

---

# **hipBLAS Documentation**

**Advanced Micro Devices**

**Nov 10, 2022**



## CONTENTS:

<b>1</b>	<b>User Guide</b>	<b>1</b>
1.1	Disclaimer	1
1.2	Introduction	1
1.3	Building and Installing	2
1.3.1	Prerequisites	2
1.3.2	Installing pre-built packages	2
1.3.3	hipBLAS build	2
1.3.3.1	Build library dependencies + library	2
1.3.3.2	Build library dependencies + client dependencies + library + client	3
1.3.4	Dependencies	3
1.3.5	Manual build (all supported platforms)	3
1.3.5.1	Dependencies For Building Library	3
1.3.5.2	Build Library Using Individual Commands	3
1.3.5.3	Build Library + Tests + Benchmarks + Samples Using Individual Commands	3
	Dependencies (only necessary for hipBLAS clients)	4
	(optional, one time only)	4
1.4	Library Source Code Organization	4
1.4.1	The <i>library</i> directory	4
1.4.1.1	library/include	4
1.4.1.2	library/src/hcc_detail	4
1.4.1.3	library/src/nvcc_detail	5
1.4.1.4	library/src/include	5
1.4.2	The <i>clients</i> directory	5
1.4.2.1	clients/gtest	5
1.4.2.2	clients/benchmarks	5
1.4.2.3	clients/include	5
1.4.2.4	clients/common	5
1.4.2.5	clients/samples	5
1.4.3	Infrastructure	5
1.5	API	6
1.5.1	hipBLAS interface examples	6
1.5.2	GEMV API	6
1.5.3	Batched and strided GEMM API	6
1.6	Guidelines	6
1.6.1	Naming conventions	6
1.6.2	Notations	7
1.7	hipBLAS Types	7
1.7.1	Definitions	7
1.7.1.1	hipblasHandle_t	7
1.7.1.2	hipblasHalf	7

1.7.1.3	hipblasInt8 . . . . .	7
1.7.1.4	hipblasStride . . . . .	7
1.7.1.5	hipblasBfloat16 . . . . .	8
1.7.1.6	hipblasComplex . . . . .	8
1.7.1.7	hipblasDoubleComplex . . . . .	8
1.7.2	Enums . . . . .	8
1.7.2.1	hipblasStatus_t . . . . .	8
1.7.2.2	hipblasOperation_t . . . . .	9
1.7.2.3	hipblasPointerMode_t . . . . .	9
1.7.2.4	hipblasFillMode_t . . . . .	10
1.7.2.5	hipblasDiagType_t . . . . .	10
1.7.2.6	hipblasSideMode_t . . . . .	10
1.7.2.7	hipblasDatatype_t . . . . .	11
1.7.2.8	hipblasGemmAlgo_t . . . . .	12
1.7.2.9	hipblasAtomicsMode_t . . . . .	12
1.8	hipBLAS Functions . . . . .	12
1.8.1	Level 1 BLAS . . . . .	12
1.8.1.1	hipblasIXamax + Batched, StridedBatched . . . . .	13
1.8.1.2	hipblasIXamin + Batched, StridedBatched . . . . .	15
1.8.1.3	hipblasXasum + Batched, StridedBatched . . . . .	17
1.8.1.4	hipblasXaxpy + Batched, StridedBatched . . . . .	19
1.8.1.5	hipblasXcopy + Batched, StridedBatched . . . . .	21
1.8.1.6	hipblasXdot + Batched, StridedBatched . . . . .	23
1.8.1.7	hipblasXnrm2 + Batched, StridedBatched . . . . .	27
1.8.1.8	hipblasXrot + Batched, StridedBatched . . . . .	29
1.8.1.9	hipblasXrotg + Batched, StridedBatched . . . . .	32
1.8.1.10	hipblasXrotm + Batched, StridedBatched . . . . .	34
1.8.1.11	hipblasXrotmg + Batched, StridedBatched . . . . .	36
1.8.1.12	hipblasXscal + Batched, StridedBatched . . . . .	39
1.8.1.13	hipblasXswap + Batched, StridedBatched . . . . .	41
1.8.2	Level 2 BLAS . . . . .	43
1.8.2.1	hipblasXgbmv + Batched, StridedBatched . . . . .	44
1.8.2.2	hipblasXgemv + Batched, StridedBatched . . . . .	48
1.8.2.3	hipblasXger + Batched, StridedBatched . . . . .	52
1.8.2.4	hipblasXhbmvm + Batched, StridedBatched . . . . .	55
1.8.2.5	hipblasXhemv + Batched, StridedBatched . . . . .	59
1.8.2.6	hipblasXher + Batched, StridedBatched . . . . .	62
1.8.2.7	hipblasXher2 + Batched, StridedBatched . . . . .	64
1.8.2.8	hipblasXhpmv + Batched, StridedBatched . . . . .	67
1.8.2.9	hipblasXhpr + Batched, StridedBatched . . . . .	71
1.8.2.10	hipblasXhpr2 + Batched, StridedBatched . . . . .	74
1.8.2.11	hipblasXsbmv + Batched, StridedBatched . . . . .	77
1.8.2.12	hipblasXspmv + Batched, StridedBatched . . . . .	80
1.8.2.13	hipblasXspr + Batched, StridedBatched . . . . .	82
1.8.2.14	hipblasXspr2 + Batched, StridedBatched . . . . .	85
1.8.2.15	hipblasXsymv + Batched, StridedBatched . . . . .	88
1.8.2.16	hipblasXsyr + Batched, StridedBatched . . . . .	92
1.8.2.17	hipblasXsyr2 + Batched, StridedBatched . . . . .	95
1.8.2.18	hipblasXtbmv + Batched, StridedBatched . . . . .	98
1.8.2.19	hipblasXtbsv + Batched, StridedBatched . . . . .	102
1.8.2.20	hipblasXtpmv + Batched, StridedBatched . . . . .	105
1.8.2.21	hipblasXtpsv + Batched, StridedBatched . . . . .	108
1.8.2.22	hipblasXtrmv + Batched, StridedBatched . . . . .	111
1.8.2.23	hipblasXtrsv + Batched, StridedBatched . . . . .	114

