored in column-major format, and $\text{lda}$, $\text{ldb}$, and $\text{ldc}$ are the leading dimensions of the two-dimensional arrays containing $A$, $B$, and $C$.

**Input**

- `transa` specifies $\text{op}(A)$. If `transa` is `'N'` or `'n'`, $\text{op}(A) = A$. If `transa` is `'T'`, `'t'`, `'C'`, or `'c'`, $\text{op}(A) = A^T$.
- `transb` specifies $\text{op}(B)$. If `transb` is `'N'` or `'n'`, $\text{op}(B) = B$. If `transb` is `'T'`, `'t'`, `'C'`, or `'c'`, $\text{op}(B) = B^T$.
- `m` number of rows of matrix $\text{op}(A)$ and rows of matrix $C$; $m$ must be at least zero.
- `n` number of columns of matrix $\text{op}(B)$ and number of columns of $C$; $n$ must be at least zero.
- `k` number of columns of matrix $\text{op}(A)$ and number of rows of $\text{op}(B)$; $k$ must be at least zero.
- `alpha` double-precision scalar multiplier applied to $\text{op}(A) \cdot \text{op}(B)$.
- `A` double-precision array of dimensions $(\text{lda}, k)$ if `transa` is `'N'` or `'n'`, and of dimensions $(\text{lda}, m)$ otherwise. If `transa` is `'N'` or `'n'`, $\text{lda}$ must be at least $\max(1, m)$; otherwise, $\text{lda}$ must be at least $\max(1, k)$.
- `lda` leading dimension of two-dimensional array used to store matrix $A$. 


CUDA

CUBLAS Library

Input (continued)

B          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 C.

Output

C          updated based on C = alpha * op(A) * op(B) + beta * C.

Reference: http://www.netlib.org/blas/dgemm.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 cublasDsymm()

void

cublasDsymm (char side, char uplo, int m, int n,
             double alpha, const double *A, int lda,
             const double *B, int ldb, double beta,
             double *C, int ldc)

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**

- \( \text{side} \) specifies whether the symmetric matrix \( A \) appears on the left-hand side or right-hand side of matrix \( B \).
  - If \( \text{side} == 'L' \) or \( 'l' \), \( C = \alpha A * B + \beta C \).
  - If \( \text{side} == 'R' \) or \( 'r' \), \( C = \alpha B * A + \beta C \).

- \( \text{uplo} \) specifies whether the symmetric matrix \( A \) is stored in upper or lower storage mode. If \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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.

- \( m \) specifies the number of rows of matrix \( C \), and the number of rows of matrix \( B \). It also specifies the dimensions of symmetric matrix \( A \) when \( \text{side} == 'L' \) 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 \( \text{side} == 'R' \) or \( 'r' \); \( n \) must be at least zero.

- \( \alpha \) double-precision scalar multiplier applied to \( A * B \) or \( B * A \).

- \( A \) double-precision array of dimensions (\( \text{lda} \), \( \text{ka} \)), where \( \text{ka} \) is \( m \) when \( \text{side} == 'L' \) or \( 'l' \) and is \( n \) otherwise. If \( \text{side} == 'L' \) or \( 'l' \), the leading \( m \times m \) part of array \( A \) must contain the symmetric matrix such that when \( \text{uplo} == 'U' \) or \( 'u' \), the leading \( m \times m \) part stores the upper triangular part of the symmetric matrix, and the strictly lower triangular part of \( A \) is not referenced; and when \( \text{uplo} == 'L' \) or \( 'l' \), the leading \( m \times m \) part stores the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced. If \( \text{side} == 'R' \) or \( 'r' \), the leading \( n \times n \) part of array \( A \) must contain the symmetric matrix such that when \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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.

- \( \text{lda} \) leading dimension of \( A \). When \( \text{side} == 'L' \) or \( 'L' \), it must be at least \( \max (1, m) \) and at least \( \max (1, n) \) otherwise.
CUDA

NVIDIA

CUDA CUBLAS Library

Input (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>double-precision array of dimensions (ldb, n). On entry, the leading m×n part of the array contains the matrix B.</td>
</tr>
<tr>
<td>ldb</td>
<td>leading dimension of B; ldb must be at least max(1, m).</td>
</tr>
<tr>
<td>beta</td>
<td>double-precision scalar multiplier applied to C. If beta is zero, C does not have to be a valid input.</td>
</tr>
<tr>
<td>C</td>
<td>double-precision array of dimensions (ldc, n).</td>
</tr>
<tr>
<td>ldc</td>
<td>leading dimension of C; ldc must be at least max(1, m).</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>updated according to C = alpha * A * B + beta * C or C = alpha * B * A + beta * C.</td>
</tr>
</tbody>
</table>

Reference: [http://www.netlib.org/blas/dsymm.f](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()`

```c
void

void

void

cublasDsyrk (char uplo, char trans, int n, int k,
             double alpha, const double *A, int lda,
             double beta, double *C, int ldc)
```

performs one of the symmetric rank k operations

\[ C = \alpha A A^T + \beta C \text{ or } C = \alpha A^T A + \beta C, \]

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** specifies whether the symmetric matrix $C$ is stored in upper or lower storage mode. If $\text{uplo} = 'U'$ 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 $\text{uplo} = 'L'$ 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 $\text{trans} = 'N'$ or 'n', $C = \alpha A * A^T + \beta C$. If $\text{trans} = 'T'$, 't', 'C', or 'c', $C = \alpha A^T * A + \beta C$.

**n** specifies the number of rows and the number columns of matrix $C$. If $\text{trans} = 'N'$ or 'n', $n$ specifies the number of rows of matrix $A$. If $\text{trans} = 'T'$, 't', 'C', or 'c', $n$ specifies the number of columns of matrix $A$; $n$ must be at least zero.

