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

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

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

Copyright
© 2005–2010 by NVIDIA Corporation. All rights reserved.
 Portions of the SGEMM, DGEMM and ZGEMM library routines were written by Vasily Volkov and are subject to the Modified Berkeley Software Distribution License as follows:

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

All rights reserved.

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

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

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

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

THIS SOFTWARE IS PROVIDED BY THE AUTHOR “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Portions of the SGEMM, DGEMM and ZGEMM library routines were written by Davide Barbieri and are subject to the Modified Berkeley Software Distribution License as follows:

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

All rights reserved.

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

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

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

3- The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

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

## 1. The CUBLAS Library

CUBLAS Types
- Type cublasStatus

CUBLAS Helper Functions
- Function cublasInit()
- Function cublasShutdown()
- Function cublasGetError()
- Function cublasAlloc()
- Function cublasFree()
- Function cublasSetVector()
- Function cublasGetVector()
- Function cublasSetMatrix()
- Function cublasGetMatrix()

## 2. BLAS1 Functions

**Single-Precision BLAS1 Functions**
- Function cublasIsamax()
- Function cublasIsamin()
- Function cublasSasum()
- Function cublasSaxpy()
- Function cublasScopy()
- Function cublasSdot()
- Function cublasSnrm2()
- Function cublasSrot()
- Function cublasSrotg()
- Function cublasSrotmg()
- Function cublasSscal()
- Function cublasSswap()

**Single-Precision Complex BLAS1 Functions**
- Function cublasCaxpy()
- Function cublasCcopy()
- Function cublasCdotc()
- Function cublasCdotu()
- Function cublasCrot()
Function cublasCrotg() ................................................................. 35
Function cublasCscale() ................................................................. 36
Function cublasCsrot() ................................................................. 36
Function cublasCsscal() ................................................................. 37
Function cublasCswap() ................................................................. 38
Function cublasCamax() ................................................................. 39
Function cublasCamin() ................................................................. 40
Function cublasScasum() ................................................................. 40
Function cublasScnrm2() ................................................................. 41

Double-Precision BLAS1 Functions ................................................. 43
Function cublasIdamax() ................................................................. 44
Function cublasIdamin() ................................................................. 44
Function cublasDasum() ................................................................. 45
Function cublasDaxpy() ................................................................. 46
Function cublasDcopy() ................................................................. 47
Function cublasDdot() ................................................................. 48
Function cublasDnrm2() ................................................................. 49
Function cublasDrot() ................................................................. 50
Function cublasDrotg() ................................................................. 51
Function cublasDrotm() ................................................................. 52
Function cublasDrotmg() ................................................................. 53
Function cublasDscal() ................................................................. 54
Function cublasDswap() ................................................................. 55

Double-Precision Complex BLAS1 functions .................................... 57
Function cublasDzasum() ................................................................. 58
Function cublasDznrm2() ................................................................. 59
Function cublasZamax() ................................................................. 59
Function cublasZamin() ................................................................. 60
Function cublasZaxpy() ................................................................. 61
Function cublasZcopy() ................................................................. 62
Function cublasZdotc() ................................................................. 63
Function cublasZdotu() ................................................................. 64
Function cublasZdrot() ................................................................. 65
Function cublasZdscal() ................................................................. 66
Function cublasZrot() ................................................................. 67
Function cublasZrotg() ................................................................. 68
Function cublasZscal() ................................................................. 68
Function cublasZswap() ................................................................. 69

3. Single-Precision BLAS2 Functions .............................................. 71
Single-Precision BLAS2 Functions .................................................. 72
Function cublasSgbmv() ................................................................. 73
Function cublasSgemv() ................................................................. 74
Function cublasSger() ................................................................. 75
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cublasDtbmv()</td>
<td>136</td>
</tr>
<tr>
<td>cublasDsyr2()</td>
<td>135</td>
</tr>
<tr>
<td>cublasDsyr()</td>
<td>134</td>
</tr>
<tr>
<td>cublasDsymv()</td>
<td>133</td>
</tr>
<tr>
<td>cublasDspr2()</td>
<td>132</td>
</tr>
<tr>
<td>cublasDspr()</td>
<td>131</td>
</tr>
<tr>
<td>cublasDspmv()</td>
<td>130</td>
</tr>
<tr>
<td>cublasDsbmv()</td>
<td>129</td>
</tr>
<tr>
<td>cublasDger()</td>
<td>128</td>
</tr>
<tr>
<td>cublasDgemv()</td>
<td>127</td>
</tr>
<tr>
<td>cublasDgbmv()</td>
<td>126</td>
</tr>
<tr>
<td>cublasCtrsv()</td>
<td>125</td>
</tr>
<tr>
<td>cublasCtrmv()</td>
<td>124</td>
</tr>
<tr>
<td>cublasCtpsv()</td>
<td>123</td>
</tr>
<tr>
<td>cublasCtpmv()</td>
<td>122</td>
</tr>
<tr>
<td>cublasCtbsv()</td>
<td>121</td>
</tr>
<tr>
<td>cublasCtbmv()</td>
<td>120</td>
</tr>
<tr>
<td>cublasChpr2()</td>
<td>119</td>
</tr>
<tr>
<td>cublasChpr()</td>
<td>118</td>
</tr>
<tr>
<td>cublasChpmv()</td>
<td>117</td>
</tr>
<tr>
<td>cublasCher2()</td>
<td>116</td>
</tr>
<tr>
<td>cublasCher()</td>
<td>115</td>
</tr>
<tr>
<td>cublasChemv()</td>
<td>114</td>
</tr>
<tr>
<td>cublasChbmv()</td>
<td>113</td>
</tr>
<tr>
<td>cublasCgeru()</td>
<td>112</td>
</tr>
<tr>
<td>cublasCgerc()</td>
<td>111</td>
</tr>
<tr>
<td>cublasCgemv()</td>
<td>110</td>
</tr>
<tr>
<td>cublasCgbmv()</td>
<td>109</td>
</tr>
<tr>
<td>cublasStbsv()</td>
<td>108</td>
</tr>
<tr>
<td>cublasStrsv()</td>
<td>107</td>
</tr>
<tr>
<td>cublasStpmv()</td>
<td>106</td>
</tr>
<tr>
<td>cublasStbsv()</td>
<td>105</td>
</tr>
<tr>
<td>cublasStrmv()</td>
<td>104</td>
</tr>
<tr>
<td>cublasStpmv()</td>
<td>103</td>
</tr>
<tr>
<td>cublasStbmv()</td>
<td>102</td>
</tr>
<tr>
<td>cublasSsbmv()</td>
<td>101</td>
</tr>
<tr>
<td>cublasSsymv()</td>
<td>100</td>
</tr>
<tr>
<td>cublasSspr2()</td>
<td>99</td>
</tr>
<tr>
<td>cublasSspr()</td>
<td>98</td>
</tr>
<tr>
<td>cublasSspmv()</td>
<td>97</td>
</tr>
<tr>
<td>cublasSsbmv()</td>
<td>96</td>
</tr>
<tr>
<td>cublasSsyr2()</td>
<td>95</td>
</tr>
<tr>
<td>cublasSsyr()</td>
<td>94</td>
</tr>
<tr>
<td>cublasSsymv()</td>
<td>93</td>
</tr>
<tr>
<td>cublasSspr()</td>
<td>92</td>
</tr>
<tr>
<td>cublasSspmv()</td>
<td>91</td>
</tr>
<tr>
<td>cublasSgbmv()</td>
<td>90</td>
</tr>
<tr>
<td>cublasSgbmv()</td>
<td>89</td>
</tr>
<tr>
<td>cublasSsymv()</td>
<td>88</td>
</tr>
</tbody>
</table>

4. Double-Precision BLAS2 Functions .............................................. 121
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cublasDtbsv()</td>
<td>138</td>
</tr>
<tr>
<td>cublasDtpmv()</td>
<td>140</td>
</tr>
<tr>
<td>cublasDtpsv()</td>
<td>141</td>
</tr>
<tr>
<td>cublasDtrmv()</td>
<td>142</td>
</tr>
<tr>
<td>cublasDtrsv()</td>
<td>144</td>
</tr>
<tr>
<td>Double-Precision Complex BLAS2 functions</td>
<td>146</td>
</tr>
<tr>
<td>cublasZgbmv()</td>
<td>147</td>
</tr>
<tr>
<td>cublasZgemv()</td>
<td>149</td>
</tr>
<tr>
<td>cublasZgerc()</td>
<td>150</td>
</tr>
<tr>
<td>cublasZgeru()</td>
<td>151</td>
</tr>
<tr>
<td>cublasZhemv()</td>
<td>153</td>
</tr>
<tr>
<td>cublasZher()</td>
<td>155</td>
</tr>
<tr>
<td>cublasZher2()</td>
<td>156</td>
</tr>
<tr>
<td>cublasZ跟她()</td>
<td>158</td>
</tr>
<tr>
<td>cublasZlep()</td>
<td>159</td>
</tr>
<tr>
<td>cublasZher()</td>
<td>161</td>
</tr>
<tr>
<td>cublasZher2()</td>
<td>162</td>
</tr>
<tr>
<td>cublasZhemv()</td>
<td>163</td>
</tr>
<tr>
<td>Single-Precision BLAS3 Functions</td>
<td>173</td>
</tr>
<tr>
<td>cublasSgemm()</td>
<td>175</td>
</tr>
<tr>
<td>cublasSsymm()</td>
<td>176</td>
</tr>
<tr>
<td>cublasSyrk()</td>
<td>178</td>
</tr>
<tr>
<td>cublasSyr2k()</td>
<td>180</td>
</tr>
<tr>
<td>cublasStrmm()</td>
<td>182</td>
</tr>
<tr>
<td>cublasStrsm()</td>
<td>184</td>
</tr>
<tr>
<td>Double-Precision BLAS3 Functions</td>
<td>206</td>
</tr>
<tr>
<td>cublasDgemm()</td>
<td>207</td>
</tr>
</tbody>
</table>

5. BLAS3 Functions ............................................................... 173
Function cublasDsymm() .............................................................. 208
Function cublasDsyrk() ............................................................ 210
Function cublasDsyrk2() ............................................................. 212
Function cublasDtrmm() ............................................................. 214
Function cublasDtrsm() ............................................................. 216

Double-Precision Complex BLAS3 Functions ........................................ 219
Function cublasZgemm() ............................................................. 220
Function cublasZhemm() ............................................................. 221
Function cublasZherk() ............................................................... 223
Function cublasZher2k() .............................................................. 226
Function cublasZsymm() ............................................................. 228
Function cublasZsyrk() ............................................................... 230
Function cublasZsym2k() ............................................................. 232
Function cublasZtrmm() ............................................................. 234
Function cublasZtrsm() ............................................................. 236

A. CUBLAS Fortran Bindings .......................................................... 239
CHAPTER 1

The CUBLAS Library

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

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

For maximum compatibility with existing Fortran environments, CUBLAS uses column-major storage and 1-based indexing. Since C and C++ use row-major storage, applications cannot use the native array semantics for two-dimensional arrays. Instead, macros or inline functions should be defined to implement matrices on top of one-dimensional arrays. For Fortran code ported to C in mechanical fashion, one may choose 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:

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

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 chose 0-based indexing, in which case the indexing macro becomes

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

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

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

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

Following these three examples, the remainder of this chapter discusses “CUBLAS Types” on page 8 and “CUBLAS Helper Functions” on page 9.
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-00000-002_V3.0

NVIDIA

3


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");
    cublasShutdown();
    return EXIT_FAILURE;
}
stat = cublasSetMatrix (M, N, sizeof(*a), a, M, devPtrA, M);
if (stat != CUBLAS_STATUS_SUCCESS) {
    printf("data download failed");
    cublasFree (devPtrA);
    cublasShutdown();
    return EXIT_FAILURE;
}
modify (devPtrA, M, N, 2, 3, 16.0f, 12.0f);
stat = cublasGetMatrix (M, N, sizeof(*a), devPtrA, M, a, M);
if (stat != CUBLAS_STATUS_SUCCESS) {
    printf("data upload failed");
    cublasFree (devPtrA);
    cublasShutdown();
    return EXIT_FAILURE;
}
cublasFree (devPtrA);
cublasShutdown();
for (j = 1; j <= N; j++) {
    for (i = 1; i <= M; i++) {
        printf("%7.0f", a[IDX2F(i,j,M)]);
    }
    printf("\n");
}
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)
{
    cublasScal (n-p, alpha, &m[IDX2C(p,q,ldm)], ldm);
    cublasScal (ldm-p, beta, &m[IDX2C(p,q,ldm)], ldm);
}

#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><code>CUBLAS_STATUS_SUCCESS</code></td>
<td>operation completed successfully</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_NOT_INITIALIZED</code></td>
<td>CUBLAS library not initialized</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ALLOC_FAILED</code></td>
<td>resource allocation failed</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_INVALID_VALUE</code></td>
<td>unsupported numerical value was passed to function</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_ARCH_MISMATCH</code></td>
<td>function requires an architectural feature absent from the architecture of the device</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_MAPPING_ERROR</code></td>
<td>access to GPU memory space failed</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_EXECUTION_FAILED</code></td>
<td>GPU program failed to execute</td>
</tr>
<tr>
<td><code>CUBLAS_STATUS_INTERNAL_ERROR</code></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 9
- “Function cublasShutdown()” on page 10
- “Function cublasGetError()” on page 10
- “Function cublasAlloc()” on page 10
- “Function cublasFree()” on page 11
- “Function cublasSetVector()” on page 11
- “Function cublasGetVector()” on page 12
- “Function cublasSetMatrix()” on page 13
- “Function cublasGetMatrix()” on page 13

Function cublasInit()

cublasStatus

cublasInit (void)

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

Return Values

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

```c
void *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** if CUBLAS library shut down successfully

Function **cublasGetError()**

```c
void *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
int *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 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()

CUBLASStatus

cublasFree (const void *devicePtr)

destructors 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()

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 elemSize bytes. Storage spacing between consecutive elements is incx for the source vector x and incy for the destination vector y. In general, y points to an object, or part of an object, allocated via cublasAlloc(). Column-major format for two-dimensional matrices is assumed throughout CUBLAS. If the vector is part of a matrix, a vector increment equal to 1 accesses a (partial) column of the matrix.
Similarly, using an increment equal to the leading dimension of the matrix accesses a (partial) row.

Return Values

<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>
<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()

```c

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 \( \text{elemSize} \) bytes. Storage spacing between consecutive elements is \( \text{incx} \) for the source vector \( x \) and \( \text{incy} \) for the destination vector \( y \). In general, \( x \) 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>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 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>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 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()`.

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>
CHAPTER 2

BLAS1 Functions

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

- “Single-Precision BLAS1 Functions” on page 16
- “Single-Precision Complex BLAS1 Functions” on page 29
- “Double-Precision BLAS1 Functions” on page 43
- “Double-Precision Complex BLAS1 functions” on page 57
Single-Precision BLAS1 Functions

The single-precision BLAS1 functions are as follows:

- “Function cublasIsamax()” on page 17
- “Function cublasIsamin()” on page 17
- “Function cublasSasum()” on page 18
- “Function cublasSaxpy()” on page 19
- “Function cublasScopy()” on page 20
- “Function cublasSdot()” on page 21
- “Function cublasSnrm2()” on page 22
- “Function cublasSrot()” on page 22
- “Function cublasSrotg()” on page 23
- “Function cublasSrotm()” on page 24
- “Function cublasSrotmg()” on page 26
- “Function cublasSscal()” on page 27
- “Function cublasSswap()” on page 28
CHAPTER 2

BLAS1 Functions

Function cublasIsamax()

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

Input

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

Output

returns the single-precision sum of absolute values
(returns zero if \( n \leq 0 \) or \( \text{incx} \leq 0 \), or if an error occurred)

Reference: http://www.netlib.org/blas/sasum.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_ALLOC_FAILED</td>
<td>if function could not allocate reduction buffer</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Output

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

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

where

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

\( ly \) 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

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

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 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

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 *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

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

where

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

\( 1y \) is defined in a similar way using \( incy \).

**Input**

- \( n \) number of elements in input vectors
- \( x \) single-precision 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()

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

computes the Euclidean norm of the single-precision \(n\)-vector \(x\) (with storage increment \(\text{incx}\)). 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
- \(\text{incx}\) storage spacing between elements of \(x\)

Output

returns the Euclidean norm

(returns zero if \(n \leq 0\), \(\text{incx} \leq 0\), 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 \text{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_EXECUTION_FAILED} if function failed to launch on GPU

Function cublasSrot()

```c
void
cublasSrot (int n, float *x, int incx, float *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 \(x[1x + i * \text{incx}], i = 0\) to \(n - 1\), where

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

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 $y$  
$sc$  element of rotation matrix  
$ss$  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/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 $[sa \ sb]^T$.  

---

**CHAPTER 2**

**BLAS1 Functions**

---

**NVIDIA**

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

- If \( z = 1 \), set \( sc = 0.0 \) and \( ss = 1.0 \).
- If \( \text{abs}(z) < 1 \), set \( sc = \sqrt{1 - z^2} \) and \( ss = z \).
- If \( \text{abs}(z) > 1 \), set \( sc = 1/z \) and \( ss = \sqrt{1 - sc^2} \).

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

**Input**

- \( sa \) 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×n matrix \[
\begin{bmatrix}
x^T \\
y^T
\end{bmatrix}
\]

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

- \( 1x = 1 \) if \( \text{incx} \geq 0 \), 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:

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

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: \text{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} if CUBLAS library was not initialized
- \text{CUBLAS\_STATUS\_EXECUTION\_FAILED} if function failed to launch on GPU
Function `cublasSrotmg()`

```c
void

cublasSrotmg (float *host_sd1, float *host_sd2,
    float *host_sx1, const float *host_sy1,
    float *host_sparam)
```

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

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

```plaintext
sflag = -1.0f  sflag = 0.0f
\[ h = \begin{bmatrix} sh00 & sh01 \\ sh10 & sh11 \end{bmatrix} \]
\[ h = \begin{bmatrix} 1.0f & sh01 \\ sh10 & 1.0f \end{bmatrix} \]

sflag = 1.0f  sflag = -2.0f
\[ h = \begin{bmatrix} sh00 & 1.0f \\ -1.0f & sh11 \end{bmatrix} \]
\[ h = \begin{bmatrix} 1.0f & 0.0f \\ 0.0f & 1.0f \end{bmatrix} \]
```

`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.
- `sy1` single-precision scalar.