1.8.3	Level 3 BLAS . . . . .	117
1.8.3.1	hipblasXgemm + Batched, StridedBatched . . . . .	117
1.8.3.2	hipblasXherk + Batched, StridedBatched . . . . .	121
1.8.3.3	hipblasXherkx + Batched, StridedBatched . . . . .	124
1.8.3.4	hipblasXher2k + Batched, StridedBatched . . . . .	128
1.8.3.5	hipblasXsymm + Batched, StridedBatched . . . . .	132
1.8.3.6	hipblasXsyrk + Batched, StridedBatched . . . . .	136
1.8.3.7	hipblasXsyr2k + Batched, StridedBatched . . . . .	139
1.8.3.8	hipblasXsyrkx + Batched, StridedBatched . . . . .	143
1.8.3.9	hipblasXgeam + Batched, StridedBatched . . . . .	148
1.8.3.10	hipblasXhemm + Batched, StridedBatched . . . . .	151
1.8.3.11	hipblasXtrmm + Batched, StridedBatched . . . . .	155
1.8.3.12	hipblasXtrsm + Batched, StridedBatched . . . . .	159
1.8.3.13	hipblasXtrtri + Batched, StridedBatched . . . . .	164
1.8.3.14	hipblasXdgmm + Batched, StridedBatched . . . . .	167
1.8.4	SOLVER API . . . . .	170
1.8.4.1	hipblasXgetrf + Batched, stridedBatched . . . . .	170
1.8.4.2	hipblasXgetrs + Batched, stridedBatched . . . . .	174
1.8.4.3	hipblasXgetri + Batched, stridedBatched . . . . .	178
1.8.4.4	hipblasXgeqrf + Batched, stridedBatched . . . . .	179
1.8.5	BLAS Extensions . . . . .	183
1.8.5.1	hipblasGemmEx + Batched, StridedBatched . . . . .	184
1.8.5.2	hipblasTrsmEx + Batched, StridedBatched . . . . .	187
1.8.5.3	hipblasAxyEx + Batched, StridedBatched . . . . .	192
1.8.5.4	hipblasDotEx + Batched, StridedBatched . . . . .	194
1.8.5.5	hipblasDotcEx + Batched, StridedBatched . . . . .	196
1.8.5.6	hipblasNrm2Ex + Batched, StridedBatched . . . . .	198
1.8.5.7	hipblasRotEx + Batched, StridedBatched . . . . .	200
1.8.5.8	hipblasScalEx + Batched, StridedBatched . . . . .	202
1.8.6	Auxiliary . . . . .	204
1.8.6.1	hipblasCreate . . . . .	204
1.8.6.2	hipblasDestroy . . . . .	204
1.8.6.3	hipblasSetStream . . . . .	204
1.8.6.4	hipblasGetStream . . . . .	204
1.8.6.5	hipblasSetPointerMode . . . . .	204
1.8.6.6	hipblasGetPointerMode . . . . .	204
1.8.6.7	hipblasSetVector . . . . .	204
1.8.6.8	hipblasGetVector . . . . .	205
1.8.6.9	hipblasSetMatrix . . . . .	205
1.8.6.10	hipblasGetMatrix . . . . .	205
1.8.6.11	hipblasSetVectorAsync . . . . .	206
1.8.6.12	hipblasGetVectorAsync . . . . .	206
1.8.6.13	hipblasSetMatrixAsync . . . . .	207
1.8.6.14	hipblasGetMatrixAsync . . . . .	207
1.8.6.15	hipblasSetAtomicsMode . . . . .	208
1.8.6.16	hipblasGetAtomicsMode . . . . .	208
1.8.6.17	hipblasStatusToString . . . . .	208
1.9	Contributing . . . . .	208
1.9.1	Pull-request guidelines . . . . .	208
1.9.2	Coding Guidelines: . . . . .	208
1.9.2.1	Format . . . . .	209
1.9.3	Static Code Analysis . . . . .	209