**k** If $\text{trans} = 'N'$ or 'n', $k$ specifies the number of columns of matrix $A$. If $\text{trans} = 'T'$, '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 $A * A^T$ or $A^T * A$.

**A** double-precision array of dimensions $(\text{lda}, \text{ka})$, where $\text{ka}$ is $k$ when $\text{trans} = 'N'$ or 'n' and is $n$ otherwise. When $\text{trans} = 'N'$ 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 $\text{trans} = 'N'$ or 'n', $\text{lda}$ must be at least $\max(1, n)$. Otherwise $\text{lda}$ must be at least $\max(1, k)$.

**beta** double-precision scalar multiplier applied to $C$. If $\beta$ is zero, $C$ is not read.
CUDA

CUBLAS Library

Input (continued)

C is a 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 is the leading dimension of C; ldc must be at least max(1, n).

Output

C is 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_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 cublasDsyrk (char uplo, char trans, int n, int k, double alpha, const double *A, int lda, const double *B, int ldb, double beta, double *C, int ldc)

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×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×k in the first case and k×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 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×k 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', lda must be at least max(1, n). Otherwise lda must be at least max(1, k).

B 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×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 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.
Input (continued)

C: 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

C updated according to

\[
C = \alpha * A^T * B + \alpha * B * A^T + \beta * C \quad \text{or} \\
C = \alpha * A^T * B + \alpha * B^T * A + \beta * C. \\
\]

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

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if m &lt; 0, n &lt; 0, or k &lt; 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function cublasDtrmm()

```c
void

\text{cublasDtrmm (char side, char uplo, char transa, char diag, int m, int n, double alpha, const double }^*A, \text{int lda, const double }^*B, \text{int ldb})
```

performs one of the matrix-matrix operations

\[
B = \alpha \cdot \text{op}(\text{A}) \cdot B \quad \text{or} \quad B = \alpha \cdot B \cdot \text{op}(\text{A}), \\
\text{where} \quad \text{op}(\text{A}) = \text{A} \quad \text{or} \quad \text{op}(\text{A}) = \text{A}^T,
\]
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 lda and ldb are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

<table>
<thead>
<tr>
<th>Side</th>
<th>Specifies whether ( \text{op}(A) ) multiplies B from the left or right.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{side} )</td>
<td>If ( \text{side} == \text{'L'} ) or \text{'l'} ), ( B = \alpha \cdot \text{op}(A) \cdot B ).</td>
</tr>
<tr>
<td></td>
<td>If ( \text{side} == \text{'R'} ) or \text{'r'} ), ( B = \alpha \cdot B \cdot \text{op}(A) ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Upl</th>
<th>Specifies whether the matrix A is an upper or lower triangular matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{uplo} )</td>
<td>If ( \text{uplo} == \text{'U'} ) or \text{'u'} ), A is an upper triangular matrix.</td>
</tr>
<tr>
<td></td>
<td>If ( \text{uplo} == \text{'L'} ) or \text{'l'} ), A is a lower triangular matrix.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TransA</th>
<th>Specifies the form of ( \text{op}(A) ) to be used in the matrix multiplication.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{transa} )</td>
<td>If ( \text{transa} == \text{'N'} ) or \text{'n'} ), ( \text{op}(A) = A ).</td>
</tr>
<tr>
<td></td>
<td>If ( \text{transa} == \text{'T'} ), \text{'t'}, \text{'C'}, \text{or 'c'}, ( \text{op}(A) = A^T ).</td>
</tr>
</tbody>
</table>

| Diag    | Specifies whether or not A is a unit triangular matrix. If \( \text{diag} == \text{'U'} \) or \text{'u'} \), A is assumed to be unit triangular. If \( \text{diag} == \text{'N'} \) or \text{'n'} \), A is not assumed to be unit triangular. |

| Number of rows of matrix B; m must be at least zero. |
| Number of columns of matrix B; n must be at least zero. |

| Alpha    | Double-precision scalar multiplier applied to \( \text{op}(A) \cdot B \) or \( B \cdot \text{op}(A) \), respectively. If \( \alpha \) is zero, no accesses are made to matrix \( A \), and no read accesses are made to matrix \( B \). |

\( A \) double-precision array of dimensions \( (\text{lda}, k) \). If \( \text{side} == \text{'L'} \) or \text{'l'} \), \( k = m \). If \( \text{side} == \text{'R'} \) or \text{'r'} \), \( k = n \). If \( \text{uplo} == \text{'U'} \) or \text{'u'} \), the leading \( k \times k \) 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 \( \text{uplo} == \text{'L'} \) or \text{'l'} \), the leading \( k \times k \) 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 \( \text{diag} == \text{'U'} \) or \text{'u'} \), the diagonal elements of \( A \) are not referenced and are assumed to be unity.

| Leading dimension of \( A \). When \( \text{side} == \text{'L'} \) or \text{'l'} \), it must be at least \( \max(1,m) \) and at least \( \max(1,n) \) otherwise. |
Function cublasDtrsm()

```c
void

cublasDtrsm (char side, char uplo, char transa,
char diag, int m, int n, double alpha,
const double *A, int lda, double *B,
int ldb)
```

solves one of the matrix equations

\[
\text{op}(A) \times X = \alpha \times B \text{ or } X \times \text{op}(A) = \alpha \times B,
\]

where \( \text{op}(A) = A \text{ or } \text{op}(A) = A^T \),

\( \alpha \) 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 \( \text{lda} \) and \( \text{ldb} \) are the leading dimensions of the two-dimensional arrays that contain \( A \) and \( B \), respectively.
## CHAPTER 5 BLAS3 Functions

### Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>side</code></td>
<td>specifies whether ( \text{op}(A) ) appears on the left or right of ( X ):&lt;br&gt;( \text{side} = 'L' ) or ( 'l' ) indicates solve ( \text{op}(A) \times X = \alpha \times B );&lt;br&gt;( \text{side} = 'R' ) or ( 'r' ) indicates solve ( X \times \text{op}(A) = \alpha \times B ).</td>
</tr>
<tr>
<td><code>uplo</code></td>
<td>specifies whether the matrix ( A ) is an upper or lower triangular matrix:&lt;br&gt;( \text{uplo} = 'U' ) or ( 'u' ) indicates ( A ) is an upper triangular matrix;&lt;br&gt;( \text{uplo} = 'L' ) or ( 'l' ) indicates ( A ) is a lower triangular matrix.</td>
</tr>
<tr>
<td><code>transa</code></td>
<td>specifies the form of ( \text{op}(A) ) to be used in matrix multiplication.&lt;br&gt;( \text{transa} = 'N' ) or ( 'n' ), .&lt;br&gt;( \text{transa} = 'T' ), ( 't' ), ( 'C' ), or ( 'c' ), .</td>
</tr>
<tr>
<td><code>diag</code></td>
<td>specifies whether or not ( A ) is a unit triangular matrix.&lt;br&gt;( \text{diag} = 'U' ) or ( 'u' ), ( A ) is assumed to be unit triangular.&lt;br&gt;( \text{diag} = 'N' ) or ( 'n' ), ( A ) is not assumed to be unit triangular.</td>
</tr>
<tr>
<td><code>m</code></td>
<td>specifies the number of rows of ( B ); ( m ) must be at least zero.</td>
</tr>
<tr>
<td><code>n</code></td>
<td>specifies the number of columns of ( B ); ( n ) must be at least zero.</td>
</tr>
<tr>
<td><code>alpha</code></td>
<td>double-precision scalar multiplier applied to ( B ). When ( \alpha ) is zero, ( A ) is not referenced and ( B ) does not have to be a valid input.</td>
</tr>
<tr>
<td><code>A</code></td>
<td>double-precision array of dimensions ( (\text{lda}, k) ), where ( k ) is ( m ) when ( \text{side} = 'L' ) or ( 'l' ) and is ( n ) when ( \text{side} = 'R' ) or ( 'r' ).&lt;br&gt;( \text{uplo} = 'U' ) or ( 'u' ), the leading ( k \times 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 ( \text{uplo} = 'L' ) or ( 'l' ), the leading ( k \times 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 ( \text{diag} = 'U' ) or ( 'u' ), the diagonal elements of ( A ) are not referenced and are assumed to be unity.</td>
</tr>
<tr>
<td><code>lda</code></td>
<td>leading dimension of the two-dimensional array containing ( A ).&lt;br&gt;( \text{when} \ \text{side} = 'L' ) or ( 'l' ), ( \text{lda} ) must be at least ( \max(1, m) ).&lt;br&gt;( \text{when} \ \text{side} = 'R' ) or ( 'r' ), ( \text{lda} ) must be at least ( \max(1, n) ).</td>
</tr>
<tr>
<td><code>B</code></td>
<td>double-precision array of dimensions ( (\text{ldb}, n) ); ( \text{ldb} ) must be at least ( \max(1, m) ). The leading ( m \times n ) part of the array ( B ) must contain the right-hand side matrix ( B ). On exit, ( B ) is overwritten by the solution matrix ( X ).</td>
</tr>
<tr>
<td><code>ldb</code></td>
<td>leading dimension of the two-dimensional array containing ( B ); ( \text{ldb} ) must be at least ( \max(1, m) ).</td>
</tr>
</tbody>
</table>
Output

\[ B \] contains the solution matrix \( X \) satisfying \( \text{op}(A) \cdot X = \alpha \cdot B \) or
\[ X \cdot \text{op}(A) = \alpha \cdot B. \]

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

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

- \texttt{CUBLAS_STATUS_NOT_INITIALIZED} if CUBLAS library was not initialized
- \texttt{CUBLAS_STATUS_INVALID_VALUE} if \( m < 0, n < 0, \) or \( k < 0 \)
- \texttt{CUBLAS_STATUS_ARCH_MISMATCH} if function invoked on device that does not support double precision
- \texttt{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 232
- “Function cublasZhemm()” on page 233
- “Function cublasZherk()” on page 235
- “Function cublasZher2k()” on page 238
- “Function cublasZsymm()” on page 240
- “Function cublasZsyrk()” on page 242
- “Function cublasZsyr2k()” on page 244
- “Function cublasZtrmm()” on page 246
- “Function cublasZtrsm()” on page 248
Function cublasZgemm()

```c
void

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

performs one of the matrix-matrix operations

\[
C = \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C,
\]

where \( \text{op}(X) = X, \text{op}(X) = X^T, \) or \( \text{op}(X) = X^H; \)

and \( \alpha \) and \( \beta \) are double-precision complex scalars. \( A, B, \) and \( C \) are matrices consisting of double-precision complex elements, with \( \text{op}(A) \) an \( m \times k \) matrix, \( \text{op}(B) \) a \( k \times n \) matrix, and \( C \) an \( m \times n \) matrix.

**Input**

- **transa** specifies \( \text{op}(A) \). If \( \text{transa} = 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  - If \( \text{transa} = 'T' \) or \( 't' \), \( \text{op}(A) = A^T \).
  - If \( \text{transa} = 'C' \) or \( 'c' \), \( \text{op}(A) = A^H \).
- **transb** specifies \( \text{op}(B) \). If \( \text{transb} = 'N' \) or \( 'n' \), \( \text{op}(B) = B \).
  - If \( \text{transb} = 'T' \) or \( 't' \), \( \text{op}(B) = B^T \).
  - If \( \text{transb} = 'C' \) or \( 'c' \), \( \text{op}(B) = B^H \).
- **m** number of rows of matrix \( \text{op}(A) \) and rows of matrix \( C \); \( m \) must be at least zero.
- **n** number of columns of matrix \( \text{op}(B) \) and number of columns of \( C \); \( n \) must be at least zero.
- **k** number of columns of matrix \( \text{op}(A) \) and number of rows of \( \text{op}(B) \); \( k \) must be at least zero.
- **alpha** double-precision complex scalar multiplier applied to \( \text{op}(A) \cdot \text{op}(B) \).
- **A** double-precision complex array of dimension \((\text{lda}, k)\) if \( \text{transa} = 'N' \) or \( 'n' \), and of dimension \((\text{lda}, m)\) otherwise.
- **lda** leading dimension of \( A \). When \( \text{transa} = 'N' \) or \( 'n' \), it must be at least \( \max(1, m) \) and at least \( \max(1, k) \) otherwise.
- **B** double-precision complex array of dimension \((\text{ldb}, n)\) if \( \text{transb} = 'N' \) or \( 'n' \), and of dimension \((\text{ldb}, k)\) otherwise.
CHAPTER 5 BLAS3 Functions

Function cublasZhemm()