**Output**

- `sd1` changed to represent the effect of the transformation.
- `sd2` changed to represent the effect of the transformation.
- `sx1` changed to represent the effect of the transformation.

---

NVIDIA

PG-00000-002_V3.0
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 * x. For
i = 0 to n-1, it replaces
\[
x[lx + i * incx] \text{ with } \alpha x[lx + i * incx],
\]
where
\[
lx = 1 \text{ if } incx >= 0, \text{else}
lx = 1 + (1 - n) * incx.
\]

**Input**

- **n**: number of elements in input vector
- **alpha**: 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 <= 0 or incx <= 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

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

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` 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 <= 0`)
- `y` single-precision vector `x` (unchanged from input if `n <= 0`)

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

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

**Error Status**
- `CUBLAS_STATUS_NOT_INITIALIZED` if CUBLAS library was not initialized
- `CUBLAS_STATUS_EXECUTION_FAILED` if function failed to launch on GPU
Single-Precision Complex BLAS1 Functions

The single-precision complex BLAS1 functions are as follows:

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

```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

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

where

\[
l x = \begin{cases} 
0 & \text{if incx } \geq 0, \\
1 & \text{if incx } < 0
\end{cases}
\]

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

Input

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

<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_EXECUTION_FAILED</code></td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
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 * incx] to y[ly+i * incy],
```

where

```
lx = 1 if incx >= 0, 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
- `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()

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
\[ \text{x}[lx + i \ast \text{incx}] \ast \text{y}[ly + i \ast \text{incy}], \]
where
\[ lx = 1 \text{ if incx} \geq 0, \text{ else} \]
\[ lx = 1 + (1 \ast \text{n}) \ast \text{incx}; \]
ly is defined in a similar way using incy.

Input

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>number of elements in input vectors</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>
<tr>
<td>y</td>
<td>single-precision complex vector with n elements</td>
</tr>
<tr>
<td>incy</td>
<td>storage spacing between elements of y</td>
</tr>
</tbody>
</table>

Output

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

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

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

Error Status

<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_ALLOC_FAILED</td>
<td>if function could not allocate reduction buffer</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
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

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

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

Input

\( n \) \hspace{1em} number of elements in input vectors
\( x \) \hspace{1em} single-precision complex vector with \( n \) elements
\( incx \) \hspace{1em} storage spacing between elements of \( x \)
\( y \) \hspace{1em} single-precision complex vector with \( n \) elements
\( incy \) \hspace{1em} 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 2×2 matrix \[
\begin{bmatrix}
sc & cs \\
-cs & sc
\end{bmatrix}
\] with the 2×n matrix \[
\begin{bmatrix}
x^T \\
y^T
\end{bmatrix}
\].

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

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

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
- **cs**: single-precision complex sine component of rotation matrix

Output

- **x**: rotated vector x (unchanged if n <= 0)
- **y**: rotated vector y (unchanged if n <= 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 \(\frac{ca}{|ca|}||ca, \ cb||\) overwrites \(ca\) in storage. In this case,

\[
||ca, \ cb|| = \text{scale} \cdot \sqrt{\left|\frac{ca}{\text{scale}}\right|^2 + \left|\frac{cb}{\text{scale}}\right|^2}, \quad \text{where}
\]

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

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 \(\frac{ca}{|ca|}||ca, \ cb||\)
- `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 \( x \) with single-precision complex \( \alpha \times x \).

For \( i = 0 \) to \( n-1 \), it replaces
\[
x[lx+i*incx] \text{ with } \alpha \times x[lx+i*incx],
\]

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

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

Output
- \( x \): single-precision complex result (unchanged if \( n <= 0 \) or \( incx <= 0 \))

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

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

\( y \) 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 \texttt{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
bublasCsscal (int n, float alpha, cuComplex *x, int incx)
```

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

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

where

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

NVIDIA

CUDA CUBLAS Library

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

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

where

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

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

Input

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>number of elements in input vectors</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>
<tr>
<td>( y )</td>
<td>single-precision complex vector with ( n ) elements</td>
</tr>
<tr>
<td>incy</td>
<td>storage spacing between elements of ( y )</td>
</tr>
</tbody>
</table>

Output

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

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

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

Error Status

- CUBLAS_STATUS_NOT_INITIALIZED if CUBLAS library was not initialized
- CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU
CHAPTER 2 BLAS1 Functions

Output

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>single-precision complex vector y (unchanged from input if ( n \leq 0 ))</td>
</tr>
<tr>
<td>y</td>
<td>single-precision complex vector x (unchanged from input if ( n \leq 0 ))</td>
</tr>
</tbody>
</table>

Reference: [http://www.netlib.org/blas/cswap.f](http://www.netlib.org/blas/cswap.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_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function `cublasIcamax()`

```c
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 \( \text{abs}(x[1+i*incx]) \). The result reflects 1-based indexing for compatibility with Fortran.

Input

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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 \( n \leq 0 \) or \( \text{incx} \leq 0 \))

Reference: [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

<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_ALLOC_FAILED</td>
<td>if function could not allocate reduction buffer</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
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 \( \text{abs}(x[1+i\times\text{incx}]) \). 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

**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

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

where
- \( lx = 1 \) if `incx <= 0`, else
- \( lx = 1+(1-n)\times\text{incx} \).
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

floatScnrm2 (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()`.  

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_ALLOC_FAILED</td>
<td>if function could not allocate</td>
</tr>
<tr>
<td></td>
<td>reduction buffer</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
CHAPTER 2

BLAS1 Functions

Double-Precision BLAS1 Functions

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

The double-precision BLAS1 functions are as follows:

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

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

finds the smallest index of the maximum magnitude element of
double-precision vector 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>n</th>
<th>number of elements in input vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>double-precision 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/idamax.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_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 cublasIdamin()

    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 abs(x[1 + i * incx]). The result reflects 1-based
indexing for compatibility with Fortran.
CHAPTER 2

BLAS1 Functions

Input

\begin{itemize}
\item \textbf{n} \quad \text{number of elements in input vector}
\item \textbf{x} \quad \text{double-precision vector with } n \text{ elements}
\item \textbf{incx} \quad \text{storage spacing between elements of } x
\end{itemize}

Output

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

Analogous to \url{http://www.netlib.org/blas/idamax.f}

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

Error Status

\begin{itemize}
\item \texttt{CUBLAS_STATUS_NOT_INITIALIZED} \quad \text{if CUBLAS library was not initialized}
\item \texttt{CUBLAS_STATUS_ALLOC_FAILED} \quad \text{if function could not allocate reduction buffer}
\item \texttt{CUBLAS_STATUS_ARCH_MISMATCH} \quad \text{if function invoked on device that does not support double precision}
\item \texttt{CUBLAS_STATUS_EXECUTION_FAILED} \quad \text{if function failed to launch on GPU}
\end{itemize}

\textbf{Function cublasDasum()}

\begin{verbatim}
double
\text{cublasDasum (int n, const double *x, int incx)}
\end{verbatim}

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

Input

\begin{itemize}
\item \textbf{n} \quad \text{number of elements in input vector}
\item \textbf{x} \quad \text{double-precision vector with } n \text{ elements}
\item \textbf{incx} \quad \text{storage spacing between elements of } x
\end{itemize}

Output

\begin{itemize}
\item \text{returns the double-precision sum of absolute values (returns zero if } n \leq 0 \text{ or } \text{incx} \leq 0, \text{ or if an error occurred)}
\end{itemize}

Reference: \url{http://www.netlib.org/blas/dasum.f}
Error status for this function can be retrieved via `cublasGetError()`.

### Error Status

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

### Function cublasDaxpy()

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

This function 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

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

where

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

`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`)
CHAPTER 2  BLAS1 Functions

Reference: http://www.netlib.org/blas/daxpy.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_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 cublasDcopy()

```c
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

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

where

\[
lx = 1 \text{ if } incx \geq 0,
\]

\[
lx = 1 + (1-n) \times incx;
\]

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

Input

<table>
<thead>
<tr>
<th>n</th>
<th>number of elements in input vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>double-precision vector with ( n ) elements</td>
</tr>
<tr>
<td>incx</td>
<td>storage spacing between elements of ( x )</td>
</tr>
<tr>
<td>y</td>
<td>double-precision vector with ( n ) elements</td>
</tr>
<tr>
<td>incy</td>
<td>storage spacing between elements of ( y )</td>
</tr>
</tbody>
</table>

Output

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

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

Error Status

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

### Function cublasDdot()

```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

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

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>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_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 `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

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

Output

returns the 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>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_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 cublasDrot()

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

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

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

$lx = 1$ if incx >= 0, 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 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 $n <= 0$)
- $y$ : rotated vector $y$ (unchanged if $n <= 0$)

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
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:

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

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

**Input**

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

**Output**

- \(da\) double-precision value
- \(db\) double-precision value
- \(dc\) 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()

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

applies the modified Givens transformation, h, to the 2×n matrix

\[
\begin{bmatrix}
  x^T \\
y^T
\end{bmatrix}
\]

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

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

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.

Output

- x: rotated vector x (unchanged if n <= 0)
- y: rotated vector y (unchanged if n <= 0)
CHAPTER 2  BLAS1 Functions

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

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

Error Status

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

Function `cublasDrotmg()`

```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 `(dd1*dx1, 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
CUDA CUBLAS Library

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

This function does not set any error status.

Function cublasDscal()

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

replaces double-precision vector `x` with double-precision `alpha` \* `x`. For `i = 0` to `n-1`, it replaces

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

where

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

Input

- `n`: number of elements in input vector
- `alpha`: double-precision scalar multiplier
- `x`: double-precision 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
Error status for this function can be retrieved via \texttt{cublasGetError()}.  

\textbf{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_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>

\textbf{Function cublasDswap()}

\begin{verbatim}
void cublasDswap (int n, double *x, int incx, double *y, int incy)
\end{verbatim}

interchanges double-precision vector $x$ with double-precision vector $y$. For $i = 0$ to $n-1$, it interchanges
\[ x[1x+i*incx] \text{ with } y[1y+i*incy], \]

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

$1y$ is defined in a similar manner using $incy$.

\textbf{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 $y$                  |

\textbf{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: \url{http://www.netlib.org/blas/dswap.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><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>
Double-Precision Complex BLAS1 functions

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

The double-precision complex BLAS1 functions are listed below:

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

double

cublasDzasum (int n, const cuDoubleComplex *x, int incx)
takes the sum of the absolute values of a complex vector and returns a
double-precision result. Note that this is not the L1 norm of the vector.
The result is the sum from 0 to n-1 of

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

where

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

Input

\[ \begin{align*}
\text{n} & \quad \text{number of elements in input vector} \\
\text{x} & \quad \text{double-precision complex vector with n elements} \\
\text{incx} & \quad \text{storage spacing between elements of x}
\end{align*} \]

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

<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_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>


Chapter 2  BLAS1 Functions

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

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

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 \( n \leq 0 \) or \( \text{incx} \leq 0 \))

Reference: [http://www.netlib.org/blas/izamax.f](http://www.netlib.org/blas/izamax.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_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 `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 \( \text{abs}(\text{real}(x[i+i*\text{incx}])) + \text{abs}(\text{imag}(x[i+i*\text{incx}])) \).

Input

- \( n \) number of elements in input vector
- \( x \) double-precision complex 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: analogous to “Function `cublasIzamax()`” on page 59.

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_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 cublasZaxpy()

```c
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

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

where

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

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

Input

- \( n \): number of elements in input vectors
- \( \alpha \): double-precision complex scalar multiplier
- \( x \): 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

- \( y \): 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

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

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** 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**

<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_EXECUTION_FAILED</code></td>
<td>if function failed to launch on GPU</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>
</tbody>
</table>
Function cublasZdotc()

cuDoubleComplex

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

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

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 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 \texttt{cublasGetError()}.

Error Status

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

\texttt{CUBLAS\_STATUS\_ALLOC\_FAILED} if function could not allocate
reduction buffer

\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 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

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

where

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

$ly$ is defined in a similar way using $incy$.

**Input**

- $n$: number of elements in input vectors
- $x$: 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()

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

multiplies a 2x2 matrix \[
\begin{bmatrix}
c & s \\
-s & c 
\end{bmatrix}
\] with the 2x\(n\) 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
\[
1x = 1 \ if \ incx >= 0, \ else \\
1x = 1 + (1-n) \times incx;
\]
y is treated similarly using \(ly\) and \(incy\).

Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>number of elements in input vectors</td>
</tr>
<tr>
<td>(x)</td>
<td>double-precision complex vector with (n) elements</td>
</tr>
<tr>
<td>(incx)</td>
<td>storage spacing between elements of (x)</td>
</tr>
<tr>
<td>(y)</td>
<td>double-precision complex vector with (n) elements</td>
</tr>
<tr>
<td>(incy)</td>
<td>storage spacing between elements of (y)</td>
</tr>
<tr>
<td>(c)</td>
<td>double-precision cosine component of rotation matrix</td>
</tr>
<tr>
<td>(s)</td>
<td>double-precision sine component of rotation matrix</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>rotated vector (x) (unchanged if (n &lt;= 0))</td>
</tr>
<tr>
<td>(y)</td>
<td>rotated vector (y) (unchanged if (n &lt;= 0))</td>
</tr>
</tbody>
</table>

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

<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 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

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

where

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

**Input**

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

**Output**

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

**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 \(2\times2\) 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 \(x[lx + i \times \text{incx}], i = 0 \text{ to } n-1\), 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
- \(incx\): storage spacing between elements of \(x\)
- \(y\): double-precision complex vector with \(n\) elements
- \(incy\): storage spacing between elements of \(y\)
- \(sc\): double-precision cosine component of rotation matrix
- \(cs\): double-precision complex sine component of rotation matrix

**Output**

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

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

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

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

Input

- \(ca\) double-precision complex scalar
- \(cb\) double-precision complex scalar

Output

- \(ca\) double-precision complex \(ca/|ca||ca, \ cb||\)
- \(sc\) double-precision cosine component of rotation matrix
- \(cs\) double-precision complex sine component of rotation matrix

Reference: [http://www.netlib.org/blas/zrotg.f](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 \(alpha * x\).
CHAPTER 2

BLAS1 Functions

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

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

where

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

Input

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

Output

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

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 \texttt{cublasZswap()}

\[
\text{void}
\text{cublasZswap} \ (\text{int} \ n, \text{cuDoubleComplex} \ *x, \ \text{int} \ \text{incx},
\text{cuDoubleComplex} \ *y, \ \text{int} \ \text{incy})
\]

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

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

where

\[ \text{lx} = 1 \text{ if incx} \geq 0, \text{else} \]
\[ \text{lx} = 1 + (1 - n) \cdot \text{incx}; \]
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 72
- “Single-Precision Complex BLAS2 Functions” on page 95
Single-Precision BLAS2 Functions

The single-precision BLAS2 functions are as follows:

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

```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 `trans` == 'N' or 'n', \( \text{op}(A) = A \).
- If `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 \((lda, n)\). 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 \((1 + (n - 1) \cdot \text{abs}(incx))\) when `trans` == 'N' or 'n', and at least \((1 + (m - 1) \cdot \text{abs}(incx))\) otherwise.
- `incx` storage spacing between elements of \( x \); \( incx \) must not be zero.
- `beta` single-precision scalar multiplier applied to vector \( y \). If `beta` is zero, \( y \) is not read.
Function cublasSgemv()

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

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

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

\( \alpha \) 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

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

Input (continued)

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

Output

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

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 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

\[ A = \alpha \times x \times y^T + A, \]
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 \( x \) and \( y \).
- \( x \): single-precision array of length at least \((1 + (m-1) \cdot \text{abs(incx)})\).
- \( \text{incx} \): the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- \( y \): single-precision array of length at least \((1 + (n-1) \cdot \text{abs(incy)})\).
- \( \text{incy} \): 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**

- **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 \( A \times x \).
- \( A \) 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 \( y = \alpha \times A \times x + \beta \times y \).
Reference: [http://www.netlib.org/blas/ssbmv.f](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 $k &lt; 0$, $n &lt; 0$, $\text{incx} = 0$, 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 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

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

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 AR. If $\text{uplo} = 'U'$ or 'u', the upper triangular part of $A$ is supplied in AR. If $\text{uplo} = 'L'$ or 'l', the lower triangular part of $A$ is supplied in AR.
- **n**: the number of rows and columns of matrix $A$; $n$ must be at least zero.
- **alpha**: single-precision scalar multiplier applied to $A \times x$.
- **AP**: single-precision array with at least $(n \times (n + 1))/2$ elements. If $\text{uplo} = 'U'$ 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 $\text{AP}[i + (j \times (j+1)/2)]$. If $\text{uplo} = 'L'$ 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 $\text{AP}[i + ((2 \times n-j+1) \times j)/2]$.
- **x**: single-precision array of length at least $(1 + (n-1) \times \text{abs(incx)})$.
- **incy**: storage spacing between elements of $x$; incy must not be zero.
CHAPTER 3 Single-Precision BLAS2 Functions

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

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

Input

- `uplo`: specifies whether the matrix data is stored in the upper or the lower triangular part of array `AP`. If `uplo` is `'U'` or `'u'`, the upper triangular part of \( A \) is supplied in `AP`. If `uplo` is `'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 \( x^T x \).
- `x`: single-precision array of length at least \((1+(n-1)\cdot\text{abs}(\text{incy}))\).
Function `cublasSspr2()`

```c
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

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

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.

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

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

Error Status

- `CUBLAS_STATUS_NOT_INITIALIZED` if CUBLAS library was not initialized
- `CUBLAS_STATUS_INVALID_VALUE` if \( n < 0 \) or \( \text{incx} == 0 \)
- `CUBLAS_STATUS_EXECUTION_FAILED` if function failed to launch on GPU
### 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 \( A \) 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 scalar multiplier applied to \( x \cdot y^T + \alpha \cdot y \cdot x^T \).

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

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

**y**  
single-precision array of length at least \( (1 + (n - 1) \cdot \text{abs}(\text{incy})) \).

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

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

### Output

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

### 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

- \(\text{uplo}\) specifies whether the upper or lower triangular part of the array \(A\) is referenced. If \(\text{uplo} == 'U'\) 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 \(\text{uplo} == 'L'\) 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.

- \(\text{n}\) 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 \(A x\).

- \(A\) single-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 part of the symmetric 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 symmetric matrix, and the strictly upper triangular part of \(A\) is not referenced.

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

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

- \(\text{incx}\) 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

Input (continued)

<table>
<thead>
<tr>
<th>Y</th>
<th>single-precision array of length at least ((1+(n-1) \cdot \text{abs}(\text{incy}))).</th>
</tr>
</thead>
<tbody>
<tr>
<td>incy</td>
<td>storage spacing between elements of y; incy must not be zero.</td>
</tr>
</tbody>
</table>

Output

| y       | updated according to \(y = \alpha \cdot A \cdot x + \beta \cdot y\).                  |

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

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

Error Status

| CUBLAS_STATUS_NOT_INITIALIZED    | if CUBLAS library was not initialized                                             |
| CUBLAS_STATUS_INVALID_VALUE     | if \(n < 0\), \(\text{incx} == 0\), or \(\text{incy} == 0\)                         |
| CUBLAS_STATUS_EXECUTION_FAILED  | if function failed to launch on GPU                                              |

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

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

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

Input

<table>
<thead>
<tr>
<th>uplo</th>
<th>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'}), only the upper triangular part of (A) is referenced. If (\text{uplo} == \text{'L'}) or (\text{'l'}), only the lower triangular part of (A) is referenced.</th>
</tr>
</thead>
<tbody>
<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 \cdot x^T).</td>
</tr>
<tr>
<td>x</td>
<td>single-precision array of length at least ((1+(n-1) \cdot \text{abs}(\text{incx}))).</td>
</tr>
</tbody>
</table>
CUDA CUBLAS Library

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 \( \text{incx} = 0 \)
- `CUBLAS_STATUS_EXECUTION_FAILED` if function failed to launch on GPU

**Function cublasSsyr2()**

```c
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

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

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 uplo == 'U' or 'u', only the upper triangular part of A is referenced and the lower triangular part of A is inferred. If uplo == 'L' or 'l', only the lower triangular part of A is referenced and the upper triangular part of A is inferred.