## 2 Design Documentation

211

2.1	Clients . . . . .	211
2.1.1	hipblas-bench . . . . .	211
2.1.2	hipblas-test . . . . .	212
	<b>Index</b>	<b>213</b>

## 1.1 Disclaimer

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

- AMD, the AMD Arrow logo, Radeon, Ryzen, Epyc, and combinations thereof are trademarks of Advanced Micro Devices, Inc.
- Google(R) is a registered trademark of Google LLC.
- PCIe(R) is a registered trademark of PCI-SIG Corporation.
- Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
- Ubuntu and the Ubuntu logo are registered trademarks of Canonical Ltd.
- Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

## 1.2 Introduction

AMD **ROCm** has two classification of libraries,

- **roc\*** : AMD GPU Libraries, written in **HIP**.
- **hip\*** : AMD CPU library that is a thin interface to either AMD **roc\*** or Nvidia **cu\*** libraries.

Users targeting both CUDA and AMD devices must use the **hip\*** libraries.

hipBLAS is a BLAS marshaling library with multiple supported backends. It sits between the application and a 'worker' BLAS library, marshalling inputs into the backend library and marshalling results back to the application. hipBLAS exports an interface that does not require the client to change, regardless of the chosen backend. Currently, it supports **rocBLAS** and **cuBLAS** as backends.

## 1.3 Building and Installing

### 1.3.1 Prerequisites

- If using the rocBLAS backend on an AMD machine:
  - A ROCm enabled platform, more information [ROCm Documentation](#).
  - A compatible version of rocBLAS
  - A compatible version of rocSOLVER for full functionality
- If using the cuBLAS backend on a Nvidia machine:
  - A HIP enabled platform, more information [HIP installation](#).
  - A working CUDA toolkit, including cuBLAS, see [CUDA toolkit](#).

### 1.3.2 Installing pre-built packages

Download pre-built packages either from [ROCm's package servers](#) or by clicking the GitHub releases tab and manually downloading, which could be newer. Release notes are available for each release on the releases tab.

### 1.3.3 hipBLAS build

#### 1.3.3.1 Build library dependencies + library

The root of this repository has a helper bash script *install.sh* to build and install hipBLAS with a single command. It does take a lot of options and hard-codes configuration that can be specified through invoking cmake directly, but it's a great way to get started quickly and can serve as an example of how to build/install. A few commands in the script need sudo access so that it may prompt you for a password.

Typical uses of *install.sh* to build (library dependencies + library) are in the table below.

Command	Description
<code>./install.sh -h</code>	Help information.
<code>./install.sh -d</code>	Build library dependencies and library in your local directory. The <code>-d</code> flag only needs to be used once. For subsequent invocations of <i>install.sh</i> it is not necessary to rebuild the dependencies.
<code>./install.sh</code>	Build library in your local directory. It is assumed dependencies have been built.
<code>./install.sh -i</code>	Build library, then build and install hipBLAS package in <code>/opt/rocm/hipblas</code> . You will be prompted for sudo access. This will install for all users. If you want to keep hip-BLAS in your local directory, you do not need the <code>-i</code> flag.



### 1.3.3.2 Build library dependencies + client dependencies + library + client

The client contains executables in the table below.

executable name	description
hipblas-test	runs Google Tests to test the library
hipblas-bench	executable to benchmark or test individual functions
example-sscal	example C code calling hipblas_sscal function

Common uses of install.sh to build (dependencies + library + client) are in the table below.