```c
void
cublasZhemm (char side, char uplo, int m, int n,
cuDoubleComplex alpha,
const cuDoubleComplex *A, int lda,
const cuDoubleComplex *B, int ldb,
cuDoubleComplex beta, cuDoubleComplex *C,
int ldc)
```

performs one of the matrix-matrix operations

\[
C = \alpha A \times B + \beta C \quad \text{or} \quad C = \alpha B \times A + \beta C,
\]

where \(\alpha\) and \(\beta\) are double-precision complex scalars, \(A\) is a Hermitian matrix consisting of double-precision complex elements

Input (continued)

\(\text{ldb}\) leading dimension of \(B\). When \(\text{transb} == 'N'\) or \(\text{transb}'n'\), it must be at least \(\max(1,k)\) and at least \(\max(1,n)\) otherwise.

\(\text{beta}\) double-precision complex scalar multiplier applied to \(C\). If \(\beta\) is zero, \(C\) does not have to be a valid input.

\(\text{C}\) double-precision array of dimensions \((\text{ldc},n)\).

\(\text{ldc}\) leading dimension of \(C\); \(\text{ldc}\) must be at least \(\max(1,m)\).

Output

\(C\) updated according to \(C = \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C\).

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

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

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

Input (continued)

\(\alpha\) and \(\beta\) are double-precision complex scalars.
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 Hermitian matrix \( A \) appears on the left-hand side or right-hand side of matrix \( B \).
  - If \( \text{side} == 'L' \) or \( 'l' \), \( C = \alpha \cdot A \cdot B + \beta \cdot C \).
  - If \( \text{side} == 'R' \) or \( 'r' \), \( C = \alpha \cdot B \cdot A + \beta \cdot C \).

- **uplo** specifies whether the Hermitian matrix \( A \) is stored in upper or lower storage mode. If \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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.

- **m** specifies the number of rows of matrix \( C \), and the number of rows of matrix \( B \). It also specifies the dimensions of Hermitian matrix \( A \) when \( \text{side} == 'L' \) 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 Hermitian matrix \( A \) when \( \text{side} == 'R' \) or \( 'r' \); \( n \) must be at least zero.

- **alpha** double-precision complex scalar multiplier applied to \( A \cdot B \) or \( B \cdot A \).

- **A** double-precision complex array of dimensions \( (\text{lda}, \text{ka}) \), where \( \text{ka} \) is \( m \) when \( \text{side} == 'L' \) or \( 'l' \) and is \( n \) otherwise. If \( \text{side} == 'L' \) or \( 'l' \), the leading \( m \times m \) part of array \( A \) must contain the Hermitian matrix such that when \( \text{uplo} == 'U' \) or \( 'u' \), the leading \( n \times n \) part stores the upper triangular part of the Hermitian matrix, and the strictly lower triangular part of \( A \) is not referenced; and when \( \text{uplo} == 'L' \) or \( 'l' \), the leading \( m \times m \) part stores the lower triangular part of the Hermitian matrix, and the strictly upper triangular part is not referenced. If \( \text{side} == 'R' \) or \( 'r' \), the leading \( n \times n \) part of array \( A \) must contain the Hermitian matrix such that when \( \text{uplo} == 'U' \) or \( 'u' \), the leading \( n \times n \) part stores the upper triangular part of the Hermitian matrix, and the strictly lower triangular part of \( A \) is not referenced; and when \( \text{uplo} == 'L' \) 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 \( \text{side} == 'L' \) or \( 'l' \), it must be at least \( \max(1, m) \) and at least \( \max(1, n) \) otherwise.
CHAPTER 5  BLAS3 Functions

Input (continued)

- **B**: 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

- **C**: updated according to \(C = alpha * A \cdot B + beta \cdot C\) or
- \(C = alpha * B \cdot A + beta \cdot C\).

Reference: [http://www.netlib.org/blas/zhemm.f](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()