- **n**: The number of rows and columns of matrix A; n must be at least zero.

- **alpha**: Single-precision scalar multiplier applied to \( x \cdot y^T + y \cdot x^T \).

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

- **incx**: Storage spacing between elements of x; incx must not be zero.

- **y**: Single-precision array of length at least \( (1 + (n - 1) \cdot \text{abs}(\text{incy})) \).

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

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

- **lda**: Leading dimension of A; lda must be at least \( \max(1, n) \).

### Output

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

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

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

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

\( \mathbf{x} \) is an \( n \)-element single-precision vector, and \( \mathbf{A} \) 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 \( \mathbf{A} \) is an upper or lower triangular band matrix. If \( \text{uplo} == \text{‘U’} \) or \( \text{‘u’} \), \( \mathbf{A} \) is an upper triangular band matrix. If \( \text{uplo} == \text{‘L’} \) or \( \text{‘l’} \), \( \mathbf{A} \) is a lower triangular band matrix.
- **trans** specifies \( \text{op}(\mathbf{A}) \). If \( \text{trans} == \text{‘N’} \) or \( \text{‘n’} \), \( \text{op}(\mathbf{A}) = \mathbf{A} \). If \( \text{trans} == \text{‘T’} \), \( \text{‘t’} \), \( \text{‘C’} \), or \( \text{‘c’} \), \( \text{op}(\mathbf{A}) = \mathbf{A}^T \).
- **diag** specifies whether or not matrix \( \mathbf{A} \) is unit triangular. If \( \text{diag} == \text{‘U’} \) or \( \text{‘u’} \), \( \mathbf{A} \) is assumed to be unit triangular. If \( \text{diag} == \text{‘N’} \) or \( \text{‘n’} \), \( \mathbf{A} \) is not assumed to be unit triangular.
- **n** specifies the number of rows and columns of the matrix \( \mathbf{A} \); \( n \) must be at least zero.
- **k** specifies the number of superdiagonals or subdiagonals. If \( \text{uplo} == \text{‘U’} \) or \( \text{‘u’} \), \( k \) specifies the number of superdiagonals. If \( \text{uplo} == \text{‘L’} \) or \( \text{‘l’} \), \( k \) specifies the number of subdiagonals; \( k \) must at least be zero.
- **A** single-precision array of dimension \((\text{lda}, n)\). If \( \text{uplo} == \text{‘U’} \) or \( \text{‘u’} \), the leading \((k+1)\times n\) part of the array \( \mathbf{A} \) 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 \( \mathbf{A} \) is not referenced. If \( \text{uplo} == \text{‘L’} \) or \( \text{‘l’} \), the leading \((k+1)\times n\) part of the array \( \mathbf{A} \) 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

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

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

Error Status

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

Function cublasStbsv()

```c
void
cublasStbsv (char uplo, char trans, char diag, int n, int k, const float *A, int lda, 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 vectors, and \( A \) is an \( n \times n \), unit or non-unit, upper or lower, triangular band matrix with \( k+1 \) diagonals.

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

- **uplo** specifies whether the matrix is an upper or lower triangular band matrix. If `uplo == 'U'` or `u`, `A` is an upper triangular band matrix. If `uplo == 'L'` or `l`, `A` is a lower triangular band matrix.

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

- **diag** specifies whether `A` is unit triangular. If `diag == 'U'` or `u`, `A` is assumed to be unit triangular; that is, diagonal elements are not read and are assumed to be unity. If `diag == 'N'` or `n`, `A` is not assumed to be unit triangular.

- **n** the number of rows and columns of matrix `A`; `n` must be at least zero. In the current implementation `n` must not exceed 4070.

- **k** specifies the number of superdiagonals or subdiagonals.
  
  If `uplo == 'U'` or `u`, `k` specifies the number of superdiagonals. If `uplo == 'L'` or `l`, `k` specifies the number of subdiagonals; `k` must be at least zero.

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

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

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

**Output**

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

Reference: [http://www.netlib.org/blas/stbsv.f](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 <code>incx == 0</code>, <code>n &lt; 0</code>, or <code>n &gt; 4070</code></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 \( \text{op}(A) = A \) 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()

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. In the current implementation n must not exceed 4070.

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

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

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

Output

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

Reference: http://www.netlib.org/blas/tpsv.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, n &lt; 0, or n &gt; 4070</td>
</tr>
<tr>
<td>CUBLAS_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
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 \( \text{op}(A) = A^T \),

\( x \) 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 \( \text{uplo} = 'U' \) or \( 'u' \), \( A \) is an upper triangular matrix.
  - If \( \text{uplo} = 'L' \) or \( 'l' \), \( A \) is an 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 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 \( 'u' \), 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** single-precision 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** 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

<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 incx == 0 or n &lt; 0</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 `cublasStrsv()`

```c
void

\[
\text{cublasStrsv}\left(\text{char uplo, char trans, char diag, int n, const float *A, int lda, float *x, int incx}\right)
\]

``` solves a system of equations

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

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

\( b \) 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 \( \text{lda} \) 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 \( \text{uplo} = 'U' \text{ or } 'u' \), only the upper triangular part of \( A \) may be referenced. If \( \text{uplo} = 'L' \text{ or } 'l' \), only the lower triangular part of \( A \) may be referenced.

`trans` specifies \( \text{op}(A) \). If \( \text{trans} = 'N' \text{ or } 'n' \), \( \text{op}(A) = A \).
If \( \text{trans} = 'T', 't', 'C', \text{ or } 'c' \), \( \text{op}(A) = \overline{A^T} \).
CUDA CUBLAS Library

Input (continued)

diag specifies whether or not \( A \) is a unit triangular matrix.
- If \( diag = \text{'U'} \) or \text{'u'} \, \( A \) is assumed to be unit triangular.
- If \( diag = \text{'N'} \) 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.
A single-precision array of dimensions \((\text{lda}, n)\). If \( \text{uplo} = \text{'U'} \) or \text{'u'}, \( A \) contains the upper triangular part of the symmetric matrix, and the strictly lower triangular part is not referenced. If \( \text{uplo} = \text{'L'} \) or \text{'l'}, \( A \) contains the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.
lda leading dimension of the two-dimensional array containing \( A \); lda must be at least \( \max(1, n) \).
\( x \) single-precision 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 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/strsv.f](http://www.netlib.org/blas/strsv.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 incx == 0 or \( n < 0 \)
- \text{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 96
- “Function cublasCgemv()” on page 97
- “Function cublasCgerc()” on page 99
- “Function cublasCgeru()” on page 100
- “Function cublasChbmv()” on page 101
- “Function cublasChemv()” on page 103
- “Function cublasCher()” on page 104
- “Function cublasCher2()” on page 105
- “Function cublasChpmv()” on page 107
- “Function cublasChpr()” on page 108
- “Function cublasChpr2()” on page 109
- “Function cublasCtbmv()” on page 111
- “Function cublasCtbsv()” on page 113
- “Function cublasCtpmv()” on page 114
- “Function cublasCtpsv()” on page 116
- “Function cublasCtrmv()” on page 117
- “Function cublasCtrsv()” on page 119
Function cublasCgbmv()

```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

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

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

\[
\text{alpha} \quad \text{and} \quad \text{beta} \quad \text{are single-precision complex scalars, and} \quad \text{x} \quad \text{and} \quad \text{y} \quad \text{are single-precision complex vectors.} \quad A \quad \text{is an} \quad m \times n \quad \text{band matrix consisting of single-precision complex elements with} \quad kl \quad \text{subdiagonals and} \quad ku \quad \text{superdiagonals.}
\]

**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.
- `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 \((\text{kl+ku+1}) \times n\) 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 \( ku \), 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 \( ku \times ku \) triangle) are not referenced.
- `lda` leading dimension \( A \); `lda` must be at least \((\text{kl+ku+1})\).
CHAPTER 3 Single-Precision BLAS2 Functions

Input (continued)

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

\[ \text{incx} \quad \text{specifies the increment for the elements of } x; \text{ incx must not be zero.} \]

\[ \text{beta} \quad \text{single-precision complex scalar multiplier applied to vector } y. \text{ If } \text{beta} \]
\[ \text{is zero, } y \text{ is not read.} \]

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

\[ \text{incy} \quad \text{on entry, incy specifies the increment for the elements of } y; \text{ incy} \]
\[ \text{must not be zero.} \]

Output

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

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

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

Error Status

- **CUBLAS_STATUS_NOT_INITIALIZED** if CUBLAS library was not initialized
- **CUBLAS_STATUS_INVALID_VALUE** if \( m < 0, n < 0, \text{incx} == 0, \text{ or } \text{incy} == 0 \)
- **CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU

Function \texttt{cublasCgemv}()

\[
\text{void cublasCgemv (char trans, int m, int n,} \]
\[
cuComplex alpha, const cuComplex *A,} \]
\[
\text{int lda, const cuComplex *x, int incx,} \]
\[
cuComplex beta, cuComplex *y, int incy)}
\]

performs one of the matrix-vector operations

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

where \( \text{op}(A) = A, \text{op}(A) = A^T, \text{ or } \text{op}(A) = A^H; \)
alpha and beta are single-precision complex scalars; and x and y are single-precision complex vectors. A is 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 single-precision complex scalar multiplier applied to op(A).
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 leading dimension of two-dimensional array used to store matrix A.
x single-precision complex array of length at least
(1 + (n - 1) * abs(incx)) if trans == 'N' or 'n', else at least
(1 + (m - 1) * abs(incx)).
incx specifies the storage spacing for elements of x; incx must not be zero.
beta single-precision complex scalar multiplier applied to vector y. If beta is zero, y is not read.
y 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 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
CHAPTER 3  Single-Precision BLAS2 Functions

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_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

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

where $\alpha$ is a single-precision complex scalar, $x$ is an $m$-element single-precision complex vector, $y$ is an $n$-element single-precision complex vector, and $A$ is an $m \times n$ matrix consisting of single-precision complex elements. Matrix $A$ is stored in column-major format, and $\text{lda}$ 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 complex scalar multiplier applied to $x * y^H$.
- $x$ single-precision complex array of length at least $(1 + (m-1) * \text{abs(incx)})$.
- $\text{incx}$ the storage spacing between elements of $x$; $\text{incx}$ must not be zero.
- $y$ single-precision complex array of length at least $(1 + (n-1) * \text{abs(incy)})$.
- $\text{incy}$ the storage spacing between elements of $y$; $\text{incy}$ must not be zero.
- $A$ single-precision complex array of dimensions $(\text{lda}, n)$.
- $\text{lda}$ 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 \(x \times y^T\).
- \(x\): 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

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 * x * y^T + A.</td>
</tr>
</tbody>
</table>

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

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

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×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 uplo == 'U' or 'u', the upper triangular part is being supplied. If 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 \(\text{u}'\), 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 \(\text{l}'\), 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 \(y = \text{alpha} \times A \times x + \text{beta} \times y\).

### Reference

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

- **CUBLAS_STATUS_EXECUTION_FAILED**: if function failed to launch on GPU
Function cublasChemv()

```c
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 \( A x \).
- **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 \( 1 + (n - 1) \times \text{abs(incx)} \).
- **incy** storage spacing between elements of \( x \); \( \text{incy} \) must not be zero.
CUDA

CUBLAS Library

Function cublasCher()

```
void

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

performs the Hermitian rank 1 operation

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

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

Input

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

\(n\) the number of rows and columns of matrix \(A\); \(n\) must be at least zero.
Input (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>single-precision scalar multiplier applied to ( x \times x^H ).</td>
</tr>
<tr>
<td>x</td>
<td>single-precision complex array of length at least ((l + (n - 1) \times \text{abs}(\text{incx}))).</td>
</tr>
<tr>
<td>incx</td>
<td>the storage spacing between elements of ( x ); ( \text{incx} ) must not be zero.</td>
</tr>
<tr>
<td>A</td>
<td>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.</td>
</tr>
<tr>
<td>lda</td>
<td>leading dimension of the two-dimensional array containing ( A ); ( \text{lda} ) must be at least ( \max(1, n) ).</td>
</tr>
</tbody>
</table>

Output

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

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

- `CUBLAS_STATUS_NOT_INITIALIZED` if CUBLAS library was not initialized
- `CUBLAS_STATUS_INVALID_VALUE` if \( n < 0 \) or \( \text{incx} == 0 \)
- `CUBLAS_STATUS_EXECUTION_FAILED` if function failed to launch on GPU

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

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

- **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 \( x \times y^H \) and whose conjugate is applied to \( y \times x^H \).

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

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

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

- **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.

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

**Output**

- **A** updated according to

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

**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

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

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 \( n < 0, \text{incx} == 0, \text{incy} == 0 \)
- CUBLAS_STATUS_EXECUTION_FAILED if function failed to launch on GPU

Function cublasChpr()

\[
\text{void}
\]

\[
\text{cublasChpr (char uplo, int n, float alpha,}
\]

\[
\text{const cuComplex *x, int incx, cuComplex *AP)}
\]

performs the Hermitian rank 1 operation

\[ A \times A^H \]

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

- \( \text{uplo} \) specifies whether the matrix data is stored in the upper or the lower triangular part of array \( A \). If \( \text{uplo} == 'U' \) or \( 'u' \), the upper triangular part of \( A \) is supplied in \( A \). If \( \text{uplo} == 'L' \) or \( 'l' \), 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.
- \( \text{alpha} \) is a single-precision scalar multiplier applied to \( x \).
- \( x \) is a single-precision complex array of length at least \( 1 + (n - 1) \times \text{abs}(\text{incx}) \).
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()**

```c
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

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

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 that is supplied in packed form.

**Input**

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

- **n**: The number of rows and columns of matrix A; `n` must be at least zero.

- **alpha**: Single-precision complex scalar multiplier applied to `x` and whose conjugate is applied to `y`.

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

- **incx**: The storage spacing between elements of `x`; `incx` must not be zero.

- **y**: Single-precision complex array of length at least 
  \[(1 + (n-1) \times \text{abs}(\text{incy}))\].

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

- **AP**: 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 <= 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 >= 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**: Updated according to 
  \[A = \alpha \times \text{x}^\dagger + \text{alpha} \times \text{y}^\dagger + \text{A}\]

**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 \(n < 0, \text{incx} == 0, \text{or incy} == 0\)
- **CUBLAS_STATUS_EXECUTION_FAILED**: If function failed to launch on GPU
Function cublasCtbmv()

```c
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

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

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

\( x \) 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 \( \text{uplo} = 'U' \text{ or 'u'}, A \) is an upper triangular band matrix. If \( \text{uplo} = 'L' \text{ or 'l'}, A \) is a lower triangular band 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 matrix \( A \) is unit triangular. 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.

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

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
\( (1 + (n-1) \times \text{abs}(\text{incx})) \).
\( x \) on entry, 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 \( x = \text{op}(\Lambda) \times x \).