Command	Description
<code>./install.sh -h</code>	Help information.
<code>./install.sh -dc</code>	Build library dependencies, client dependencies, library, and client in your local directory. The -d flag only needs to be used once. For subsequent invocations of install.sh it is not necessary to rebuild the dependencies.
<code>./install.sh -c</code>	Build library and client in your local directory. It is assumed the dependencies have been built.
<code>./install.sh -idc</code>	Build library dependencies, client dependencies, library, client, then build and install the hipBLAS package. You will be prompted for sudo access. It is expected that if you want to install for all users you use the -i flag. If you want to keep hipBLAS in your local directory, you do not need the -i flag.
<code>./install.sh -ic</code>	Build and install hipBLAS package, and build the client. You will be prompted for sudo access. This will install for all users. If you want to keep hipBLAS in your local directory, you do not need the -i flag.

### 1.3.4 Dependencies

Dependencies are listed in the script install.sh. Use `install.sh` with `-d` option to install dependencies. CMake has a minimum version requirement listed in the file install.sh. See `-cmake_install` flag in install.sh to upgrade automatically.

### 1.3.5 Manual build (all supported platforms)

This section has useful information on how to configure cmake and manually build.

#### 1.3.5.1 Dependencies For Building Library

#### 1.3.5.2 Build Library Using Individual Commands

#### 1.3.5.3 Build Library + Tests + Benchmarks + Samples Using Individual Commands

The repository contains source for clients that serve as samples, tests and benchmarks. Clients source can be found in the clients subdir.

### Dependencies (only necessary for hipBLAS clients)

The hipBLAS samples have no external dependencies, but our unit test and benchmarking applications do. These clients introduce the following dependencies:

- `lapack`, lapack itself brings a dependency on a fortran compiler
- `googletest`

Unfortunately, googletest and lapack are not as easy to install. Many distros do not provide a googletest package with pre-compiled libraries, and the lapack packages do not have the necessary cmake config files for cmake to configure linking the cblas library. hipBLAS provide a cmake script that builds the above dependencies from source. This is an optional step; users can provide their own builds of these dependencies and help cmake find them by setting the `CMAKE_PREFIX_PATH` definition. The following is a sequence of steps to build dependencies and install them to the cmake default `/usr/local`.

### (optional, one time only)

Once dependencies are available on the system, it is possible to configure the clients to build. This requires a few extra cmake flags to the library cmake configure script. If the dependencies are not installed into system defaults (like `/usr/local`), you should pass the `CMAKE_PREFIX_PATH` to cmake to help find them.

## 1.4 Library Source Code Organization

The hipBLAS code is split into two major parts:

- The *library* directory contains all source code for the library.
- The *clients* directory contains all test code and code to build clients.
- Infrastructure

### 1.4.1 The *library* directory

#### 1.4.1.1 `library/include`

Contains C98 include files for the external API. These files also contain Doxygen comments that document the API.

#### 1.4.1.2 `library/src/hcc_detail`

Implementation of hipBLAS interface compatible with rocBLAS APIs.

#### 1.4.1.3 library/src/nvcc\_detail

Implementation of hipBLAS interface compatible with cuBLAS-v2 APIs.

#### 1.4.1.4 library/src/include

Internal include files for:

- Converting C++ exceptions to hipBLAS status.

### 1.4.2 The *clients* directory

#### 1.4.2.1 clients/gtest

Code for client hipblas-test. This client is used to test hipBLAS.

#### 1.4.2.2 clients/benchmarks

Code for client hipblas-benchmark. This client is used to benchmark hipBLAS functions.

#### 1.4.2.3 clients/include

Code for testing and benchmarking individual hipBLAS functions, and utility code for testing.

#### 1.4.2.4 clients/common

Common code used by both hipblas-benchmark and hipblas-test.

#### 1.4.2.5 clients/samples

Sample code for calling hipBLAS functions.

### 1.4.3 Infrastructure

- CMake is used to build and package hipBLAS. There are CMakeLists.txt files throughout the code.
- Doxygen/Breathe/Sphinx/ReadTheDocs are used to produce documentation. Content for the documentation is from:
  - Doxygen comments in include files in the directory library/include
  - files in the directory docs/source.
- Jenkins is used to automate Continuous Integration testing.
- clang-format is used to format C++ code.