```c
void
cublasZherk (char uplo, char trans, int n, int k,
             double alpha, const cuDoubleComplex *A,
             int lda, double beta, cuDoubleComplex *C,
             int ldc)
```

performs one of the Hermitian rank \(k\) operations

\[
C = alpha \cdot A \cdot A^H + beta \cdot C \quad \text{or} \quad C = alpha \cdot A^H \cdot A + beta \cdot C,
\]

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
\(n \times k\) 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 \(\text{uplo} == \text{'U'}\) or \text{'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 \(\text{uplo} == \text{'L'}\) or \text{'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 \(\text{trans} == \text{'N'}\) or \text{'n'},
  \(C = \alpha A^* A + \beta C\). If \(\text{trans} == \text{'T'}\), \text{'t'}, \text{'C'}, \text{or 'c'},
  \(C = \alpha A^H A + \beta C\).

- **n** specifies the number of rows and the number columns of matrix \(C\). If
  \(\text{trans} == \text{'N'}\) or \text{'n'}, \(n\) specifies the number of rows of matrix \(A\). If
  \(\text{trans} == \text{'T'}\), \text{'t'}, \text{'C'}, \text{or 'c'}, \(n\) specifies the number of columns
  of matrix \(A\); \(n\) must be at least zero.

- **k** if \(\text{trans} == \text{'N'}\) or \text{'n'}, \(k\) specifies the number of columns of
  matrix \(A\). If \(\text{trans} == \text{'T'}\), \text{'t'}, \text{'C'}, \text{or 'c'}, \(k\) specifies the number
  of rows of matrix \(A\); \(k\) must be at least zero.

- **alpha** double-precision scalar multiplier applied to \(A^H A^*\) or \(A^* A\).

- **A** double-precision complex array of dimensions \((\text{lda}, ka)\), where \(ka\) is
  \(k\) when \(\text{trans} == \text{'N'}\) or \text{'n'}\) and is \(n\) otherwise. When \(\text{trans} ==
  \text{'N'}\) or \text{'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 \(\text{trans} == \text{'N'}\) or \text{'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, \(C\) does not have to be a valid input.
CHAPTER 5  
BLAS3 Functions

Input (continued)

C

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

C

updated according to C = alpha * A * A**H + beta * C or

C = alpha * A**H * A + beta * C.

Reference: http://www.netlib.org/blas/zherk.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 cublasZher2k()

```c
void
    cublasZher2k (char uplo, char trans, int n, int k, 
                  cuDoubleComplex alpha, 
                  const cuDoubleComplex *A, int lda, 
                  const cuDoubleComplex *B, int ldb, 
                  double beta, cuDoubleComplex *C, int ldc)
```

performs one of the Hermitian rank 2k operations

\[
C = \alpha A^H B + \alpha^* B^H A + \beta C \quad \text{or} \\
C = \alpha A^H B + \alpha^* B^H A + \beta C,
\]

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

- `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'`, \(C = \alpha A^H B + \alpha^* B^H A + \beta C\). If `trans == 'T'`, 't', 'C', or 'c', \(C = \alpha A^H B + \alpha^* B^H A + \beta C\).
- `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` double-precision complex multiplier.
CHAPTER 5

BLAS3 Functions

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

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×k part of array A contains the matrix A; otherwise, the leading k×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).

B 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×k part of array B contains the matrix B; otherwise, the leading k×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.

C 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 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

C updated according to C = alpha * A * B**H + alpha * B * A**H + beta * C or C = alpha * A**H * B + alpha * B**H * A + beta * C.
Function cublasZsymm()

```c
void
cublasZsymm (char side, char uplo, int m, int n,
   cuDoubleComplex alpha,
   const cuDoubleComplex *A,
   int lda, const cuDoubleComplex *B, int ldb,
   cuDoubleComplex beta, cuDoubleComplex *C,
   int ldc)
```

performs one of the matrix-matrix operations

\[
C = \alpha A B + \beta C \quad \text{or} \quad C = \alpha B A + \beta C,
\]

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 \(\text{side} = 'L'\) or \('l'\), \(C = \alpha A B + \beta C\).
  - If \(\text{side} = 'R'\) or \('r'\), \(C = \alpha B A + \beta C\).

- **uplo** specifies whether the symmetric matrix \(A\) is stored in upper or lower storage mode. If \(\text{uplo} = 'U'\) 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 \(\text{uplo} = 'L'\) 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.

- **m** specifies the number of rows of matrix \(C\), and the number of rows of matrix \(B\). It also specifies the dimensions of symmetric matrix \(A\) when \(\text{side} = 'L'\) or \('l'\); \(m\) must be at least zero.

**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
CHAPTER 5
BLAS3 Functions

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 \( \text{side} == 'R' \) or \( 'r' \); \( n \) must be at least zero.

\( \text{alpha} \)

double-precision complex scalar multiplier applied to \( A \) \( \times \) \( B \) or \( B \) \( \times \) \( A \).

\( A \)
double-precision complex array of dimensions \((\text{lda}, \text{ka})\), where \( \text{ka} \) is \( m \) when \( \text{side} == 'L' \) or \( 'l' \) and is \( n \) otherwise. If \( \text{side} == 'L' \) or \( 'l' \), the leading \( m \times m \) part of array \( A \) must contain the symmetric matrix such that when \( \text{uplo} == 'U' \) or \( 'u' \), the leading \( m \times m \) part stores the upper triangular part of the symmetric matrix, and the \( m \times m \) part is not referenced; and when \( \text{uplo} == 'L' \) or \( 'l' \), the leading \( m \times m \) part stores the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced. If \( \text{side} == 'R' \) or \( 'r' \), the leading \( n \times n \) part of array \( A \) must contain the symmetric matrix such that when \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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.

\( \text{lda} \)
-leading dimension of \( A \). When \( \text{side} == 'L' \) or \( 'l' \), it must be at least \( \max(1, m) \) and at least \( \max(1, n) \) otherwise.

\( \text{B} \)
double-precision complex array of dimensions \((\text{ldb}, n)\). On entry, the leading \( m \times n \) part of the array contains the matrix \( B \).

\( \text{ldb} \)
-leading dimension of \( B \); \( \text{ldb} \) must be at least \( \max(1, m) \).

\( \text{beta} \)
-double-precision complex scalar multiplier applied to \( C \). If \( \text{beta} \) is zero, \( C \) does not have to be a valid input.

\( \text{C} \)
double-precision complex array of dimensions \((\text{ldc}, n)\).

\( \text{ldc} \)
-leading dimension of \( C \); \( \text{ldc} \) must be at least \( \max(1, m) \).

Output

\( \text{C} \) updated according to \( C = \alpha \times A \times B \times \beta + C \) or \( C = \alpha \times B \times A \times \beta + C \).

Reference: http://www.netlib.org/blas/zsymm.f
Error status for this function can be retrieved via `cublasGetError()`.

---

**Error Status**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( m &lt; 0 ) or ( n &lt; 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

---

**Function cublasZsyrk()**