Reference: http://www.netlib.org/blas/ctbmv.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 \( \text{incx} == 0 \), \( k < 0 \), or \( n < 0 \)
- \text{CUBLAS_STATUS_ALLOC_FAILED} if function cannot allocate enough internal scratch vector memory
- \text{CUBLAS_STATUS_EXECUTION_FAILED} if function failed to launch on GPU
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) * x = b,
where op(A) = A, op(A) = A^T, or op(A) = A^H;
```

where `b` and `x` are `n`-element vectors, and `A` is an `n×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. In the current implementation `n` must not exceed 2035.
- **k** specifies the number of superdiagonals or subdiagonals.
  - If `uplo` == 'U' or 'u', `k` specifies the number of superdiagonals.
  - If `uplo` == 'L' or 'l', `k` specifies the number of subdiagonals; `k` must be at least zero.
Input (continued)

\( A \) is 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.

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

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

Output

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

Reference: [http://www.netlib.org/blas/ctbsv.f](http://www.netlib.org/blas/ctbsv.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 \( n = 0, n < 0, \text{or} n > 2035 \)
- \text{CUBLAS_STATUS_EXECUTION_FAILED} if function failed to launch on GPU

Function \text{cublasCtpmv}()

```c
void
\text{cublasCtpmv} (char \text{uplo}, char \text{trans}, char \text{diag}, int \text{n},
   const cuComplex *\text{AP}, cuComplex *\text{x}, int \text{incx})
```

performs one of the matrix-vector operations

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

where \( \text{op}(A) = A, \text{op}(A) = A^T, \text{or} \ \text{op}(A) = A^H \).
**CHAPTER 3 Single-Precision BLAS2 Functions**

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**

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

- **x** 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.

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

**Output**

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

**Reference:** [http://www.netlib.org/blas/ctpmv.f](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 incx == 0 or n < 0
Function cublasCtpsv()

```c
void
cublasCtpsv (char uplo, char trans, char diag, int n,
    const cuComplex *AP, cuComplex *X, int incx)
```

solves one of the systems of equations

```latex
\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. In the current implementation \( n \) must not exceed 2035.
CHAPTER 3 Single-Precision BLAS2 Functions

Input (continued)

**AP** single-precision complex array with at least \((n \times (n+1))/2\) elements.
If \(\text{uplo} = 'U'\) 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 \(\text{AP}[i+(j \times (j+1)/2)]\). If \(\text{uplo} = 'L'\) 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
\(\text{AP}[i+((2 \times n-j+1) \times j)/2]\). When \(\text{diag} = 'U'\) or 'u', the
diagonal elements of \(A\) are not referenced and are assumed to be unity.

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

**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/ctpsv.f](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, n < 0, or n > 2035
- `CUBLAS_STATUS_EXECUTION_FAILED` if function failed to launch on GPU

**Function cublasCtrmv()**

```c
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

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

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 matrix consisting of single-
precision complex elements.
**CUDA CUBLAS Library**

**Input**

- **uplo** specifies whether the matrix $A$ is an upper or lower triangular matrix.
  - If $uplo == 'U'$ or 'u', $A$ is an upper triangular matrix.
  - If $uplo == 'L'$ or 'l', $A$ is an lower triangular matrix.

- **trans** specifies $\text{op}(A)$. 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 $A$ 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 complex 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 $A$; lda must be at least $\max(1, n)$.

- **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.

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

**Output**

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

**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()`.

**Error Status**

- `CUBLAS_STATUS_NOT_INITIALIZED` if CUBLAS library was not initialized
- `CUBLAS_STATUS_INVALID_VALUE` if incx == 0 or n < 0
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 \( lda \) 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

\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.

\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 \( 'c' \), \( \text{op}(A) = A^H \).

\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.

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

\[ \text{CUBLAS_STATUS_ALLOC_FAILED} \]
if function cannot allocate enough internal scratch vector memory

\[ \text{CUBLAS_STATUS_EXECUTION_FAILED} \]
if function failed to launch on GPU
CUDA

CUBLAS Library

Input (continued)

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

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

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

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

Output

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

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

Error status for this function can be retrieved via 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
Double-Precision BLAS2 Functions

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

- “Double-Precision BLAS2 Functions” on page 122
- “Double-Precision Complex BLAS2 functions” on page 146
Double-Precision BLAS2 Functions

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

The double-precision BLAS2 functions are as follows:

- “Function cublasDgbmv()” on page 123
- “Function cublasDgemv()” on page 124
- “Function cublasDger()” on page 126
- “Function cublasDsbmv()” on page 127
- “Function cublasDspmv()” on page 129
- “Function cublasDspr()” on page 130
- “Function cublasDspr2()” on page 131
- “Function cublasDsymv()” on page 132
- “Function cublasDsyr()” on page 134
- “Function cublasDsyr2()” on page 135
- “Function cublasDtbmv()” on page 136
- “Function cublasDtbsv()” on page 138
- “Function cublasDtpmv()” on page 140
- “Function cublasDtpsv()” on page 141
- “Function cublasDtrmv()” on page 142
- “Function cublasDtrsv()” on page 144
Function cublasDgbmv()

```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

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

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 \( kl \) 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** 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 scalar multiplier applied to \( \text{op}(A) \).

- **A** double-precision 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 \( ku \times ku \) triangle) are not referenced.

- **lda** leading dimension \( A \); \( lda \) must be at least \((\text{kl}+\text{ku}+1)\).

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

- **incx** specifies the increment for the elements of \( x \); \( incx \) must not be zero.
Function cublasDgemv()

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

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

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 \( m \times n \) 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**

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

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

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 \( x \times y^T \).
- \( x \) 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 \((lda,n)\).
- \( \text{lda} \) leading dimension of two-dimensional array used to store matrix \( A \).

**Output**

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

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 \( m < 0, n < 0, \text{incx} = 0, \text{or incy} = 0 \)
Chapters 4 Double-Precision BLAS2 Functions

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 \(\text{uplo} = \text{'U'}\) or \(\text{'u'}\), the upper triangular part is being supplied. If \(\text{uplo} = \text{'L'}\) or \(\text{'l'}\), 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)

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

\(A\) double-precision array of dimensions \((\text{lda}, n)\). When \(\text{uplo} = \text{'U'}\) 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 \(\text{uplo} = \text{'L'}\) 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 \(k+1\).

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

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

\(\text{incy}\) 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/dsbmv.f](http://www.netlib.org/blas/dsbmv.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 \(k < 0, n < 0, \text{incx} = 0\), 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 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 `uplo == 'u'`, the upper triangular part of \( A \) is supplied in `AP`. If `uplo == 'L'` or `uplo == '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 \( A x \).
- `AP` is a double-precision array with at least \( (n \times (n + 1)) / 2 \) elements. If `uplo == 'U'` or `uplo == '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 `uplo == '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 \( AP[i+(2 \times n-j+1) \times j]/2 \).
- `x` is a double-precision array of length at least \( (1+(n-1) \times \text{abs}(\text{incx})) \).
- `incx` is the storage spacing between elements of `x`; `incx` must not be zero.
- `beta` is a double-precision scalar multiplier applied to vector `y`.
- `y` is a double-precision array of length at least \( (1+(n-1) \times \text{abs}(\text{incy})) \). If `beta` is zero, `y` is not read.
- `incy` is the storage spacing between elements of `y`; `incy` must not be zero.

**Output**

- `y` is 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

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

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 \( A \). 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 \( AR \).
- **n** the number of rows and columns of matrix \( A \); \( n \) must be at least zero.
- **alpha** double-precision scalar multiplier applied to \( x^T \).
- **x** double-precision array of length at least \( (1 + (n-1) \times \text{abs} \text{(incx)}) \).
- **incx** storage spacing between elements of \( x \); incx must not be zero.
- **AP** double-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 \( i <= j \) \( A[i,j] \) is stored in \( AP[i+(j*(j+1)/2)] \). If \( uplo == \'L' \) or \'l' \), the array \( AP \) contains the lower triangular part of the symmetric matrix \( A \), packed sequentially, column by column; that is, if \( i >= j \) \( A[i,j] \) is stored in \( AP[i+((2 \times n-j+1) \times j)/2] \).
Function cublasDspr2()

```c
void
cublasDspr2 (char uplo, int n, double alpha,
            const double *x, int incx, const double *y,
            int incy, double *AP)
```

performs the symmetric rank 2 operation

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

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

- **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 \( x y^T \) and \( y x^T \).
- **x** double-precision array of length at least \((1+(n-1)*\text{abs}(\text{incx})).\)
- **incy** storage spacing between elements of \( x \); \( \text{incy} \) must not be zero.
- **y** double-precision array of length at least \((1+(n-1)*\text{abs}(\text{incy})).\)
Input (continued)

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

\[\text{AP}\] double-precision 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
symmetric 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
symmetric matrix \(A\), packed sequentially, column by column; that is, if
\(i \geq j, A[i,j]\) is stored in \(\text{AP}[i +((2 \times n-j+1) \times j)/2]\).

Output

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

Reference: http://www.netlib.org/blas/dspr2.f

Error status for this function can be retrieved via \text{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_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 \text{cublasDsymv}()

\[
\text{void} \quad \text{cublasDsymv} (\text{char} \text{ uplo}, \text{ int} \text{ n}, \text{ double} \text{ alpha,} \\
\quad \text{ const double} \text{ *A, int lda, const double} \text{ *x,} \\
\quad \text{ int incx, double beta, double} \text{ *y, int incy})
\]

performs the matrix-vector operation
\[
y = \alpha \times A \times x + \beta \times y,
\]

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 \( \text{uplo} = 'U' \text{ or } 'u' \), 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 \( \text{uplo} = 'L' \text{ or } 'l' \), 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 \( A \ast x \).

- **A** double-precision 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 part of the symmetric 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 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** double-precision array of length at least \((1+(n-1) \ast \text{abs(incx)})\).

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

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

- **Y** single-precision array of length at least \((1+(n-1) \ast \text{abs(incy)})\). 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 \( y = \text{alpha} \ast A \ast x + \text{beta} \ast y \).

**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 \( n < 0, \text{incx} == 0, \text{or incy} == 0 \)
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 \( \text{uplo} == \text{'U'} \) or \( \text{'u'} \), only the upper triangular part of \( A \) is referenced. If \( \text{uplo} == \text{'L'} \) or \( \text{'l'} \), only the lower triangular part of \( A \) is referenced.
- **n** the number of rows and columns of matrix \( A \); \( n \) must be at least zero.
- **alpha** double-precision scalar multiplier applied to \( x \times x^T \).
- **x** double-precision array of length at least \( (1+(n-1) \times \text{abs(incx)}) \).
- **incx** the storage spacing between elements of \( x \); \( \text{incx} \) must not be zero.
- **A** double-precision array of dimensions \( (\text{lda}, n) \). If \( \text{uplo} == \text{'U'} \) or \( \text{'u'} \), \( A \) contains the upper triangular part of the symmetric matrix, and the strictly lower triangular part is not referenced. If \( \text{uplo} == \text{'L'} \) or \( \text{'l'} \), \( A \) 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 \( \text{max(1, n)} \).
CHAPTER 4 Double-Precision BLAS2 Functions

Output

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

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

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

Error Status

- **CUBLAS_STATUS_NOT_INITIALIZED** if CUBLAS library was not initialized
- **CUBLAS_STATUS_INVALID_VALUE** if $n < 0$ or incx == 0
- **CUBLAS_STATUS_ARCH_MISMATCH** if function invoked on device that does not support double precision
- **CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU

Function cublasDsyr2()

```c
void
cublasDsyr2 (char uplo, int n, double alpha,
              const double *x, int incx, const double *y,
              int incy, double *A, int lda)
```

performs the symmetric rank 2 operation

$$A = \alpha x \times y^T + \alpha y \times x^T + A,$$

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

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 $\text{'u'}$, only the upper triangular part of $A$ is referenced and the lower triangular part of $A$ is inferred. If $\text{uplo} = \text{'L'}$ or $\text{'l'}$, only the lower triangular part of $A$ is referenced and the upper triangular part of $A$ is inferred.
- **n** the number of rows and columns of matrix $A$; $n$ must be at least zero.
- **alpha** double-precision scalar multiplier applied to $x \times y^T$ and $y \times x^T$.
- **x** double-precision array of length at least $(1 + (n-1) \times \text{abs(incx)))$.
- **incy** storage spacing between elements of $x$; incy must not be zero.
- **y** double-precision array of length at least $(1 + (n-1) \times \text{abs(incy)))$. 


Function cublasDtbmv()

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

performs one of the matrix-vector operations

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

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 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 $\text{uplo} == 'U' \text{ or } 'u'$, $A$ is an upper triangular band matrix. If $\text{uplo} == 'L' \text{ or } 'l'$, $A$ is a lower triangular band matrix.
- **trans** specifies $\text{op}(A)$. If $\text{trans} == 'N' \text{ or } 'n'$, $\text{op}(A) = A$.
- If $\text{trans} == 'T' \text{, 't' , 'C' , or } 'c'$, $\text{op}(A) = A^T$.
- **diag** specifies whether or not matrix $A$ is unit triangular. 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.
- **k** specifies the number of superdiagonals or subdiagonals. If $\text{uplo} == 'U' \text{ or } 'u'$, $k$ specifies the number of superdiagonals. If $\text{uplo} == 'L' \text{ or } 'l'$, $k$ specifies the number of subdiagonals; $k$ must at least be zero.
- **A** double-precision array of dimension $(\text{lda}, n)$. If $\text{uplo} == 'U' \text{ or } 'u'$, 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 $\text{uplo} == 'L' \text{ or } 'l'$, 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.
- **lda** is the leading dimension of $A$; $\text{lda}$ must be at least $k+1$.
- **x** double-precision 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 $x$; $\text{incx}$ must not be zero.

Output

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

Reference: [http://www.netlib.org/blas/dtbmv.f](http://www.netlib.org/blas/dtbmv.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, k &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_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 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

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

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**

- **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. In the current implementation \( n \) must not exceed 2035.
CHAPTER 4 Double-Precision BLAS2 Functions

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, n &lt; 0, or n &gt; 2035</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>

Reference: [http://www.netlib.org/blas/dtbsv.f](http://www.netlib.org/blas/dtbsv.f)
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 \( \text{op}(A) = A^T \),

\( x \) 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 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.

- **AP** double-precision array with at least \( (n + (n + 1))/2 \) elements. If \( \text{uplo} == 'U' \) 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 \( \text{uplo} == 'L' \) 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 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** 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>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

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

\[b\] 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

- **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', '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.
Function cublasDtrmv()

```c
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

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

where \( \text{op}(A) = A \) or \( \text{op}(A) = A^T \),
**CHAPTER 4 Double-Precision BLAS2 Functions**

\[ x \text{ is an } n\text{-element double-precision vector, and } A \text{ is an } n \times n, \text{ unit or non-unit, upper or lower, triangular matrix consisting of double-precision elements.} \]

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

- **x**
  - A 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**

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

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

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

- **CUBLAS_STATUS_NOT_INITIALIZED** if CUBLAS library was not initialized
- **CUBLAS_STATUS_INVALID_VALUE** if \( \text{incx} = 0 \) or \( n < 0 \)
Function cublasDtrsv()

```c
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 \( \text{lda} \) 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 \( \text{uplo} = 'U' \) 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.

**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 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.
CHAPTER 4

## Double-Precision BLAS2 Functions

### 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

### 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) * abs(incx))`. On entry, x contains the n-element, right-hand-side vector b. On exit, it is overwritten with the solution vector x.

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

### Output

- **x**: updated to contain the solution vector x that solves `op(A) * x = b`.
Double-Precision Complex BLAS2 functions

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

Two double-precision complex BLAS2 functions are implemented:

- “Function cublasZgbmv()” on page 147
- “Function cublasZgemv()” on page 149
- “Function cublasZgerc()” on page 150
- “Function cublasZgeru()” on page 151
- “Function cublasZhbmv()” on page 153
- “Function cublasZhemv()” on page 155
- “Function cublasZher()” on page 156
- “Function cublasZher2()” on page 158
- “Function cublasZhpmv()” on page 159
- “Function cublasZhpr()” on page 161
- “Function cublasZhpr2()” on page 162
- “Function cublasZtbmv()” on page 163
- “Function cublasZtbsv()” on page 165
- “Function cublasZtpmv()” on page 167
- “Function cublasZtpsv()” on page 168
- “Function cublasZtrmv()” on page 170
- “Function cublasZtrsv()” on page 171
CHAPTER 4 Double-Precision BLAS2 Functions

Function cublasZgbmv()

```
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

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

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

**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.

- \(kl\) 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 \((lda, n)\). 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 
  \((1 + (m-1) \cdot \text{abs}(\text{incx}))\).
- **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 \(\text{beta}\) 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 \(\text{beta}\) is zero, \(y\) is not read.
- **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](http://www.netlib.org/blas/zgbmv.f)

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

**Error Status**

- `CUBLAS_STATUS_NOT_INITIALIZED` if CUBLAS library was not initialized
- `CUBLAS_STATUS_INVALID_VALUE` if \(m < 0, n < 0, \text{incx} == 0,\) 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 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

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

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 \( \mathbf{x} \) and \( \mathbf{y} \) 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 \( \text{lda} \) 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 \( \mathbf{x} \); incx must not be zero.

- **beta** double-precision complex scalar multiplier applied to vector \( \mathbf{y} \). If beta is zero, \( \mathbf{y} \) is not read.
Function `cublasZgerc()`