## 1.5 API

### 1.5.1 hipBLAS interface examples

The hipBLAS interface is compatible with rocBLAS and cuBLAS-v2 APIs. Porting a CUDA application which originally calls the cuBLAS API to an application calling hipBLAS API should be relatively straightforward. For example, the hipBLAS SGEMV interface is

### 1.5.2 GEMV API

```
hipblasStatus_t
hipblasSgemv( hipblasHandle_t handle,
              hipblasOperation_t trans,
              int m, int n, const float *alpha,
              const float *A, int lda,
              const float *x, int incx, const float *beta,
              float *y, int incy );
```

### 1.5.3 Batched and strided GEMM API

hipBLAS GEMM can process matrices in batches with regular strides. There are several permutations of these API's, the following is an example that takes everything

```
hipblasStatus_t
hipblasSgemmStridedBatched( hipblasHandle_t handle,
                             hipblasOperation_t transa, hipblasOperation_t transb,
                             int m, int n, int k, const float *alpha,
                             const float *A, int lda, long long bsa,
                             const float *B, int ldb, long long bsb, const float *beta,
                             float *C, int ldc, long long bsc,
                             int batchCount);
```

hipBLAS assumes matrices A and vectors x, y are allocated in GPU memory space filled with data. Users are responsible for copying data from/to the host and device memory.

## 1.6 Guidelines

### 1.6.1 Naming conventions

hipBLAS follows the following naming conventions,

- Big case for matrix, e.g. matrix A, B, C GEMM ( $C = A * B$ )
- Lower case for vector, e.g. vector x, y GEMV ( $y = A * x$ )

## 1.6.2 Notations

hipBLAS function uses the following notations to denote precisions,

- h = half
- bf = 16 bit brian floating point
- s = single
- d = double
- c = single complex
- z = double complex

## 1.7 hipBLAS Types

### 1.7.1 Definitions

#### 1.7.1.1 hipblasHandle\_t

typedef void \***hipblasHandle\_t**

hipblasHandle\_t is a void pointer, to store the library context (either rocBLAS or cuBLAS)

#### 1.7.1.2 hipblasHalf

typedef uint16\_t **hipblasHalf**

To specify the datatype to be signed short.

#### 1.7.1.3 hipblasInt8

typedef int8\_t **hipblasInt8**

To specify the datatype to be signed char.

#### 1.7.1.4 hipblasStride

typedef int64\_t **hipblasStride**

Stride between matrices or vectors in strided\_batched functions.

### 1.7.1.5 hipblasBfloat16

struct **hipblasBfloat16**

Struct to represent a 16 bit Brain floating-point number.

### 1.7.1.6 hipblasComplex

struct **hipblasComplex**

Struct to represent a complex number with single precision real and imaginary parts.

### 1.7.1.7 hipblasDoubleComplex

struct **hipblasDoubleComplex**

Struct to represent a complex number with double precision real and imaginary parts.

## 1.7.2 Enums

Enumeration constants have numbering that is consistent with CBLAS, ACML and most standard C BLAS libraries.

### 1.7.2.1 hipblasStatus\_t

enum **hipblasStatus\_t**

hipblas status codes definition

*Values:*

enumerator **HIPBLAS\_STATUS\_SUCCESS**

Function succeeds

enumerator **HIPBLAS\_STATUS\_NOT\_INITIALIZED**

HIPBLAS library not initialized

enumerator **HIPBLAS\_STATUS\_ALLOC\_FAILED**

resource allocation failed

enumerator **HIPBLAS\_STATUS\_INVALID\_VALUE**

unsupported numerical value was passed to function

enumerator **HIPBLAS\_STATUS\_MAPPING\_ERROR**

access to GPU memory space failed

enumerator **HIPBLAS\_STATUS\_EXECUTION\_FAILED**

GPU program failed to execute

enumerator **HIPBLAS\_STATUS\_INTERNAL\_ERROR**

an internal HIPBLAS operation failed

enumerator **HIPBLAS\_STATUS\_NOT\_SUPPORTED**  
function not implemented

enumerator **HIPBLAS\_STATUS\_ARCH\_MISMATCH**  
architecture mismatch

enumerator **HIPBLAS\_STATUS\_HANDLE\_IS\_NULLPTR**  
hipBLAS handle is null pointer

enumerator **HIPBLAS\_STATUS\_INVALID\_ENUM**  
unsupported enum value was passed to function

enumerator **HIPBLAS\_STATUS\_UNKNOWN**  
back-end returned an unsupported status code

### 1.7.2.2 hipblasOperation\_t

enum **hipblasOperation\_t**  
Used to specify whether the matrix is to be transposed or not.

*Values:*

enumerator **HIPBLAS\_OP\_N**  
Operate with the matrix.

enumerator **HIPBLAS\_OP\_T**  
Operate with the transpose of the matrix.

enumerator **HIPBLAS\_OP\_C**  
Operate with the conjugate transpose of the matrix.

### 1.7.2.3 hipblasPointerMode\_t

enum **hipblasPointerMode\_t**  
Indicates if scalar pointers are on host or device. This is used for scalars alpha and beta and for scalar function return values.