```c
void
cublasZsyrk (char uplo, char trans, int n, int k,
cuDoublesComplex alpha,
const cuDoublesComplex *A, int lda,
cuDoublesComplex beta,
cuDoublesComplex *C, int ldc)
```

performs one of the symmetric rank \( k \) operations

\[ C = \text{alpha} \cdot A \cdot A^T + \text{beta} \cdot C \quad \text{or} \quad C = \text{alpha} \cdot A^T \cdot A + \text{beta} \cdot C, \]  

where \( \text{alpha} \) and \( \text{beta} \) 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**

- `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'`, \( C = \text{alpha} \cdot A \cdot A^T + \text{beta} \cdot C \). If `trans == 'T'`, `'t'`, `'C'`, or `'c'`, \( C = \text{alpha} \cdot A^T \cdot A + \text{beta} \cdot C \).
**Input (continued)**

- \(n\) specifies the number of rows and the number columns of matrix \(C\). If \(\text{trans} = 'N'\) or 'n', \(n\) specifies the number of rows of matrix \(A\). If \(\text{trans} = 'T', 't', 'C', \text{or} 'c'\), \(n\) specifies the number of columns of matrix \(A\); \(n\) must be at least zero.
  
- \(k\) specifies the number of columns of matrix \(A\). If \(\text{trans} = 'N'\) or 'n', \(k\) specifies the number of columns of matrix \(A\). If \(\text{trans} = 'T', 't', 'C', \text{or} 'c'\), \(k\) specifies the number of rows of matrix \(A\); \(k\) must be at least zero.
  
- \(\alpha\) is a double-precision complex scalar multiplier applied to \(A * A^{\dagger}\) or \(A^T * A\).
  
- \(A\) is a double-precision complex array of dimensions \((\text{lda, ka})\), where \(\text{ka}\) is \(k\) when \(\text{trans} = 'N'\) or 'n' and is \(n\) otherwise. When \(\text{trans} = 'N'\) 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 \(A\).
  
- \(\text{lda}\) is the leading dimension of \(A\). When \(\text{trans} = 'N'\) or 'n', \(\text{lda}\) must be at least \(\max(1, n)\). Otherwise, \(\text{lda}\) must be at least \(\max(1, k)\).
  
- \(\beta\) is a double-precision complex scalar multiplier applied to \(C\). If \(\beta\) is zero, \(C\) is not read.
  
- \(C\) is a double-precision complex array of dimensions \((\text{ldc, n})\). If \(\text{uplo} = 'U'\) 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 \(\text{uplo} = 'L'\) 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.
  
- \(\text{ldc}\) is the leading dimension of \(C\); \(\text{ldc}\) must be at least \(\max(1, n)\).

**Output**

- \(C\) is updated according to \(C = \alpha * A * A^{\dagger} + \beta * C\) or \(C = \alpha * A^{\dagger} * A + \beta * C\).

**Reference:** [http://www.netlib.org/blas/zsyrk.f](http://www.netlib.org/blas/zsyrk.f)
Error status for this function can be retrieved via cublasGetError().

Error Status

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if n &lt; 0 or k &lt; 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function cublasZsyr2k()

```c
void

 void cublasZsyr2k (char uplo, char trans, int n, int k,
                    cuDoubleComplex alpha,
                    const cuDoubleComplex *A, int lda,
                    const cuDoubleComplex *B, int ldb,
                    cuDoubleComplex beta,
                    cuDoubleComplex *C, int ldc)
``` performs one of the symmetric rank 2k operations

\[
C = \alpha A * B^T + \alpha B * A^T + \beta C \quad \text{or} \quad C = \alpha A^T * B + \alpha B^T * A + \beta C,
\]

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**

- **uplo** specifies whether the symmetric matrix \( C \) is stored in upper or lower storage mode. If \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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 \( \text{trans} == 'N' \) or \( 'n' \), \( C = \alpha A * B^T + \alpha B * A^T + \beta C \). If \( \text{trans} == 'T', 't', 'C', \) or \( 'c' \), \( C = \alpha A^T * B + \alpha B^T * A + \beta C \).
CHAPTER 5  BLAS3 Functions

Input (continued)

\( n \)

specifies the number of rows and the number columns of matrix \( C \). If \( \text{trans} == \text{'N'} \) or \( \text{'n'} \), \( n \) specifies the number of rows of matrices \( A \) and \( B \). If \( \text{trans} == \text{'T'}, \text{'t'}, \text{'C'}, \text{or 'c'} \), \( n \) specifies the number of columns of matrices \( A \) and \( B \); \( n \) must be at least zero.

\( k \)

If \( \text{trans} == \text{'N'} \) or \( \text{'n'} \), \( k \) specifies the number of columns of matrices \( A \) and \( B \). If \( \text{trans} == \text{'T'}, \text{'t'}, \text{'C'}, \text{or 'c'} \), \( k \) specifies the number of rows of matrices \( A \) and \( B \); \( k \) must be at least zero.

\( \text{alpha} \)

double-precision scalar multiplier.

\( A \)

double-precision array of dimensions \((\text{lda}, \text{ka})\), where \( \text{ka} \) is \( k \) when \( \text{trans} == \text{'N'} \) or \( \text{'n'} \) and is \( n \) otherwise. When \( \text{trans} == \text{'N'} \) or \( \text{'n'} \), 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 \( A \).

\( \text{lda} \)

leading dimension of \( A \). When \( \text{trans} == \text{'N'} \) or \( \text{'n'} \), \( \text{lda} \) must be at least \( \max(1, n) \). Otherwise \( \text{lda} \) must be at least \( \max(1, k) \).

\( B \)

double-precision array of dimensions \((\text{ldb}, \text{kb})\), where \( \text{kb} = k \) when \( \text{trans} == \text{'N'} \) or \( \text{'n'} \), and \( k = n \) otherwise. When \( \text{trans} == \text{'N'} \) or \( \text{'n'} \), 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 \( B \).

\( \text{ldb} \)

leading dimension of \( B \). When \( \text{trans} == \text{'N'} \) or \( \text{'n'} \), \( \text{ldb} \) must be at least \( \max(1, n) \). Otherwise \( \text{ldb} \) must be at least \( \max(1, k) \).

\( \text{beta} \)

double-precision scalar multiplier applied to \( C \). If \( \text{beta} \) is zero, \( C \) does not have to be a valid input.

\( C \)

double-precision array of dimensions \((\text{ldc}, n)\). If \( \text{uplo} == \text{'U'} \) or \( \text{'u'} \), 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 \( \text{uplo} == \text{'L'} \) or \( \text{'l'} \), 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.

\( \text{ldc} \)

leading dimension of \( C \); \( \text{ldc} \) must be at least \( \max(1, n) \).
### CUDA

#### CUBLAS Library

**Output**

- Updated according to:
  
  \[
  C = \alpha A^* B + \alpha B^* A + \beta C \quad \text{or} \quad 
  C = \alpha A^T B + \alpha B^T A + \beta C.
  \]

**Reference:** [http://www.netlib.org/blas/zsyr2k.f](http://www.netlib.org/blas/zsyr2k.f)

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

**Error Status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CUBLAS_STATUS_NOT_INITIALIZED</code></td>
<td>If CUBLAS library was not initialized</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_INVALID_VALUE</code></td>
<td>If ( m &lt; 0, n &lt; 0, ) or ( k &lt; 0 )</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ARCH_MISMATCH</code></td>
<td>If function invoked on device that does not support double precision</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_EXECUTION_FAILED</code></td>
<td>If function failed to launch on GPU</td>
</tr>
</tbody>
</table>

**Function `cublasZtrmm()`**