```c
void
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

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

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>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 $m &lt; 0$, $n &lt; 0$, $incx = 0$, 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 cublasZgeru()

```c
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

$$A = \alpha \cdot x \cdot y^\text{H} + A,$$
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 \( A \).

**Input**

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

**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 \( m < 0, n < 0, \text{incx} = 0, \text{incy} = 0 \)
- `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()

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

performs the matrix-vector operation

\[ y = \alpha A \ast x + \beta y, \]

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 \( A \ast x \).
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.

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

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

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

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

**y**  
double-precision complex array of length at least \((1 + (n - 1) \times \text{abs(incy)})\). If \(\text{beta} = 0\), \(y\) is not read. \(\text{incy}\) 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 \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 (k &lt; 0, n &lt; 0, \text{incx} == 0, ) or (\text{incy} == 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 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 \( \text{uplo} = 'U' \) 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** double-precision complex scalar multiplier applied to \( A x \).

- **A** double-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** double-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.
CUDA

Input (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta</td>
<td>double-precision complex scalar multiplier applied to vector y.</td>
</tr>
<tr>
<td>y</td>
<td>double-precision complex array of length at least $(1 + (n - 1) \cdot \text{abs}(\text{incy}))$. If beta is zero, y is not read.</td>
</tr>
<tr>
<td>incy</td>
<td>storage spacing between elements of y; incy must not be zero.</td>
</tr>
</tbody>
</table>

Output

| y          | updated according to $y = \alpha \cdot A \cdot x + \beta \cdot y$. |

Reference: [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

<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 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

$$A = \alpha \cdot x \cdot x^H + A,$$

where alpha is a double-precision scalar, x is an n-element double-precision complex vector, and A is an n×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.
## 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 <code>A</code>. If <code>uplo == 'U'</code> or <code>'u'</code>, only the upper triangular part of <code>A</code> is referenced. If <code>uplo == 'L'</code> or <code>'l'</code>, only the lower triangular part of <code>A</code> is referenced.</td>
</tr>
<tr>
<td><code>n</code></td>
<td>The number of rows and columns of matrix <code>A</code>; <code>n</code> must be at least zero.</td>
</tr>
<tr>
<td><code>alpha</code></td>
<td>Double-precision scalar multiplier applied to <code>x * x'</code>.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>Double-precision complex array of length at least <code>(1 + (n-1) * abs(incx))</code>.</td>
</tr>
<tr>
<td><code>incx</code></td>
<td>The storage spacing between elements of <code>x</code>; <code>incx</code> must not be zero.</td>
</tr>
<tr>
<td><code>A</code></td>
<td>Double-precision complex array of dimensions <code>(lda, n)</code>. If <code>uplo == 'U'</code> or <code>'u'</code>, <code>A</code> contains the upper triangular part of the Hermitian matrix, and the strictly lower triangular part is not referenced. If <code>uplo == 'L'</code> or <code>'l'</code>, <code>A</code> 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 the two-dimensional array containing <code>A</code>; <code>lda</code> must be at least <code>max(1, n)</code>.</td>
</tr>
</tbody>
</table>

## Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>A</code></td>
<td>Updated according to <code>A = alpha * x * x' + A</code>.</td>
</tr>
</tbody>
</table>

## Reference

[http://www.netlib.org/blas/zher.f](http://www.netlib.org/blas/zher.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_INVALID_VALUE</code></td>
<td>If <code>n &lt; 0</code> or <code>incx == 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_EXECUTION_FAILED</code></td>
<td>If function failed to launch on GPU</td>
</tr>
</tbody>
</table>
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

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

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.

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


CHAPTER 4 Double-Precision BLAS2 Functions

Output

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

Reference: http://www.netlib.org/blas/zher2.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 \( n < 0, \text{incx} == 0, \text{incy} == 0 \)
- **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 \texttt{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

- \texttt{uplo} specifies whether the matrix data is stored in the upper or the lower triangular part of array \( AP \). If \texttt{uplo} == 'U' or 'u', the upper triangular part of \( A \) is supplied in \( AP \). If \texttt{uplo} == 'L' or 'l', the lower triangular part of \( A \) is supplied in \( AP \).
- \texttt{n} the number of rows and columns of matrix \( A; n \) must be at least zero.
- \texttt{alpha} double-precision complex scalar multiplier applied to \( A^* x \).
Input (continued)

AP  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.

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

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 $(1 + (n-1) \times \text{abs}(\text{incy}))$.
    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 $y = \alpha \times A \times x + \beta \times y$.

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

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

Error Status

- **CUBLAS_STATUS_NOT_INITIALIZED** if CUBLAS library was not initialized
- **CUBLAS_STATUS_INVALID_VALUE** if $n < 0$, incx == 0, or incy == 0
- **CUBLAS_STATUS_ARCH_MISMATCH** if function invoked on device that
  does not support double precision
- **CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU
Function cublasZhpr()

```c
void

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

performs the Hermitian rank 1 operation

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

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

**Input**

- **uplo** specifies whether the matrix data is stored in the upper or the lower triangular part of array \( A \). If \( uplo == 'U' \) or \( 'u' \), the upper triangular part of \( A \) is supplied in \( A \). If \( uplo == 'L' \) or \( 'l' \), 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 double-precision scalar multiplier applied to \( x^* x^H \).
- **x** is a double-precision complex array of length at least \((1+(n-1)*\text{abs}(incx))\).
- **incx** is the storage spacing between elements of \( x \); \( incx \) must not be zero.
- **AP** is a double-precision complex array with at least \((n*(n+1))/2\) elements. If \( uplo == 'U' \) or \( 'u' \), array \( AP \) contains the upper triangular part of the Hermitian matrix \( A \), packed sequentially, column by column; that is, if \( i \leq j \), \( A[i,j] \) is stored in \( AP[i+(j*(j+1)/2)] \). If \( uplo == 'L' \) or \( 'l' \), the array \( AP \) contains the lower triangular part of the Hermitian matrix \( A \), packed sequentially, column by column; that is, if \( i \geq j \), \( A[i,j] \) is stored in \( AP[i+((2*n-j-1)*j)/2] \). 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 \( A = \alpha x^* x^H + A \).

**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**

- **CUBLAS_STATUS_NOT_INITIALIZED** if CUBLAS library was not initialized
- **CUBLAS_STATUS_INVALID_VALUE** if \( n < 0 \) or \( \text{incx} == 0 \)
- **CUBLAS_STATUS_ARCH_MISMATCH** if function invoked on device that does not support double precision
- **CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU

**Function cublasZhpr2()**

```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

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

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 that is supplied in packed form.

**Input**

- **uplo** specifies whether the matrix data is stored in the upper or the lower triangular part of array \( A \). If \( \text{uplo} == \text{'U'} \) or \( \text{'u'} \), only the upper triangular part of \( A \) may be referenced and the lower triangular part of \( A \) is inferred. If \( \text{uplo} == \text{'L'} \) or \( \text{'l'} \), only the lower triangular part of \( A \) may be referenced and the upper triangular part of \( A \) is inferred.
- **n** the number of rows and columns of matrix \( A; n \) must be at least zero.
- **alpha** double-precision complex scalar multiplier applied to \( x^* y^H \) and whose conjugate is applied to \( y^* x^H \).
- **x** double-precision complex array of length at least \( (1 + (n-1) \cdot \text{abs(incx)}) \).
- **incy** the storage spacing between elements of \( x; \text{incy} \) must not be zero.
- **y** double-precision complex array of length at least \( (1 + (n-1) \cdot \text{abs(incy)}) \).
CHAPTER 4  Double-Precision BLAS2 Functions

Input (continued)

<table>
<thead>
<tr>
<th>Variable</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>AP</td>
<td>double-precision complex array with at least ((n * (n + 1)) / 2) elements. If uplo == 'U' or 'u', array AP contains the upper triangular part of the Hermitian matrix (A), packed sequentially, column by column; that is, if (i \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 Hermitian 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]). 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

\[ A \text{ updated according to } A = \alpha x * y^H + \alpha y * x^H + A \]

Reference: [http://www.netlib.org/blas/zhpr2.f](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

\[ x = op(A) * x, \]

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

**Input**

uplo  specifies whether the matrix A is an upper or lower triangular band matrix. If uplo == 'U' or 'u', A is an upper triangular band matrix. If uplo == 'L' or 'l', A is a lower triangular band matrix.

trans  specifies op(A). If trans == 'N' or 'n', op(A) = A.
If trans == 'T' or 't', op(A) = A^T.
If trans == 'C', or 'c', op(A) = A^H.

diag  specifies whether or not matrix A is unit triangular. If 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 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 complex array of length at least (1+(n-1) * abs(incx)).
On entry, x contains the source vector. On exit, x is overwritten with the result vector.

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

**Output**

x  updated according to x = op(A) * x.
Reference: http://www.netlib.org/blas/ztbmv.f

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

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, k &lt; 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 cublasZtbsv()

```c
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

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

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 \( \text{uplo} == 'U' \) or \( 'u' \), \( A \) is an upper triangular band matrix. If \( \text{uplo} == 'L' \) or \( 'l' \), \( A \) is a lower triangular band 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 \( 'c' \), \( \text{op}(A) = A^H \).
Input (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>n</td>
<td>The number of rows and columns of matrix A; n must be at least zero. In the current implementation n must not exceed 1016.</td>
</tr>
<tr>
<td>k</td>
<td>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.</td>
</tr>
<tr>
<td>A</td>
<td>Double-precision complex array of dimension (lda,n). If uplo == 'U' or 'u', the leading (k+1)×n part of the array A must contain the upper triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row k+1 of the array, the first superdiagonal starting at position 2 in row k, and so on. The top left k×k triangle of the array A is not referenced. If uplo == 'L' or 'l', the leading (k+1)×n part of the array A must contain the lower triangular band matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k×k triangle of the array is not referenced.</td>
</tr>
<tr>
<td>x</td>
<td>Double-precision complex array of length at least (1+(n-1) abs(incx)).</td>
</tr>
<tr>
<td>incx</td>
<td>Storage spacing between elements of x; incx must not be zero.</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Updated to contain the solution vector x that solves ( \text{op}(A) \ast x = b ).</td>
</tr>
</tbody>
</table>

Reference: [http://www.netlib.org/blas/ztbsv.f](http://www.netlib.org/blas/ztbsv.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>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, n &lt; 0, or n &gt; 1016</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 cublasZtpmv()

```c
void
cublasZtpmv (char uplo, char trans, char diag, int n, 
const cuDoubleComplex *AP, 
cuDoublesComplex *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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 \( \text{trans} = 'T' \) or \( 't' \), \( \text{op}(A) = A^T \).  
If \( \text{trans} = 'C' \), or \( 'c' \), \( \text{op}(A) = A^H \). |
| 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. |
| AP        | double-precision complex array with at least \( n \times (n+1)/2 \) elements.  
If \( \text{uplo} = 'U' \) 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 \( \text{uplo} = 'L' \) 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  
\( (1 + (n-1) \times \text{abs}(\text{incx})) \). On entry, \( x \) contains the source vector.  
On exit, \( x \) is overwritten with the result vector. |
| incx      | specifies the storage spacing for elements of \( x \); \( \text{incx} \) must not be zero. |
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

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

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

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

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

Input

\( \text{uplo} \) 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.

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

Input (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diag</td>
<td>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.</td>
</tr>
<tr>
<td>$n$</td>
<td>Specifies the number of rows and columns of the matrix $A$; $n$ must be at least zero. In the current implementation $n$ must not exceed 1016.</td>
</tr>
<tr>
<td>$\text{AP}$</td>
<td>Double-precision complex array with at least $(n * (n + 1) / 2)$ elements. If $\text{uplo} == 'U'$ or '$u'$, array $\text{AP}$ contains the upper triangular matrix $A$, packed sequentially, column by column; that is, if $i &lt;= j, A[i, j]$ is stored in $\text{AP}[i + (j * (j + 1) / 2)]$. If $\text{uplo} == 'L'$ or '$l'$, array $\text{AP}$ contains the lower triangular matrix $A$, packed sequentially, column by column; that is, if $i &gt;= j, A[i, j]$ is stored in $\text{AP}[i + (2 * n - j + 1) * j / 2]$. When $\text{diag} == 'U'$ or '$u'$, the diagonal elements of $A$ are not referenced and are assumed to be unity.</td>
</tr>
<tr>
<td>$\text{x}$</td>
<td>Double-precision complex array of length at least $1 + (n - 1) * \text{abs(incx)}$.</td>
</tr>
<tr>
<td>incx</td>
<td>Storage spacing between elements of $\text{x}$; incx must not be zero.</td>
</tr>
</tbody>
</table>

Output

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

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

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

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 $== 0$, $n &lt; 0$, or $n &gt; 1016$</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 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 \( \text{op}(A) = A, \text{op}(A) = A^T, \text{or} \ \text{op}(A) = A^H; \)

\( x \) 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 \( \text{uplo} == 'U' \) 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 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** double-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 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 \( \max(1, n) \).
CHAPTER 4 Double-Precision BLAS2 Functions

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

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

Function `cublasZtrsv()`

```c
void
cublasZtrsv(char uplo, char trans, char diag, int n,
const cuDoubleComplex *A, int lda,
cuDoubleComplex *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 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 \( lda \) 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.
CUDA CUBLAS Library

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

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

Error Status

- **CUBLAS_STATUS_NOT_INITIALIZED** if CUBLAS library was not initialized
- **CUBLAS_STATUS_INVALID_VALUE** if incx == 0 or n < 0
- **CUBLAS_STATUS_ARCH_MISMATCH** if function invoked on device that does not support double precision
- **CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU

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' or 't', \( \text{op}(A) = A^T \).
  - If trans == '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.
- **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 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 \((1 + (n - 1) \times \text{abs(incx)})\). 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 \( \text{op}(A) \times x = b \).
CHAPTER 5

BLAS3 Functions

Level 3 Basic Linear Algebra Subprograms (BLAS3) perform matrix-matrix operations. The CUBLAS implementations are described in the following sections:

- “Single-Precision BLAS3 Functions” on page 174
- “Single-Precision Complex BLAS3 Functions” on page 187
- “Double-Precision BLAS3 Functions” on page 206
- “Double-Precision Complex BLAS3 Functions” on page 219
Single-Precision BLAS3 Functions

The single-precision BLAS3 functions are listed below:

- “Function cublasSgemm()” on page 175
- “Function cublasSsymm()” on page 176
- “Function cublasSsyrk()” on page 178
- “Function cublasSsyr2k()” on page 180
- “Function cublasStrmm()” on page 182
- “Function cublasStrsm()” on page 184
Function cublasSgemm()

void

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:

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

where \( \text{op}(X) = X \) 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 \( \text{op}(C) \) an \( m \times n \) matrix. Matrices A, B, and C 
are stored in column-major format, and lda, ldb, and ldc are the 
leading dimensions of the two-dimensional arrays containing A, B, 
and C.

Input

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

- **transb** specifies \( \text{op}(B) \). If transb == 'N' or 'n', \( \text{op}(B) = B \).
  
  - If 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 \( \text{op}(A) \) * \( \text{op}(B) \).

- **A** single-precision array of dimensions (lda, k) if transa == 'N' or 
  'n', and of dimensions (lda,m) otherwise. If transa == 'N' or 
  'n', lda must be at least \( \max(1, m) \); otherwise, lda must be at least 
  \( \max(1, k) \).

- **lda** leading dimension of two-dimensional array used to store matrix A.
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

\[
C = \alpha A^T B + \beta C \quad \text{or} \quad C = \alpha B^T A + \beta C,
\]

where \( \alpha \) and \( \beta \) are single-precision scalars, \( A \) is a symmetric matrix consisting of single-precision elements and is stored in either
lower or upper storage mode. B and C are \( m \times n \) matrices consisting of single-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', \( C = \alpha A \ast B + \beta C \).
  
  - If \( \text{side} == 'R' \) or 'r', \( C = \alpha B \ast A + \beta C \).

- **\( \text{uplo} \)**: 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.

- **\( \alpha \)**: single-precision scalar multiplier applied to \( A \ast B \) or \( B \ast A \).

- **\( A \)**: single-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.
Input (continued)

<table>
<thead>
<tr>
<th>B</th>
<th>single-precision array of dimensions $(ldb, n)$. On entry, the leading $m \times n$ part of the array contains the matrix $B$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldb</td>
<td>leading dimension of $B$; $ldb$ must be at least $\max(1, m)$.</td>
</tr>
<tr>
<td>beta</td>
<td>single-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>single-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

| $C$ | updated according to \[ C = \alpha * A * B + \beta* C \text{ or } C = \alpha * B * A + \beta* C. \] |

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

\[ C = \alpha * A * A^T + \beta * C \text{ or } C = \alpha * A^T * A + \beta * C, \]

where $\alpha$ and $\beta$ are single-precision scalars. $C$ is an $n \times n$ symmetric matrix consisting of single-precision elements and is stored in either lower or upper storage mode. $A$ is a matrix consisting of single-precision elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.
Input

uplo 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 * A^T + beta * C. If 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 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 specifies the number of columns of matrix A. If trans == 'N' or 'n', k specifies the number of columns 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 (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading n x k part of array A contains the matrix A; otherwise, the leading k x n part of the array contains the matrix A.

lda leading dimension of A. When trans == 'N' or 'n', lda must be at least max(1, n). Otherwise lda must be at least max(1, k).

beta single-precision scalar multiplier applied to C. If beta is zero, C is not read.

C = alpha * A * A^T + beta * C.

If trans == 'T', 't', 'C', or 'c',
C = alpha * A^T * A + beta * C.

C = alpha * A * A^T + beta * C.

C = alpha * A^T * A + beta * C.
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

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 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 cublasSsyr2k()

```c
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

\[
C = alpha * A * B^T + alpha * B * A^T + beta * C \quad \text{or} \\
C = alpha * A^T * B + alpha * B^T * A + beta * C,
\]

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 \( \text{uplo} = 'U' \) 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 \( 'n' \), \( C = \alpha A B^T + \alpha A^T B + \beta C \). If \( \text{trans} = 'T' \), \( 't' \), \( 'C' \), 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 \( \text{trans} = 'N' \) 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.

- **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 \( A \) 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 \( \text{trans} = 'N' \) or \( 'n' \), \( \text{lda} \) must be at least \( \max(1, n) \). Otherwise \( \text{lda} \) must be at least \( \max(1, k) \).

- **B** single-precision array of dimensions \((\text{ldb}, \text{kb})\), where \( \text{kb} = k \) when \( \text{trans} = 'N' \) or \( 'n' \), and \( k = n \) otherwise. When \( \text{trans} = 'N' \) or \( 'n' \), the leading \( n \times k \) part of array \( B \) must contain the matrix \( B \); otherwise the leading \( k \times n \) part of the array must contain the matrix \( B \).

- **ldb** leading dimension of \( B \). When \( \text{trans} = 'N' \) or \( 'n' \), \( \text{ldb} \) 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.
Input (continued)

\( C \) single-precision array of dimensions \((l_d c, n)\). 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.

\( l_d c \) leading dimension of \( C \); \( l_d c \) must be at least \( \max(1, n) \).

Output

\( C \) updated according to

\[
C = \alpha A B^T + \alpha B A^T + \beta C \quad \text{or} \\
C = \alpha A^T B + \alpha B^T A + \beta C.
\]

Reference:  http://www.netlib.org/blas/ssyr2k.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 \( n < 0 \) or \( k < 0 \)
- \texttt{CUBLAS_STATUS_EXECUTION_FAILED} if function failed to launch on GPU

Function \texttt{cublasStrmm()}

```c
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

\[
B = \alpha \text{op}(A) B \quad \text{or} \quad B = \alpha B \text{op}(A),
\]

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

- **side** specifies whether \( \text{op}(A) \) multiplies B from the left or right.
  - If side == 'L' or 'l', \( B = \alpha \cdot \text{op}(A) \cdot B \).
  - If 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 uplo == 'U' or 'u', A is an upper triangular matrix.
  - If uplo == 'L' or 'l', A is a lower triangular matrix.
- **transa** specifies the form of \( \text{op}(A) \) to be used in the 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** 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 scalar multiplier applied to \( \text{op}(A) \)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 array of dimensions (lda,k). If side == 'L' or 'l', k = m. If side == 'R' or 'r', k = n. If uplo == 'U' or 'u', the leading \( k \times k \) 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 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 diag == 'U' or 'u', the diagonal elements of A are not referenced and are assumed to be unity.
- **lda** leading dimension of A. When side == 'L' or 'l', it must be at least \( \max(m,k) \) and at least \( \max(m,n) \) 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

   \( \text{op}(A) \times X = \alpha \times B \) or \( X \times \text{op}(A) = \alpha \times B \),

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

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 \( \text{lda} \) 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 \( \text{op}(A) \) appears on the left or right of \( X \):
- \( \text{side} == \text{‘L’} \) or \( \text{‘l’} \) indicates solve \( \text{op}(A) \cdot X = \alpha \cdot B \);
- \( \text{side} == \text{‘R’} \) or \( \text{‘r’} \) indicates solve \( X \cdot \text{op}(A) = \alpha \cdot B \).