*Values:*

enumerator **HIPBLAS\_POINTER\_MODE\_HOST**  
Scalar values affected by this variable will be located on the host.

enumerator **HIPBLAS\_POINTER\_MODE\_DEVICE**  
Scalar values affected by this variable will be located on the device.

### 1.7.2.4 hipblasFillMode\_t

enum **hipblasFillMode\_t**

Used by the Hermitian, symmetric and triangular matrix routines to specify whether the upper or lower triangle is being referenced.

*Values:*

enumerator **HIPBLAS\_FILL\_MODE\_UPPER**  
Upper triangle

enumerator **HIPBLAS\_FILL\_MODE\_LOWER**  
Lower triangle

enumerator **HIPBLAS\_FILL\_MODE\_FULL**

### 1.7.2.5 hipblasDiagType\_t

enum **hipblasDiagType\_t**

It is used by the triangular matrix routines to specify whether the matrix is unit triangular.

*Values:*

enumerator **HIPBLAS\_DIAG\_NON\_UNIT**  
Non-unit triangular.

enumerator **HIPBLAS\_DIAG\_UNIT**  
Unit triangular.

### 1.7.2.6 hipblasSideMode\_t

enum **hipblasSideMode\_t**

Indicates the side matrix A is located relative to matrix B during multiplication.

*Values:*

enumerator **HIPBLAS\_SIDE\_LEFT**  
Multiply general matrix by symmetric, Hermitian or triangular matrix on the left.

enumerator **HIPBLAS\_SIDE\_RIGHT**  
Multiply general matrix by symmetric, Hermitian or triangular matrix on the right.

enumerator **HIPBLAS\_SIDE\_BOTH**



### 1.7.2.7 hipblasDatatype\_t

enum **hipblasDatatype\_t**

Indicates the precision width of data stored in a blas type.

*Values:*

enumerator **HIPBLAS\_R\_16F**  
16 bit floating point, real

enumerator **HIPBLAS\_R\_32F**  
32 bit floating point, real

enumerator **HIPBLAS\_R\_64F**  
64 bit floating point, real

enumerator **HIPBLAS\_C\_16F**  
16 bit floating point, complex

enumerator **HIPBLAS\_C\_32F**  
32 bit floating point, complex

enumerator **HIPBLAS\_C\_64F**  
64 bit floating point, complex

enumerator **HIPBLAS\_R\_8I**  
8 bit signed integer, real

enumerator **HIPBLAS\_R\_8U**  
8 bit unsigned integer, real

enumerator **HIPBLAS\_R\_32I**  
32 bit signed integer, real

enumerator **HIPBLAS\_R\_32U**  
32 bit unsigned integer, real

enumerator **HIPBLAS\_C\_8I**  
8 bit signed integer, complex

enumerator **HIPBLAS\_C\_8U**  
8 bit unsigned integer, complex

enumerator **HIPBLAS\_C\_32I**  
32 bit signed integer, complex

enumerator **HIPBLAS\_C\_32U**  
32 bit unsigned integer, complex

enumerator **HIPBLAS\_R\_16B**  
16 bit bfloat, real

enumerator **HIPBLAS\_C\_16B**  
16 bit bfloat, complex

### 1.7.2.8 hipblasGemmAlgo\_t

enum **hipblasGemmAlgo\_t**  
Indicates if layer is active with bitmask.

*Values:*

enumerator **HIPBLAS\_GEMM\_DEFAULT**  
enumerator rocblas\_gemm\_algo\_standard

### 1.7.2.9 hipblasAtomicsMode\_t

enum **hipblasAtomicsMode\_t**  
Indicates if atomics operations are allowed. Not allowing atomic operations may generally improve determinism and repeatability of results at a cost of performance.

*Values:*

enumerator **HIPBLAS\_ATOMICS\_NOT\_ALLOWED**  
Algorithms will refrain from atomics where applicable.

enumerator **HIPBLAS\_ATOMICS\_ALLOWED**  
Algorithms will take advantage of atomics where applicable.

## 1.8 hipBLAS Functions

### 1.8.1 Level 1 BLAS

#### List of Level-1 BLAS Functions

- *hipblasIXamax + Batched, StridedBatched*
- *hipblasIXamin + Batched, StridedBatched*
- *hipblasXasum + Batched, StridedBatched*
- *hipblasXaxpy + Batched, StridedBatched*
- *hipblasXcopy + Batched, StridedBatched*
- *hipblasXdot + Batched, StridedBatched*
- *hipblasXnrm2 + Batched, StridedBatched*
- *hipblasXrot + Batched, StridedBatched*
- *hipblasXrotg + Batched, StridedBatched*
- *hipblasXrotm + Batched, StridedBatched*

- *hipblasXrotmg + Batched, StridedBatched*
- *hipblasXscal + Batched, StridedBatched*
- *hipblasXswap + Batched, StridedBatched*

### 1.8.1.1 hipblasIamax + Batched, StridedBatched

*hipblasStatus\_t* **hipblasIsamax**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, int \*result)

*hipblasStatus\_t* **hipblasIdamax**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, int \*result)

*hipblasStatus\_t* **hipblasIcamax**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, int \*result)

*hipblasStatus\_t* **hipblasIzamax**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, int \*result)

BLAS Level 1 API.

amax finds the first index of the element of maximum magnitude of a vector x.