```c
void

\text{cublasZtrmm (char side, char uplo, char transa, char diag, int m, int n,
\quad cuDoubleComplex alpha,
\quad const cuDoubleComplex *A, int lda,
\quad const cuDoubleComplex *B, int ldb)}
```

performs one of the matrix-matrix operations:

\[
B = \alpha \text{op}(A) \ast B \quad \text{or} \quad B = \alpha \ast \text{op}(A),
\]

where \( \text{op}(A) = A, \text{op}(A) = A^T, \) or \( \text{op}(A) = A^H; \)
\[
\alpha \text{ is a double-precision complex scalar; } B \text{ is an } m \times n \text{ matrix consisting of double-precision complex elements; and } A \text{ is a unit or non-unit, upper or lower triangular matrix consisting of double-precision complex elements.}
\]

Matrices \( A \) and \( B \) are stored in column-major format, and \( \text{lda} \) and \( \text{ldb} \) are the leading dimensions of the two-dimensional arrays that contain \( A \) and \( B \), respectively.
CHAPTER 5 BLAS3 Functions

Input

side specifies whether \( \text{op}(A) \) multiplies \( B \) from the left or right.
- If \( \text{side} == 'L' \) or \( 'l' \), \( B = \alpha \cdot \text{op}(A) \cdot B \).
- If \( \text{side} == 'R' \) or \( 'r' \), \( B = \alpha \cdot B \cdot \text{op}(A) \).

uplo specifies whether the matrix \( A \) is an upper or lower triangular matrix.
- If \( \text{uplo} == 'U' \) or \( 'u' \), \( A \) is an upper triangular matrix.
- If \( \text{uplo} == 'L' \) or \( 'l' \), \( A \) is a lower triangular matrix.

transa specifies \( \text{op}(A) \).
- If \( \text{transa} == 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
- If \( \text{transa} == 'T' \) or \( 't' \), \( \text{op}(A) = A^T \).
- If \( \text{transa} == 'C' \) or \( 'c' \), \( \text{op}(A) = A^H \).

diag specifies whether or not \( A \) is a unit triangular matrix.
- If \( \text{diag} == 'U' \) or \( 'u' \), \( A \) is assumed to be unit triangular.
- If \( \text{diag} == 'N' \) or \( 'n' \), \( A \) is not assumed to be unit triangular.

m the number of rows of matrix \( B \); \( m \) must be at least zero.

n the number of columns of matrix \( B \); \( n \) must be at least zero.

alpha double-precision complex scalar multiplier applied to \( \text{op}(A) \cdot B \) or \( B \cdot \text{op}(A) \), respectively.
- If \( \alpha \) is zero, no accesses are made to matrix \( A \) and no read accesses are made to matrix \( B \).

\( A \) double-precision complex array of dimensions \( (\text{lda}, k) \).
- If \( \text{side} == 'L' \) or \( 'l' \), \( k = m \).
- If \( \text{side} == 'R' \) or \( 'r' \), \( k = n \).
- If \( \text{uplo} == 'U' \) or \( 'u' \), the leading \( k \times k \) 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 \( \text{uplo} == 'L' \) or \( 'l' \), the leading \( k \times k \) lower triangular part of the array \( A \) must contain the lower triangular matrix, and the strictly upper triangular part of \( A \) is not referenced.

lda leading dimension of \( A \).
- When \( \text{side} == 'L' \) or \( 'l' \), it must be at least \( \max(1, m) \) and at least \( \max(1, n) \) otherwise.

\( B \) double-precision complex array of dimensions \( (\text{ldb}, n) \).
- 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 \( B \); \( \text{ldb} \) 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/ztrmm.f

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

Error Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if m &lt; 0 or n &lt; 0</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function `cublasZtrsm()`