**uplo** specifies whether the matrix \( A \) is an upper or lower triangular matrix:
- \( \text{uplo} == \text{‘U’} \) or \( \text{‘u’} \) indicates \( A \) is an upper triangular matrix;
- \( \text{uplo} == \text{‘L’} \) or \( \text{‘l’} \) indicates \( A \) is a lower triangular matrix.

**transa** specifies the form of \( \text{op}(A) \) to be used in matrix multiplication.
- If \( \text{transa} == \text{‘N’} \) or \( \text{‘n’} \),
- If \( \text{transa} == \text{‘T’}, \text{‘t’}, \text{‘C’}, \text{or ‘c’} \),

**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.

**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 \((\text{lda}, k)\), where \( k \) is \( m \) when \( \text{side} == \text{‘L’} \) or \( \text{‘l’} \) and is \( n \) when \( \text{side} == \text{‘R’} \) or \( \text{‘r’} \). 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 matrix of \( A \) is not referenced. When \( \text{uplo} == \text{‘L’} \) 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. Note that when \( \text{diag} == \text{‘U’} \) or \( \text{‘u’} \), 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 \( \text{side} == \text{‘L’} \) or \( \text{‘l’} \), \( \text{lda} \) must be at least \( \max(1, m) \). When \( \text{side} == \text{‘R’} \) or \( \text{‘r’} \), \( \text{lda} \) must be at least \( \max(1, n) \).

**B** single-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 \( X \).

**ldb** leading dimension of the two-dimensional array containing \( B \); \( \text{ldb} \) must be at least \( \max(1, m) \).
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: \( \text{http://www.netlib.org/blas/strsm.f} \)

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 \( m < 0 \) or \( n < 0 \)
- \texttt{CUBLAS_STATUS_EXECUTION_FAILED} if function failed to launch on GPU
Single-Precision Complex BLAS3 Functions

These are the single-precision complex BLAS3 functions:

- “Function cublasCgemm()” on page 188
- “Function cublasChemm()” on page 189
- “Function cublasCherk()” on page 191
- “Function cublasCher2k()” on page 193
- “Function cublasCsymm()” on page 195
- “Function cublasCsyrk()” on page 197
- “Function cublasCsyr2k()” on page 199
- “Function cublasCtrmm()” on page 201
- “Function cublasCtrsm()” on page 203
Function cublasCgemm()

```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 \( \text{op}(X) = X^H; \)

and \( \alpha \) and \( \beta \) are single-precision complex scalars. \( A, B, \) and \( C \)
are matrices consisting of single-precision complex elements, with \( \text{op}(A) \) 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 \( \text{transa} == 'N' \) \( \text{or} \) \( 'n' \), \( \text{op}(A) = A \).
  - If \( \text{transa} == 'T' \) \( \text{or} \) \( 't' \), \( \text{op}(A) = A^T \).
  - If \( \text{transa} == 'C' \) \( \text{or} \) \( 'c' \), \( \text{op}(A) = A^H \).
- **transb** specifies \( \text{op}(B) \). If \( \text{transb} == 'N' \) \( \text{or} \) \( 'n' \), \( \text{op}(B) = B \).
  - If \( \text{transb} == 'T' \) \( \text{or} \) \( 't' \), \( \text{op}(B) = B^T \).
  - If \( \text{transb} == 'C' \) \( \text{or} \) \( 'c' \), \( \text{op}(B) = B^H \).
- **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 \( \text{transa} == 'N' \) \( \text{or} \) \( 'n' \), and of dimension \((\text{lda}, m)\) otherwise.
- **lda** leading dimension of \( A \). When \( \text{transa} == 'N' \) \( \text{or} \) \( 'n' \), 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 \( \text{transb} == 'N' \) \( \text{or} \) \( 'n' \), and of dimension \((\text{ldb}, k)\) otherwise.
- **ldb** leading dimension of \( B \). When \( \text{transb} == 'N' \) \( \text{or} \) \( 'n' \), it must be at least \( \max(1, k) \) and at least \( \max(1, n) \) otherwise.
CHAPTER 5

BLAS3 Functions

Input (continued)

**beta**

single-precision complex scalar multiplier applied to **C**. If **beta** is zero, **C** does not have to be a valid input.

**C**

single-precision array of dimensions (ldc, n).

**ldc**

leading dimension of **C**; 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/cgemm.f](http://www.netlib.org/blas/cgemm.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 \( m < 0, n < 0, \text{ or } k < 0 \)
- **CUBLAS_STATUS_EXECUTION_FAILED** if function failed to launch on GPU

Function `cublasChemm()`

```c
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

\[
C = \alpha \cdot A \cdot B + \beta \cdot C \quad \text{or} \quad C = \alpha \cdot B \cdot A + \beta \cdot C,
\]

where **alpha** and **beta** are single-precision complex scalars, **A** is a Hermitian matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. **B** and **C** are \( m \times n \) matrices consisting of single-precision complex elements.
Input

side specifies whether the Hermitian matrix \( A \) appears on the left-hand side or right-hand side of matrix \( B \).

If \( \text{side} == \text{'L'} \) or \( \text{'l'} \), \( C = \alpha A \ast B + \beta C \).

If \( \text{side} == \text{'R'} \) or \( \text{'r'} \), \( C = \alpha B \ast A + \beta C \).

uplo specifies whether the Hermitian matrix \( A \) 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.

\( m \) 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 \( \text{side} == \text{'L'} \) or \( \text{'l'} \); \( m \) 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 \( \text{side} == \text{'R'} \) or \( \text{'r'} \); \( n \) must be at least zero.

\( \alpha \) single-precision complex scalar multiplier applied to \( A \ast B \) or \( B \ast A \).

\( A \) single-precision complex array of dimensions (\( \text{lda} \), \( kA \)), where \( kA \) is \( m \) when \( \text{side} == \text{'L'} \) or \( \text{'l'} \) and is \( n \) otherwise. If \( \text{side} == \text{'L'} \) or \( \text{'l'} \), the leading \( m \times m \) part of array \( A \) must contain the Hermitian matrix such that when \( \text{uplo} == \text{'U'} \) or \( \text{'u'} \), 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 \( \text{uplo} == \text{'L'} \) or \( \text{'l'} \), 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 \( \text{side} == \text{'R'} \) or \( \text{'r'} \), the leading \( n \times n \) part of array \( A \) must contain the Hermitian matrix such that when \( \text{uplo} == \text{'U'} \) or \( \text{'u'} \), 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 \( \text{uplo} == \text{'L'} \) or \( \text{'l'} \), 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 \( \text{side} == \text{'L'} \) or \( \text{'l'} \), it must be at least \( \max(1, m) \) and at least \( \max(1, n) \) otherwise.
Input (continued)

B
single-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}
single-precision complex scalar multiplier applied to \(C\). If \(\beta\) is zero, \(C\) does not have to be a valid input.

\text{C}
single-precision complex 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 \(C = \alpha * A * B + \beta * C \) 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 \text{cublasGetError}().

Error Status

- \text{CUBLAS_STATUS_NOT_INITIALIZED}\) if CUBLAS library was not initialized
- \text{CUBLAS_STATUS_INVALID_VALUE}\) if \(m < 0\) or \(n < 0\)
- \text{CUBLAS_STATUS_EXECUTION_FAILED}\) if function failed to launch on GPU

Function \text{cublasCherk}()

\text{void}
\text{cublasCherk} (char \text{uplo}, char \text{trans}, int \text{n}, int \text{k},
float \text{alpha}, \text{const} cuComplex \*\text{A},
int \text{lda}, float \text{beta}, \text{cuComplex} \*\text{C},
int \text{ldc})

performs one of the Hermitian rank \(k\) operations

\[C = \alpha * A * A^H + \beta * C \text{ or } C = \alpha * A^H * A + \beta * C,\]

where \(\alpha\) and \(\beta\) are single-precision real scalars. \(C\) is an \(n \times n\) Hermitian matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. \(A\) is a matrix consisting of single-precision complex elements with dimensions of \(n \times k\) in the first case and \(k \times n\) in the second case.
Input

**uplo** specifies whether the Hermitian matrix C is stored in upper or lower storage mode. If uplo == 'U' or 'u', only the upper triangular part of the Hermitian matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If uplo == 'L' or 'l', only the lower triangular part of the Hermitian matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part.

**trans** specifies the operation to be performed. If trans == 'N' or 'n',

\[ C = \alpha A^H + \beta C \]

If trans == 'T', 't', 'C', or 'c',

\[ C = \alpha A^H A + \beta C \]

**n** specifies the number of rows and the number columns of matrix C. If trans == 'N' or 'n', n specifies the number of rows of matrix A. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrix A; n must be at least zero.

**k** If trans == 'N' or 'n', k specifies the number of columns of matrix A. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrix A; k must be at least zero.

**alpha** single-precision scalar multiplier applied to \( A + A^H \) or \( A^H A \).

**A** single-precision complex array of dimensions \((lda, ka)\), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading n×k part of array A contains the matrix A; otherwise, the leading k×n part of the array contains the matrix A.

**lda** leading dimension of A. When trans == 'N' or 'n', lda must be at least \( \max(1, n) \). Otherwise lda must be at least \( \max(1, k) \).

**beta** 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().

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

Function cublasCher2k()

void

performs one of the Hermitian rank $2k$ operations

$$C = \alpha A^H B + \alpha^* B A^H + \beta C \quad \text{or}$$

$$C = \alpha A^H B + \alpha^* B + \beta A + \beta C,$$
where alpha is a single-precision complex scalar and beta is a single-precision real scalar. C is an n×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×k in the first case and k×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 matrices A and B. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrices A and B; n must be at least zero.
- k: If trans == 'N' or 'n', k specifies the number of columns of matrices A and B. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrices A and B; k must be at least zero.
- alpha: single-precision complex scalar multiplier.
- A: single-precision complex array of dimensions (lda, ka), where ka is k when trans == 'N' or 'n' and is n otherwise. When trans == 'N' or 'n', the leading n×k part of array A contains the matrix A; otherwise, the leading k×n part of the array contains the matrix A.
- 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: single-precision complex 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).
CHAPTER 5 BLAS3 Functions

Reference: [http://www.netlib.org/blas/cher2k.f](http://www.netlib.org/blas/cher2k.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 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

\[
C = \text{alpha} \cdot A \cdot B + \text{beta} \cdot C \text{ or } C = \text{alpha} \cdot B \cdot A + \text{beta} \cdot C, 
\]
where \( \text{alpha} \) and \( \text{beta} \) are single-precision complex scalars, \( A \) is a symmetric matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. \( B \) and \( C \) are \( m \times n \) matrices consisting of single-precision complex elements.

**Input**

- **side** specifies whether the symmetric matrix \( A \) appears on the left-hand side or right-hand side of matrix \( B \).
  - If `side == 'L'` or `'l'`, \( C = \text{alpha} \times A \times B + \text{beta} \times C \).
  - If `side == 'R'` or `'r'`, \( C = \text{alpha} \times B \times A + \text{beta} \times C \).

- **uplo** specifies whether the symmetric matrix \( A \) is stored in upper or lower storage mode. If `uplo == 'U'` or `'u'`, only the upper triangular part of the symmetric matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If `uplo == 'L'` or `'l'`, only the lower triangular part of the symmetric matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part.

- **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 `side == 'L'` or `'l'`; \( m \) must be at least zero.

- **n** specifies the number of columns of matrix \( C \), and the number of columns of matrix \( B \). It also specifies the dimensions of symmetric matrix \( A \) when `side == 'R'` or `'r'`; \( n \) must be at least zero.

- **alpha** single-precision complex scalar multiplier applied to \( A \times B \) or \( B \times A \).

- **A** single-precision complex array of dimensions \((\text{lda}, \text{ka})\), where \( \text{ka} \) is \( m \) when `side == 'L'` or `'l'` and is \( n \) otherwise. If `side == 'L'` or `'l'`, the leading \( m \times m \) part of array \( A \) must contain the symmetric matrix such that when `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 `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 `side == 'R'` or `'r'`, the leading \( n \times n \) part of array \( A \) must contain the symmetric matrix such that when `uplo == 'U'` or `'u'`, the leading \( n \times n \) part stores the upper triangular part of the symmetric matrix, and the strictly lower triangular part of \( A \) is not referenced; and when `uplo == 'L'` or `'l'`, the leading \( n \times n \) part stores the lower triangular part of the symmetric matrix, and the strictly upper triangular part is not referenced.
CHAPTER 5

BLAS3 Functions

Input (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lda</td>
<td>leading dimension of A. When side == 'L' or 'l', it must be at least max(1, m) and at least max(1, n) otherwise.</td>
</tr>
<tr>
<td>B</td>
<td>single-precision complex 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>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 (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

C updated according to C = alpha * A * B + beta * C or C = alpha * B * A + beta * C.

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

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

Error Status

<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_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>

Function cublasCsyrk()

```c
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

\[ C = \alpha A A^T + \beta C \] or \[ C = \alpha A^T A + \beta C, \]

where \( \alpha \) and \( \beta \) are single-precision complex scalars. \( C \) is an \( n \times n \) symmetric matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. \( A \) is a matrix consisting of single-precision complex elements with dimensions of \( n \times k \) in the first case and \( k \times n \) in the second case.
Input

`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',

\[ C = \alpha A \ast A^T + \beta C. \]

If `trans` == 'T', 't', 'C', or 'c,'

\[ C = \alpha A^T \ast A + \beta C. \]

`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` specifies the number of columns of matrix `A`. If `trans` == 'N' or 'n', `k` specifies the number of columns of matrix `A`. If `trans` == 'T', 't', 'C', or 'c', `k` specifies the number of rows of matrix `A`; `k` must be at least zero.

`alpha` single-precision complex scalar multiplier applied to `A \ast A^T` or `A^T \ast A`.

`A` single-precision complex array of dimensions `(lda, ka)`, where `ka` is `k` when `trans` == 'N' or 'n' and is `n` otherwise. When `trans` == 'N' or 'n', the leading `n \times k` part of array `A` contains the matrix `A`; otherwise, the leading `k \times n` part of the array contains the matrix `A`.

`lda` leading dimension of `A`. When `trans` == 'N' or 'n', `lda` must be at least `\max(1, n)`. Otherwise `lda` must be at least `\max(1, k)`.

`beta` single-precision complex scalar multiplier applied to `C`. If `beta` is zero, `C` is not read.
CHAPTER 5  BLAS3 Functions

Input (continued)

\( C \) single-precision complex array of dimensions \((ldc, n)\). If \( uplo == 'U' \) or \( 'u' \), the leading \( n \times n \) triangular part of the array \( C \) must contain the upper triangular part of the symmetric matrix \( C \), and the strictly lower triangular part of \( C \) is not referenced. On exit, the upper triangular part of \( C \) is overwritten by the upper triangular part of the updated matrix. If \( uplo == 'L' \) or \( 'l' \), the leading \( n \times n \) triangular part of the array \( C \) must contain the lower triangular part of the symmetric matrix \( C \), and the strictly upper triangular part of \( C \) is not referenced. On exit, the lower triangular part of \( C \) is overwritten by the 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 \( C = alpha \times A \times A^T + beta \times C \) or
\( C = alpha \times A^T \times A + beta \times C \).

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

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

Error Status

\begin{itemize}
  \item \texttt{CUBLAS_STATUS_NOT_INITIALIZED} if CUBLAS library was not initialized
  \item \texttt{CUBLAS_STATUS_INVALID_VALUE} if \( n < 0 \) or \( k < 0 \)
  \item \texttt{CUBLAS_STATUS_EXECUTION_FAILED} if function failed to launch on GPU
\end{itemize}

Function \texttt{cublasCsyr2k()}

\begin{verbatim}
void
  cublasCsyr2k (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)
\end{verbatim}

performs one of the symmetric rank \( 2k \) operations
\begin{align*}
  C &= alpha \times A \times A^T + alpha \times B \times B^T + beta \times C \quad \text{or} \\
  C &= alpha \times A^T \times A + beta \times C,
\end{align*}

where \( alpha \) and \( beta \) are single-precision complex scalars. \( C \) is an \( n \times n \) symmetric matrix consisting of single-precision complex elements and is stored in either lower or upper storage mode. \( A \) and \( B \) are
matrices consisting of single-precision complex elements with dimensions of $n \times k$ in the first case and $k \times n$ in the second case.

Input

- **uplo** specifies whether the symmetric matrix $C$ is stored in upper or lower storage mode. If $\text{uplo} == \text{'U'}$ 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 $\text{'n'}$, $C = \text{alpha} \cdot A \cdot B^\top + \text{alpha} \cdot B \cdot A^\top + \beta \cdot C$. If $\text{trans} == \text{'T'}$, $\text{'t'}, \text{'C'}, \text{or 'c'}$, $C = \text{alpha} \cdot A^\top \cdot B + \text{alpha} \cdot B^\top \cdot A + \beta \cdot C$.

- **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.

- **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 $\max(1, n)$. 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 $\text{trans} == \text{'N'}$ or $\text{'n'}$ and is $n$ otherwise. When $\text{trans} == \text{'N'}$ or $\text{'n'}$, the leading $n \times k$ part of array $B$ contains the matrix $B$; otherwise, the leading $k \times n$ part of the array contains the matrix $B$.

- **ldb** leading dimension of $B$. When $\text{trans} == \text{'N'}$ 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 complex 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  

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  

\[ C = \alpha \cdot A \cdot B^T + \alpha \cdot B \cdot A^T + \beta \cdot C \]

or  

\[ C = \alpha \cdot A^T \cdot B + \alpha \cdot B^T \cdot A + \beta \cdot C. \]

Reference:  
http://www.netlib.org/blas/csy2k.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 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 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

\[ B = \alpha \cdot \text{op}(A) \cdot B \quad \text{or} \quad B = \alpha \cdot B \cdot \text{op}(A), \]

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

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 $\text{lda}$ and $\text{ldb}$ are the leading dimensions of the two-dimensional arrays that contain $A$ and $B$, respectively.

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{side}$</td>
<td>specifies whether $\text{op}(A)$ multiplies $B$ from the left or right. If $\text{side} = 'L' \text{ or } 'l'$, $B = \text{alpha} \times \text{op}(A) \times B$. If $\text{side} = 'R' \text{ or } 'r'$, $B = \text{alpha} \times B \times \text{op}(A)$.</td>
</tr>
<tr>
<td>$\text{uplo}$</td>
<td>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.</td>
</tr>
<tr>
<td>$\text{transa}$</td>
<td>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 $\text{transa} = 'C' \text{ or } 'c'$, $\text{op}(A) = A^H$.</td>
</tr>
<tr>
<td>$\text{diag}$</td>
<td>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.</td>
</tr>
<tr>
<td>$m$</td>
<td>the number of rows of matrix $B$; $m$ must be at least zero.</td>
</tr>
<tr>
<td>$n$</td>
<td>the number of columns of matrix $B$; $n$ must be at least zero.</td>
</tr>
<tr>
<td>$\text{alpha}$</td>
<td>single-precision complex 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 $B$.</td>
</tr>
<tr>
<td>$A$</td>
<td>single-precision complex array of dimensions $(\text{lda}, k)$. 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.</td>
</tr>
<tr>
<td>$\text{lda}$</td>
<td>leading dimension of $A$. When $\text{side} = 'L' \text{ or } 'l'$, it must be at least $\max(l, m)$ and at least $\max(l, n)$ otherwise.</td>
</tr>
</tbody>
</table>
CHAPTER 5  BLAS3 Functions

Input (continued)

<table>
<thead>
<tr>
<th>B</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldb</td>
<td>leading dimension of B; ldb must be at least max(1, m).</td>
</tr>
</tbody>
</table>

Output

| B     | updated according to B = alpha * op(A) * B or B = alpha * B * op(A).                                                                                                                                     |

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

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

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>

Function `cublasCtrsm()`