- Supported precisions in rocBLAS : s,d,c,z.
- Supported precisions in cuBLAS : s,d,c,z.

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of y.
- **result** – [inout] device pointer or host pointer to store the amax index. return is 0.0 if n, incx<=0.

*hipblasStatus\_t* **hipblasIsamaxBatched**(*hipblasHandle\_t* handle, int n, const float \*const x[], int incx, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIdamaxBatched**(*hipblasHandle\_t* handle, int n, const double \*const x[], int incx, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIcamaxBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*const x[], int incx, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIzamaxBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*const x[], int incx, int batchCount, int \*result)

BLAS Level 1 API.

amaxBatched finds the first index of the element of maximum magnitude of each vector x\_i in a batch, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d,c,z.
- Supported precisions in cuBLAS : No support.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each vector  $x_i$
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ . incx must be  $> 0$ .
- **batchCount** – [in] [int] number of instances in the batch, must be  $> 0$ .
- **result** – [out] device or host array of pointers of batchCount size for results. return is 0 if  $n, incx \leq 0$ .

*hipblasStatus\_t* **hipblasIsamaxStridedBatched**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, *hipblasStride* stridex, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIdamaxStridedBatched**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, *hipblasStride* stridex, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIcamaxStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIzamaxStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount, int \*result)

BLAS Level 1 API.

amaxStridedBatched finds the first index of the element of maximum magnitude of each vector  $x_i$  in a batch, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each vector  $x_i$
- **x** – [in] device pointer to the first vector  $x_1$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ . incx must be  $> 0$ .
- **stridex** – [in] [hipblasStride] specifies the pointer increment between one  $x_i$  and the next  $x_{(i+1)}$ .
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [out] device or host pointer for storing contiguous batchCount results. return is 0 if  $n \leq 0, incx \leq 0$ .

### 1.8.1.2 hipblasIamin + Batched, StridedBatched

*hipblasStatus\_t* **hipblasIsamin**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, int \*result)

*hipblasStatus\_t* **hipblasIdamin**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, int \*result)

*hipblasStatus\_t* **hipblasIcamin**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, int \*result)

*hipblasStatus\_t* **hipblasIzamin**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, int \*result)

BLAS Level 1 API.

amin finds the first index of the element of minimum magnitude of a vector x.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of y.
- **result** – [inout] device pointer or host pointer to store the amin index. return is 0.0 if n, incx<=0.

*hipblasStatus\_t* **hipblasIsaminBatched**(*hipblasHandle\_t* handle, int n, const float \*const x[], int incx, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIdaminBatched**(*hipblasHandle\_t* handle, int n, const double \*const x[], int incx, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIcaminBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*const x[], int incx, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIzaminBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*const x[], int incx, int batchCount, int \*result)

BLAS Level 1 API.

aminBatched finds the first index of the element of minimum magnitude of each vector x<sub>i</sub> in a batch, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.

- **n** – [in] [int] number of elements in each vector  $x_i$
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ . incx must be  $> 0$ .
- **batchCount** – [in] [int] number of instances in the batch, must be  $> 0$ .
- **result** – [out] device or host pointers to array of batchCount size for results. return is 0 if  $n, incx \leq 0$ .

*hipblasStatus\_t* **hipblasIsaminStridedBatched**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, *hipblasStride* stridex, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIdaminStridedBatched**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, *hipblasStride* stridex, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIcaminStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount, int \*result)

*hipblasStatus\_t* **hipblasIzaminStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount, int \*result)

BLAS Level 1 API.

aminStridedBatched finds the first index of the element of minimum magnitude of each vector  $x_i$  in a batch, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each vector  $x_i$
- **x** – [in] device pointer to the first vector  $x_1$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ . incx must be  $> 0$ .
- **stridex** – [in] [hipblasStride] specifies the pointer increment between one  $x_i$  and the next  $x_{(i+1)}$
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [out] device or host pointer to array for storing contiguous batchCount results. return is 0 if  $n \leq 0, incx \leq 0$ .