```c
void
cublasZtrsm (char side, char uplo, char transa,
    char diag, int m, int n,
    cuDoubleComplex alpha,
    const cuDoubleComplex *A, int lda,
    cuDoubleComplex *B, int ldb)
```

solves one of the matrix equations

\[
\text{op}(A) \times X = \alpha \times B \text{ or } X \times \text{op}(A) = \alpha \times B,
\]

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

\(\alpha\) 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.
CHAPTER 5 BLAS3 Functions

Input

side specifies whether $\text{op}(A)$ appears on the left or right of $X$:
side == 'L' or 'l' indicates solve $\text{op}(A) \times X = \alpha \times B$;
side == 'R' or 'r' indicates solve $X \times \text{op}(A) = \alpha \times B$.

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 $\text{op}(A)$. If transa == 'N' or 'n', $\text{op}(A) = A$.
If transa == 'T' or 't', $\text{op}(A) = A^T$.
If transa == 'C' or 'c', $\text{op}(A) = A^H$.

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.

m specifies the number of rows of $B$; $m$ must be at least zero.

n specifies the number of columns of $B$; $n$ must be at least zero.

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.

A double-precision complex array of dimensions (lda, k), where $k$ is $m$
when side == 'L' or 'l' and is $n$ when side == 'R' or 'r'. If
uplo == 'U' or 'u', the leading $k \times k$ 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 $k \times k$ 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 'l', lda must be at least max(1, m).
When side == 'R' or 'r', lda must be at least max(1, n).

B double-precision complex array of dimensions (ldb, n); ldb must be
at least max(1, m). The leading $m \times n$ 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 $B$; ldb
must be at least max(1, m).
Output

\[ B \] contains the solution matrix \( X \) satisfying \( \text{op}(A) \ast X = \alpha \ast B \) or \( X \ast \text{op}(A) = \alpha \ast B \).

Reference: [http://www.netlib.org/blas/ztrsm.f](http://www.netlib.org/blas/ztrsm.f)

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

Error Status

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBLAS_STATUS_NOT_INITIALIZED</td>
<td>if CUBLAS library was not initialized</td>
</tr>
<tr>
<td>CUBLAS_STATUS_INVALID_VALUE</td>
<td>if ( m &lt; 0 ) or ( n &lt; 0 )</td>
</tr>
<tr>
<td>CUBLAS_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
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 -fno-second-underscore is required to use these wrappers as provided. Also, the use of the default calling conventions with regard to argument and return value passing is expected. Using the flag -fno-f2c changes the default calling convention with respect to these two items.

The thunking wrappers allow interfacing to existing Fortran applications without any changes to the application. During each call, the wrappers allocate GPU memory, copy source data from CPU memory space to GPU memory space, call CUBLAS, and finally copy back the results to CPU memory space and deallocate the GPU memory. As this process causes very significant call overhead, these wrappers are intended for light testing, not for production code. To use the thunking wrappers, the application needs to be compiled with the file fortran_thunking.c.

The direct wrappers, intended for production code, substitute device pointers for vector and matrix arguments in all BLAS functions. To use these interfaces, existing applications 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 the operating-system-dependent type size_t, which is 32 bits wide on 32-bit platforms and 64 bits wide on a 64-bit platforms.

One approach to dealing with index arithmetic on device pointers in Fortran code is to use C-style macros and to use the C preprocessor to expand these, as shown below in Example A.2. On Linux and Mac OS X, preprocessing can be done using the `-E -x f77-cpp-input`
option with the g77 compiler, or simply using the '-cpp' option with g95 or gfortran. 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 254), 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 254).

The second example should be compiled with ARCH_64 defined as 1 on 64-bit operating systems and as 0 on 32-bit operating systems. For example, on g95 or gfortran it can be done directly on the command line by using the option '-cpp -DARCH_64=1'.

Example A.1. Fortran 77 Application Executing on the Host

```fortran
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
endo
do
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)
do
endo
do
write (*,*) ""
do
endo
stop
do
end
```

Example A.2. Fortran 77 Application Ported to Use CUBLAS

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

subroutine modify (devPtrM, ldm, n, p, q, alpha, beta)
```
Example A.2. Fortran 77 Application Ported to Use CUBLAS (continued)

```fortran
implicit none
integer sizeof_real
parameter (sizeof_real=4)
integer ldm, n, p, q
#if ARCH_64
  integer*8 devPtrM
#else
  integer devPtrM
#endif
real*4 alpha, beta
call cublas_sscal (n-p+1, alpha,
1                   devPtrM+IDX2F(p,q,ldm)*sizeof_real,
2                   ldm)
call cublas_sscal (ldm-p+1, beta,
1                   devPtrM+IDX2F(p,q,ldm)*sizeof_real,
2                   1)
return
end

program matrixmod
implicit none
integer M, N, sizeof_real
#if ARCH_64
  integer*8 devPtrA
#else
  integer devPtrA
#endif
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
  end do
end do
```
Example A.2. Fortran 77 Application Ported to Use CUBLAS (continued)

```
      enddo
      enddo

      call cublas_init
      stat = cublas_alloc(M*N, sizeof_real, devPtrA)
      if (stat .NE. 0) then
         write(*,*) "device memory allocation failed"
         call cublas_shutdown
         stop
      endif
      stat = cublas_set_matrix (M, N, sizeof_real, a, M, devPtrA, M)
      if (stat .NE. 0) then
         call cublas_free (devPtrA)
         write(*,*) "data download failed"
         call cublas_shutdown
         stop
      endif
      call modify (devPtrA, M, N, 2, 3, 16.0, 12.0)
      stat = cublas_get_matrix (M, N, sizeof_real, devPtrA, M, a, M)
      if (stat .NE. 0) then
         call cublas_free (devPtrA)
         write(*,*) "data upload failed"
         call cublas_shutdown
         stop
      endif
      call cublas_free (devPtrA)
      call cublas_shutdown
      do j = 1, N
         do i = 1, M
            write(*,"
            write (.*," 1", 16.0, 12.0)
      enddo
      stop
      end
```