```c
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

\[
\text{op}(A) \times X = \alpha \times \text{B} \quad \text{or} \quad X \times \text{op}(A) = \alpha \times \text{B},
\]

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

\( \alpha \) is a single-precision complex scalar, and \( X \) and \( B \) are \( m \times n \)
matrices that consist of single-precision complex elements. \( A \) is a unit or non-unit, upper or lower, triangular matrix.

The result matrix \( X \) overwrites input matrix \( B \); that is, on exit the result is stored in \( B \). Matrices \( A \) and \( B \) are stored in column-major format, and
lda and ldb are the leading dimensions of the two-dimensional arrays that contain A and B, respectively.

Input

side specifies whether \( \text{op}(A) \) appears on the left or right of \( X \):

- side == 'L' or 'l' indicates solve \( \text{op}(A) \ast X = \alpha \ast B \);
- side == 'R' or 'r' indicates solve \( X \ast \text{op}(A) = \alpha \ast B \).

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 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 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 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

\[ B \] contains the solution matrix \( X \) satisfying \( \text{op}(A) \times X = \alpha \times B \) or \( X \times \text{op}(A) = \alpha \times B \).

Reference: http://www.netlib.org/blas/ctrsm.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>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>
Double-Precision BLAS3 Functions

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

The double-precision BLAS3 functions are listed below:

- "Function cublasDgemm()" on page 207
- "Function cublasDsymm()" on page 208
- "Function cublasDsyrk()" on page 210
- "Function cublasDsyrk2k()" on page 212
- "Function cublasDtrmm()" on page 214
- "Function cublasDtrsm()" on page 216
Function cublasDgemm()

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 lda, ldb, and ldc are the leading dimensions of the two-dimensional arrays containing A, B, and C.

Input

- \( \text{transa} \) specifies \( \text{op}(A) \). If \( \text{transa} = 'N' \) or \( 'n' \), \( \text{op}(A) = A \).
  - If \( \text{transa} = 'T' \), 't', 'C', or 'c', \( \text{op}(A) = A^T \).
- \( \text{transb} \) 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 double-precision scalar multiplier applied to \( \text{op}(A) \) * \( \text{op}(B) \).
- A double-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 \( 'n' \), lda must be at least \( \max(1, m) \); otherwise, 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)

\begin{itemize}
  \item \(B\) double-precision array of dimensions \((ldb, n)\) if \(\text{transb} = 'N'\) or 'n', and of dimensions \((ldb, k)\) otherwise. If \(\text{transb} = 'N'\) or 'n', \(ldb\) must be at least \(\max(1, k)\); otherwise, \(ldb\) must be at least \(\max(1, n)\).
  \item \(\text{ldb}\) leading dimension of two-dimensional array used to store matrix \(B\).
  \item \(\text{beta}\) double-precision scalar multiplier applied to \(C\). If zero, \(C\) does not have to be a valid input.
  \item \(C\) double-precision array of dimensions \((ldc, n)\); \(ldc\) must be at least \(\max(1, m)\).
  \item \(\text{ldc}\) leading dimension of two-dimensional array used to store matrix \(C\).
\end{itemize}

Output

\(C\) updated based on \(C = \alpha \ast \text{op}(A) \ast \text{op}(B) + \beta \ast C\).

Reference: \url{http://www.netlib.org/blas/dgemm.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 \(m < 0, n < 0, \text{or } k < 0\)
- \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

Function \texttt{cublasDsymm()}

\begin{verbatim}
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)
\end{verbatim}

performs one of the matrix-matrix operations

\[
  C = \alpha \ast A \ast B + \beta \ast C \text{ or } C = \alpha \ast B \ast A + \beta \ast C,
\]

where \(\alpha\) and \(\beta\) are double-precision scalars, \(A\) is a symmetric matrix consisting of double-precision elements and is stored in either
lower or upper storage mode. \( B \) and \( C \) are \( m \times n \) matrices consisting of double-precision elements.

Input

- **side** specifies whether the symmetric matrix \( A \) appears on the left-hand side or right-hand side of matrix \( B \).
  - If \( \text{side} = 'L' \) 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** double-precision scalar multiplier applied to \( A B \).

- **A** double-precision array of dimensions \((\text{lda}, \text{ka})\), where \( \text{ka} \) is \( m \) when \( \text{side} = 'L' \) or \( 'l' \) and \( n \) 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.

- **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>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>B</code></td>
<td>double-precision array of dimensions <code>(ldb, n)</code>. On entry, the leading <code>m×n</code> part of the array contains the matrix <code>B</code>.</td>
</tr>
<tr>
<td><code>ldb</code></td>
<td>leading dimension of <code>B</code>; <code>ldb</code> must be at least <code>max(1, m)</code>.</td>
</tr>
<tr>
<td><code>beta</code></td>
<td>double-precision scalar multiplier applied to <code>C</code>. If <code>beta</code> is zero, <code>C</code> does not have to be a valid input.</td>
</tr>
<tr>
<td><code>C</code></td>
<td>double-precision array of dimensions <code>(ldc, n)</code>.</td>
</tr>
</tbody>
</table>
| `ldc`    | leading dimension of `C`; `ldc` must be at least `max(1, m)`.

Output

- `C` updated according to `C = alpha * A * B + beta * C` or `C = alpha * B * A + beta * C`.

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

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 * Aᵀ + beta * C or C = alpha * Aᵀ * 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` is a matrix consisting of double-precision elements with dimensions of `n×k` in the first case and `k×n` in the second case.
Chapter 5

BLAS3 Functions

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 \( \text{'n'} \),
\[
    C = \alpha \times A \times A^T + \beta \times C.
\]
If \( \text{trans} == \text{'T'} \), \( \text{'t'} \), \( \text{'C'} \), or \( \text{'c'} \),
\[
    C = \alpha \times A^T \times A + \beta \times C.
\]

\( n \) 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.

\( k \) specifies the number of columns of matrix \( A \). If \( \text{trans} == \text{'N'} \) or \( \text{'n'} \), \( k \) specifies the number of columns of matrix \( A \); \( k \) must be at least zero.

alpha double-precision scalar multiplier applied to \( A \times A^T \) or \( A^T \times A \).

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 \( A \) 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 double-precision scalar multiplier applied to \( C \). If \( \beta \) is zero, \( C \) is not read.

\[
    C = A A^T = A^T A
\]
Function cublasDsyr2k()

```c
void cublasDsyr2k (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

\[
C = \alpha A^T B^T + \alpha B^T A^T + \beta C \quad \text{or} \\
C = \alpha A^T B + \alpha B^T A + \beta C,
\]
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 + alpha * B * A + 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.
Function cublasDtrmm()

```c
void
cublasDtrmm (char side, char uplo, char transa,
char diag, int m, int n, double alpha,
const double *A, int lda, const double *B,
int ldb)
```

performs one of the matrix-matrix operations

\[ B = \alpha \cdot \text{op}(A) \cdot B \text{ or } B = \alpha \cdot B \cdot \text{op}(A), \]

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

Reference: [http://www.netlib.org/blas/dsy2k.f](http://www.netlib.org/blas/dsy2k.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 \( m < 0, n < 0, \text{ or } k < 0 \)
- `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
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 $\text{lda}$ and $\text{ldb}$ are the leading dimensions of the two-dimensional arrays that contain $A$ and $B$, respectively.

**Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{side}$</td>
<td>specifies whether $\text{op}(A)$ multiplies $B$ from the left or right. If $\text{side} == 'L' \text{ or } 'l'$, $B = \text{alpha} \times \text{op}(A) \times B$. If $\text{side} == 'R' \text{ or } 'r'$, $B = \text{alpha} \times B \times \text{op}(A)$.</td>
</tr>
<tr>
<td>$\text{uplo}$</td>
<td>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.</td>
</tr>
<tr>
<td>$\text{transa}$</td>
<td>specifies the form of $\text{op}(A)$ to be used in the matrix multiplication. If $\text{transa} == 'N' \text{ or } 'n'$, $\text{op}(A) = A$. If $\text{transa} == 'T' \text{, 't' , 'C' , or 'c'}$, $\text{op}(A) = A^T$.</td>
</tr>
<tr>
<td>$\text{diag}$</td>
<td>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.</td>
</tr>
<tr>
<td>$m$</td>
<td>the number of rows of matrix $B$; $m$ must be at least zero.</td>
</tr>
<tr>
<td>$n$</td>
<td>the number of columns of matrix $B$; $n$ must be at least zero.</td>
</tr>
<tr>
<td>$\text{alpha}$</td>
<td>double-precision scalar multiplier applied to $\text{op}(A) \times B \text{ or } B \times \text{op}(A)$, respectively. If $\text{alpha}$ is zero, no accesses are made to matrix $A$, and no read accesses are made to matrix $B$.</td>
</tr>
<tr>
<td>$A$</td>
<td>double-precision array of dimensions $(\text{lda}, k)$. 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.</td>
</tr>
<tr>
<td>$\text{lda}$</td>
<td>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.</td>
</tr>
</tbody>
</table>
Input (continued)

<table>
<thead>
<tr>
<th>B</th>
<th>double-precision array of dimensions (ldb, n). On entry, the leading m×n part of the array contains the matrix B. It is overwritten with the transformed matrix on exit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldb</td>
<td>leading dimension of B; ldb must be at least ( \max(1, m) ).</td>
</tr>
</tbody>
</table>

Output

| B | updated according to \( B = \alpha \cdot \text{op}(A) \cdot B \) or \( B = \alpha \cdot B \cdot \text{op}(A) \). |

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

\[
\text{op}(A) \cdot X = \alpha \cdot B \quad \text{or} \quad X \cdot \text{op}(A) = \alpha \cdot B,
\]

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

\( \alpha \) is a 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 \( lda \) and \( ldb \) are the leading dimensions of the two-dimensional arrays that contain \( A \) and \( B \), respectively.
Input

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| side     | specifies whether \( \text{op}(A) \) appears on the left or right of \( X \):
|          | \( \text{side} = 'L' \) or \( 'l' \) indicates solve \( \text{op}(A) \times X = \alpha \times B \); |
|          | \( \text{side} = 'R' \) 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:
|          | \( \text{uplo} = 'U' \) or \( 'u' \) indicates \( A \) is an upper triangular matrix; |
|          | \( \text{uplo} = 'L' \) or \( 'l' \) indicates \( A \) is a lower triangular matrix. |
| transa   | specifies the form of \( \text{op}(A) \) to be used in matrix multiplication.
|          | If \( \text{transa} = 'N' \) or \( 'n' \), \( \text{transa} = 'T' \), \( 't' \), \( 'C' \), or \( 'c' \), \( \text{transa} = \text{op}(A) = A^T \). |
| 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    | 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. |
| A        | 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 \( 'r' \). 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 \( 'l' \), \( \text{lda} \) must be at least \( \max(1,m) \). |
|          | When \( \text{side} = 'R' \) or \( 'r' \), \( \text{lda} \) must be at least \( \max(1,n) \). |
| B        | 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 \( X \). |
| ldb      | leading dimension of the two-dimensional array containing \( B \); \( \text{ldb} \) must be at least \( \max(1,m) \). |
### Output

\[ B \] contains the solution matrix \( X \) satisfying \( \text{op}(A) \times X = \alpha \times B \) or \( X \times \text{op}(A) = \alpha \times B \).

### 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

---

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

Error status for this function can be retrieved via `cublasGetError()`. 
Double-Precision Complex BLAS3 Functions

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

Five double-precision complex BLAS3 functions are implemented:

- “Function cublasZgemm()” on page 220
- “Function cublasZhemm()” on page 221
- “Function cublasZherk()” on page 223
- “Function cublasZher2k()” on page 226
- “Function cublasZsymm()” on page 228
- “Function cublasZsyrk()” on page 230
- “Function cublasZsyr2k()” on page 232
- “Function cublasZtrmm()” on page 234
- “Function cublasZtrsm()” on page 236
Function cublasZgemm()

void

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 \( \text{C} \) a \( m \times n \) 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 \( \text{op}(A) \) * \( \text{op}(B) \).

- **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

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

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

Error Status

<table>
<thead>
<tr>
<th>StatusCode</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 cublasZhemm()

```c
void 
cublasZhemm (char side, char uplo, int m, int n, 
cuDenseDoubleComplex alpha, 
const cuDenseDoubleComplex *A, int lda, 
const cuDenseDoubleComplex *B, int ldb, 
cuDenseDoubleComplex beta, cuDenseDoubleComplex *C, 
int ldc)
```

performs one of the matrix-matrix operations

\[
C = \alpha \circ A + B + \beta \circ C \quad \text{or} \quad C = \alpha \circ B + A + \beta \circ C,
\]

where \(\alpha\) and \(\beta\) are double-precision complex scalars, \(A\) is a Hermitian matrix consisting of double-precision complex elements
and is stored in either lower or upper storage mode. B and C are \( m \times n \) 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 side == 'L' or 'l', \( C = \alpha A B^H + \beta C \).
  - If side == 'R' or 'r', \( C = \alpha B A^H + \beta C \).

- **uplo**: specifies whether the Hermitian matrix \( A \) is stored in upper or lower storage mode. If uplo == 'U' or 'u', only the upper triangular part of the Hermitian matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If uplo == 'L' or 'l', only the lower triangular part of the Hermitian matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part.

- **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 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 side == 'R' or 'r'; \( n \) must be at least zero.

- **alpha**: double-precision complex scalar multiplier applied to \( A B \) or \( B^H A \).

- **A**: double-precision complex array of dimensions \((lda, ka)\), where \( ka \) is \( m \) when side == 'L' or 'l' and is \( n \) otherwise. If side == 'L' or 'l', the leading \( m \times m \) part of array \( A \) must contain the Hermitian matrix such that when 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 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 side == 'R' or 'r', the leading \( n \times n \) part of array \( A \) must contain the Hermitian matrix such that when 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 uplo == 'L' or 'l', the leading \( n \times n \) part stores the lower triangular part of the Hermitian matrix, and the strictly upper triangular part is not referenced. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero.

- **lda**: leading dimension of \( A \). When side == 'L' or 'l', it must be at least \( \max(i, m) \) 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×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

\[
C = \alpha A B + \beta C
\]

or

\[
C = \alpha B A + \beta C
\]

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

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

<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_INDEX</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 `cublasZherk()`

```c
void

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

\[
C = \alpha A A^H + \beta C \quad \text{or} \quad C = \alpha A^H A + \beta C
\]

where alpha and beta are double-precision scalars. C is an n×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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uplo</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>trans</strong></td>
<td>specifies the operation to be performed. If ( \text{trans} == 'N' ) or ( 'n' ), ( C = \alpha A \ast A^\ast + \beta C ). If ( \text{trans} == 'T', 't', 'C', ) or ( 'c' ), ( C = \alpha A^\ast \ast A + \beta C ).</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>k</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>alpha</strong></td>
<td>double-precision scalar multiplier applied to ( A \ast A^\ast ) or ( A^\ast \ast A ).</td>
</tr>
<tr>
<td><strong>A</strong></td>
<td>double-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 ).</td>
</tr>
<tr>
<td><strong>lda</strong></td>
<td>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) ).</td>
</tr>
<tr>
<td><strong>beta</strong></td>
<td>double-precision scalar multiplier applied to ( C ). If ( \beta ) is zero, ( C ) does not have to be a valid input.</td>
</tr>
</tbody>
</table>

\[
C \alpha A A^\ast H \beta C^* + ** = C \alpha A^\ast H A^\ast* + ** = \alpha A A^\ast * A^\ast * A^\ast * A^\ast *
\]
CHAPTER 5  BLAS3 Functions

Input (continued)

C  double-precision complex array of dimensions \((l_{dc}, n)\). 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 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 \(\text{uplo} == 'L'\) or \('l'\), the leading \(n \times n\) triangular part of the array \(C\) must contain the lower triangular part of the Hermitian matrix \(C\), and the strictly upper triangular part of \(C\) is not referenced. On exit, the lower triangular part of \(C\) is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.

\(l_{dc}\)  leading dimension of \(C\); \(l_{dc}\) must be at least \(\max(1, n)\).

Output

updated according to \(C = \alpha A^H + \beta C\) 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 \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 (n &lt; 0) or (k &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 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

\[
\begin{align*}
C &= \alpha A \times B^H + \alpha B^H \times A + \beta C \\
C &= \alpha A^H \times B + \alpha B^H \times A + \beta C,
\end{align*}
\]

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', \(C = \alpha A \times B^H + \alpha B^H \times A + \beta C\). If `trans` = 'T', 't', 'C', or 'c', \(C = \alpha A^H \times B + \alpha B^H \times A + \beta C\).
- `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

Input (continued)

**A**
- double-precision array of dimensions \((lda, ka)\), where \(ka\) is \(k\) when \(\text{trans} == 'N'\) or \(\text{trans} == 'n'\) and is \(n\) otherwise. When \(\text{trans} == 'N'\) or \(\text{trans} == 'n'\), 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 \(\text{trans} == 'n'\), \(lda\) 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 \(\text{trans} == 'N'\) or \(\text{trans} == 'n'\) and is \(n\) otherwise. When \(\text{trans} == 'N'\) or \(\text{trans} == 'n'\), 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 \(\text{trans} == 'N'\) or \(\text{trans} == 'n'\), \(ldb\) 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 \(\text{uplo} == 'U'\) or \(\text{uplo} == 'u'\), the leading \(n \times n\) triangular part of the array \(C\) must contain the upper triangular part of the Hermitian matrix \(C\), and the strictly lower triangular part of \(C\) is not referenced. On exit, the upper triangular part of \(C\) is overwritten by the upper triangular part of the updated matrix. If \(\text{uplo} == 'L'\) or \(\text{uplo} == 'l'\), the leading \(n \times n\) triangular part of the array \(C\) must contain the lower triangular part of the Hermitian matrix \(C\), and the strictly upper triangular part of \(C\) is not referenced. On exit, the lower triangular part of \(C\) is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set; they are assumed to be zero, and on exit they are set to zero.

**ldc**
- leading dimension of \(C\); \(ldc\) must be at least \(\max(1, n)\).

Output

**C**
- updated according to 
  \[
  C = alpha \ast A \ast B^H + alpha \ast B \ast A^H + beta \ast C \\
  \text{or } C = alpha \ast A^H \ast B + alpha \ast B^H \ast A + beta \ast C.
  \]

Reference: [http://www.netlib.org/blas/zher2k.f](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 \(k < 0\)
# 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

\[
C = \alpha A + \beta B \quad \text{or} \quad C = \alpha B + \beta A,
\]

where \(\alpha\) and \(\beta\) are double-precision complex scalars, \(A\) is a symmetric matrix consisting of double-precision complex elements and is stored in either lower or upper storage mode. \(B\) and \(C\) are \(m \times n\) matrices consisting of double-precision complex elements.

## Input

**side** specifies whether the symmetric matrix \(A\) appears on the left-hand side or right-hand side of matrix \(B\).

- If `side == 'L'` or `'l'`, \(C = \alpha A * B + \beta * C\).
- If `side == 'R'` or `'r'`, \(C = \alpha B * A + \beta * C\).

**uplo** specifies whether the symmetric matrix \(A\) is stored in upper or lower storage mode. If `uplo == 'U'` or `'u'`, only the upper triangular part of the symmetric matrix is referenced, and the elements of the strictly lower triangular part are inferred from those in the upper triangular part. If `uplo == 'L'` or `'l'`, only the lower triangular part of the symmetric matrix is referenced, and the elements of the strictly upper triangular part are inferred from those in the lower triangular part.

**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 `side == 'L'` or `'l'; m must be at least zero.

## Error Status (continued)

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA_STATUS_ARCH_MISMATCH</td>
<td>if function invoked on device that does not support double precision</td>
</tr>
<tr>
<td>CUDA_STATUS_EXECUTION_FAILED</td>
<td>if function failed to launch on GPU</td>
</tr>
</tbody>
</table>
CHAPTER 5  BLAS3 Functions

Input (continued)

\begin{itemize}
\item \(n\) 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.
\item \(\alpha\) double-precision complex scalar multiplier applied to \(A \times B\) or \(B \times A\).
\item \(A\) 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 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.
\item \(\text{lda}\) 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.
\item \(B\) double-precision complex array of dimensions \((\text{ldb}, n)\). On entry, the leading \(m\times n\) part of the array contains the matrix \(B\).
\item \(\text{ldb}\) leading dimension of \(B\); \(\text{ldb}\) must be at least \(\max(1, m)\).
\item \(\beta\) double-precision complex scalar multiplier applied to \(C\). If \(\beta\) is zero, \(C\) does not have to be a valid input.
\item \(C\) double-precision complex array of dimensions \((\text{ldc}, n)\).
\item \(\text{ldc}\) leading dimension of \(C\); \(\text{ldc}\) must be at least \(\max(1, m)\).
\end{itemize}

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/zsymm.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 `cublasZsyrk()`

```c
void

cublasZsyrk (char uplo, char trans, int n, int k,
cuDoubleComplex alpha,
const cuDoubleComplex *A, int lda,
cuDoubleComplex beta,
cuDoubleComplex *C, int ldc)
```

performs one of the symmetric rank k operations

\[
C = \alpha A A^T + \beta C \quad \text{or} \quad C = \alpha A^T A + \beta C,
\]

where \( \alpha \) and \( \beta \) are double-precision complex scalars. \( C \) is an \( n \times n \) symmetric matrix consisting of double-precision complex elements and is stored in either lower or upper storage mode. \( A \) is a matrix consisting of double-precision complex elements with dimensions of \( n \times k \) in the first case and \( k \times n \) in the second case.

Input

- **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'`, \( C = \alpha A A^T + \beta C \). If `trans == 'T'`, `'t'`, `'C'`, or `'c'`, \( C = \alpha A^T A + \beta C \).
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 \( \text{trans} = 'T', 't', 'C', \) 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 \( \text{trans} = 'T', 't', 'C', \) or \( 'c' \), \( k \) specifies the number of rows of matrix \( A \); \( k \) must be at least zero.

\( \alpha \) double-precision complex scalar multiplier applied to \( A \) or \( A^T \).

\( A \) double-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 \( \max(1, k) \).

\( \beta \) double-precision complex scalar multiplier applied to \( C \).

If \( \beta \) is zero, \( C \) is not read.

\( C \) 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.

\( \text{ldc} \) leading dimension of \( C \); \( \text{ldc} \) 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/zsyrk.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 ( 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 `cublasZsyr2k()`

```c
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 2\( k \) operations

\[
C = \alpha \cdot A \cdot B^T + \alpha \cdot B \cdot A^T + \beta \cdot C \quad \text{or} \quad C = \alpha \cdot A^T \cdot B + \alpha \cdot B^T \cdot A + \beta \cdot C,
\]

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 \( \text{uplo} = \text{'U'} \) 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 \( \text{'n'} \),

\[
C = \alpha \cdot A \cdot B^T + \alpha \cdot B \cdot A^T + \beta \cdot C.
\]

If \( \text{trans} = \text{'T'} \), \( \text{'t'} \), \( \text{'C'} \), or \( \text{'c'} \),

\[
C = \alpha \cdot A^T \cdot B + \alpha \cdot B^T \cdot A + \beta \cdot C.
\]
\textbf{Input (continued)}

\begin{itemize}
  \item \textbf{n} \hspace{1cm} \text{specifies the number of rows and the number columns of matrix C. If trans == 'N' or 'n', n specifies the number of rows of matrices A and B. If trans == 'T', 't', 'C', or 'c', n specifies the number of columns of matrices A and B; n must be at least zero.}
  \item \textbf{k} \hspace{1cm} \text{If trans == 'N' or 'n', k specifies the number of columns of matrices A and B. If trans == 'T', 't', 'C', or 'c', k specifies the number of rows of matrices A and B; k must be at least zero.}
  \item \textbf{alpha} \hspace{1cm} \text{double-precision scalar multiplier.}
  \item \textbf{A} \hspace{1cm} \text{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.}
  \item \textbf{lda} \hspace{1cm} \text{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).}
  \item \textbf{B} \hspace{1cm} \text{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.}
  \item \textbf{ldb} \hspace{1cm} \text{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).}
  \item \textbf{beta} \hspace{1cm} \text{double-precision scalar multiplier applied to C. If beta is zero, C does not have to be a valid input.}
  \item \textbf{C} \hspace{1cm} \text{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.}
  \item \textbf{ldc} \hspace{1cm} \text{leading dimension of C; ldc must be at least max(1, n).}
\end{itemize}
Output
\[
C \quad \text{updated according to}
\]
\[
C = \alpha A^T B + \alpha B^T A + \beta C \quad \text{or}
\]
\[
C = \alpha A^T B + \alpha B^T A + \beta C.
\]

Reference: \[http://www.netlib.org/blas/zsyr2k.f\]

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

Error Status

\begin{itemize}
\item \texttt{CUBLAS_STATUS_NOT_INITIALIZED} if CUBLAS library was not initialized
\item \texttt{CUBLAS_STATUS_INVALID_VALUE} if \(m < 0, n < 0\), or \(k < 0\)
\item \texttt{CUBLAS_STATUS_ARCH_MISMATCH} if function invoked on device that does not support double precision
\item \texttt{CUBLAS_STATUS_EXECUTION_FAILED} if function failed to launch on GPU
\end{itemize}

Function \texttt{cublasZtrmm()}

\begin{verbatim}
void
cublasZtrmm (char side, char uplo, char transa,
   char diag, int m, int n,
   cuDoubleComplex alpha,
   const cuDoubleComplex *A, int lda,
   const cuDoubleComplex *B, int ldb)
\end{verbatim}

performs one of the matrix-matrix operations
\[
B = \alpha \text{op}(A) \times B \quad \text{or} \quad B = \alpha B \times \text{op}(A),
\]
where \(\text{op}(A) = A, \text{op}(A) = A^T, \text{or} \quad \text{op}(A) = A^H;\)

\(\alpha\) is a double-precision complex scalar; \(B\) is an \(m \times n\) matrix consisting of double-precision complex elements; and \(A\) is a unit or non-unit, upper or lower triangular matrix consisting of double-precision complex elements.

Matrices \(A\) and \(B\) are stored in column-major format, and \(lda\) and \(ldb\) are the leading dimensions of the two-dimensional arrays that contain \(A\) and \(B\), respectively.
The image contains a page from a document discussing functions in the BLAS3 library, focusing on the parameters and descriptions for these 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. When \( \text{diag} == 'U' \) or 'u', the diagonal elements of \( A \) are not referenced and are assumed to be unity.

- **lda**: Leading dimension of \( A \). When \( \text{side} == 'L' \) 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 \( \max(1, m) \).
CUDA CUBLAS Library

Error status for this function can be retrieved via `cublasGetError()`. The error status can be one of the following:

- `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 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

\[
\text{op}(A) \times X = \text{alpha} \times B \quad \text{or} \quad X \times \text{op}(A) = \text{alpha} \times B,
\]

where \( \text{op}(A) = A, \text{op}(A) = A^T, \) 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 \( X \):
- \( \text{side} == 'L' \) or \('l' \) indicates solve \( \text{op}(A) \times X = \alpha \times B \);
- \( \text{side} == 'R' \) 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:
- \( \text{uplo} == 'U' \) 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 \('n' \), \( \text{op}(A) = A \).
- 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 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 \((\text{lda}, k)\), 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 \('l' \), \( \text{lda} \) must be at least \( \max(1, m) \).
- When \( \text{side} == 'R' \) or \('r' \), \( \text{lda} \) must be at least \( \max(1, n) \).

\( B \) double-precision complex 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 \( X \).

ldb leading dimension of the two-dimensional array containing \( B \); \( \text{ldb} \) must be at least \( \max(1, m) \).
Output

\[ \begin{align*}
B & \quad \text{contains the solution matrix } X \text{ satisfying } \alpha \cdot \text{op}(A) \cdot X = \alpha \cdot B \text{ or } \\
X \cdot \text{op}(A) & = \alpha \cdot B.
\end{align*} \]

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><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 ) or ( n &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>
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 32-bit integers on the Fortran side, regardless of whether the host platform is a 32-bit or 64-bit platform.

One approach to deal with index arithmetic on device pointers in Fortran code is to use C-style macros, and use the C preprocessor to expand these, as shown in the example below. On Linux and Mac OS X, one way of pre-processing is to invoke `g77 -E -x f77-cpp-`

When traditional fixed-form Fortran 77 code is ported to CUBLAS, line length often increases when the BLAS calls are exchanged for CUBLAS calls. Longer function names and possible macro expansion are contributing factors. Inadvertently exceeding the maximum line length can lead to run-time errors that are difficult to find, so care should be taken not to exceed the 72-column limit if fixed form is retained.

The following two examples show a small application implemented in Fortran 77 on the host (Example A.1., “Fortran 77 Application Executing on the Host” on page 242), and show the same application using the non-thunking wrappers after it has been ported to use CUBLAS (Example A.2., “Fortran 77 Application Ported to Use CUBLAS” on page 243).
Example A.1. Fortran 77 Application Executing on the Host

```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
    enddo
  enddo
  call modify (a, M, N, 2, 3, 16.0, 12.0)
  do j = 1, N
    do i = 1, M
      write(*,"
         F7.0") a(i,j)
    enddo
    write (*,*) ""  
  enddo
  stop
end
```
Example A.2. Fortran 77 Application Ported to Use CUBLAS

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

subroutine modify (devPtrM, ldm, n, p, q, alpha, beta)
imPLICIT none
integer sizeof_real
parameter (sizeof_real=4)
integer ldm, n, p, q, devPtrM
real*4 alpha, beta
call cublas_sscal (n-p+1, alpha,
1                   devPtrM+IDX2F(p,q,ldm)*sizeof_real,
2                   ldm)
call cublas_sscal (ldm-p+1, beta,
1                   devPtrM+IDX2F(p,q,ldm)*sizeof_real,
2                   1)
return
derend

program matrixmod
implicit none
integer M, N, sizeof_real, devPtrA
parameter (M=6, N=5, sizeof_real=4)
real*4 a(M,N)
integer i, j, stat
external cublas_init, cublas_set_matrix, cublas_get_matrix
external cublas_shutdown, cublas_alloc
integer cublas_alloc, cublas_set_matrix, cublas_get_matrix
do j = 1, N
do i = 1, M
   a(i,j) = (i-1) * M + j
endo
dendo
call cublas_init
stat = cublas_alloc(M*N, sizeof_real, devPtrA)
if (stat .NE. 0) then
```

This code demonstrates how to port a Fortran 77 application to use CUBLAS functions for matrix operations.

The `modify` subroutine uses `cublas_sscal` to scale the matrix elements with scalars `alpha` and `beta`. The `matrixmod` program initializes CUBLAS, allocates memory, fills the matrix `a`, and calls the `cublas_init` function to ensure compatibility with CUBLAS functions.

The `define IDX2F(i, j, ld)` macro is used to define a function for calculating the index in Fortran 77 style, which is useful for accessing matrix elements in a Fortran 77 application.

---

NVIDIA

CHAPTER A CUBLAS Fortran Bindings

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)
imPLICIT none
integer sizeof_real
parameter (sizeof_real=4)
integer ldm, n, p, q, devPtrM
real*4 alpha, beta
call cublas_sscal (n-p+1, alpha,
1                   devPtrM+IDX2F(p,q,ldm)*sizeof_real,
2                   ldm)
call cublas_sscal (ldm-p+1, beta,
1                   devPtrM+IDX2F(p,q,ldm)*sizeof_real,
2                   1)
return
derend

program matrixmod
implicit none
integer M, N, sizeof_real, devPtrA
parameter (M=6, N=5, sizeof_real=4)
real*4 a(M,N)
integer i, j, stat
external cublas_init, cublas_set_matrix, cublas_get_matrix
external cublas_shutdown, cublas_alloc
integer cublas_alloc, cublas_set_matrix, cublas_get_matrix
do j = 1, N
do i = 1, M
   a(i,j) = (i-1) * M + j
endo
dendo
call cublas_init
stat = cublas_alloc(M*N, sizeof_real, devPtrA)
if (stat .NE. 0) then
```
Example A.2. Fortran 77 Application Ported to Use CUBLAS (continued)

```fortran
write(*,*) "device memory allocation failed"
call cublas_shutdown
stop
endif
stat = cublas_set_matrix (M, N, sizeof_real, a, M, devPtrA, M)
if (stat .NE. 0) then
    call cublas_free (devPtrA)
    write(*,*) "data download failed"
call cublas_shutdown
stop
endif
call modify (devPtrA, M, N, 2, 3, 16.0, 12.0)
stat = cublas_get_matrix (M, N, sizeof_real, devPtrA, M, a, M)
if (stat .NE. 0) then
    call cublas_free (devPtrA)
    write(*,*) "data upload failed"
call cublas_shutdown
stop
endif
call cublas_free (devPtrA)
call cublas_shutdown
do j = 1, N
do i = 1, M
   write(*,"(F7.0$)"") a(i,j)
endo
dodo
   write (*,"")
endo
call cublas_shutdown
stop
end
```