### 1.8.1.3 hipblasXasum + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSasum**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, float \*result)

*hipblasStatus\_t* **hipblasDasum**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, double \*result)

*hipblasStatus\_t* **hipblasScasum**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, float \*result)

*hipblasStatus\_t* **hipblasDzasum**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, double \*result)

BLAS Level 1 API.

asum computes the sum of the magnitudes of elements of a real vector x, or the sum of magnitudes of the real and imaginary parts of elements if x is a complex vector.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x and y.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x. incx must be > 0.
- **result** – [inout] device pointer or host pointer to store the asum product. return is 0.0 if n <= 0.

*hipblasStatus\_t* **hipblasSasumBatched**(*hipblasHandle\_t* handle, int n, const float \*const x[], int incx, int batchCount, float \*result)

*hipblasStatus\_t* **hipblasDasumBatched**(*hipblasHandle\_t* handle, int n, const double \*const x[], int incx, int batchCount, double \*result)

*hipblasStatus\_t* **hipblasScasumBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*const x[], int incx, int batchCount, float \*result)

*hipblasStatus\_t* **hipblasDzasumBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*const x[], int incx, int batchCount, double \*result)

BLAS Level 1 API.

asumBatched computes the sum of the magnitudes of the elements in a batch of real vectors x\_i, or the sum of magnitudes of the real and imaginary parts of elements if x\_i is a complex vector, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each vector  $x_i$
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ . incx must be  $> 0$ .
- **batchCount** – [in] [int] number of instances in the batch.
- **result** – [out] device array or host array of batchCount size for results. return is 0.0 if n, incx $\leq$ 0.

*hipblasStatus\_t* **hipblasSasumStridedBatched**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, *hipblasStride* stridex, int batchCount, float \*result)

*hipblasStatus\_t* **hipblasDasumStridedBatched**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, *hipblasStride* stridex, int batchCount, double \*result)

*hipblasStatus\_t* **hipblasScasumStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount, float \*result)

*hipblasStatus\_t* **hipblasDzasumStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount, double \*result)

BLAS Level 1 API.

asumStridedBatched computes the sum of the magnitudes of elements of a real vectors  $x_i$ , or the sum of magnitudes of the real and imaginary parts of elements if  $x_i$  is a complex vector, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each vector  $x_i$
- **x** – [in] device pointer to the first vector  $x_1$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ . incx must be  $> 0$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ). There are no restrictions placed on stride\_x, however the user should take care to ensure that stride\_x is of appropriate size, for a typical case this means stride\_x  $\geq n * \text{incx}$ .
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [out] device pointer or host pointer to array for storing contiguous batchCount results. return is 0.0 if n, incx $\leq$ 0.



### 1.8.1.4 hipblasXaxpy + Batched, StridedBatched

*hipblasStatus\_t* **hipblasHaxpy**(*hipblasHandle\_t* handle, int n, const *hipblasHalf* \*alpha, const *hipblasHalf* \*x, int incx, *hipblasHalf* \*y, int incy)

*hipblasStatus\_t* **hipblasSaxpy**(*hipblasHandle\_t* handle, int n, const float \*alpha, const float \*x, int incx, float \*y, int incy)

*hipblasStatus\_t* **hipblasDaxpy**(*hipblasHandle\_t* handle, int n, const double \*alpha, const double \*x, int incx, double \*y, int incy)

*hipblasStatus\_t* **hipblasCaxpy**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasComplex* \*y, int incy)

*hipblasStatus\_t* **hipblasZaxpy**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*y, int incy)

BLAS Level 1 API.

axpy computes constant alpha multiplied by vector x, plus vector y

$$y := \alpha * x + y$$

- Supported precisions in rocBLAS : h,s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x and y.
- **alpha** – [in] device pointer or host pointer to specify the scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [out] device pointer storing vector y.
- **incy** – [inout] [int] specifies the increment for the elements of y.

*hipblasStatus\_t* **hipblasHaxpyBatched**(*hipblasHandle\_t* handle, int n, const *hipblasHalf* \*alpha, const *hipblasHalf* \*const x[], int incx, *hipblasHalf* \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasSaxpyBatched**(*hipblasHandle\_t* handle, int n, const float \*alpha, const float \*const x[], int incx, float \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasDaxpyBatched**(*hipblasHandle\_t* handle, int n, const double \*alpha, const double \*const x[], int incx, double \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasCaxpyBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const x[], int incx, *hipblasComplex* \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasZaxpyBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const x[], int incx, *hipblasDoubleComplex* \*const y[], int incy, int batchCount)

BLAS Level 1 API.

axpyBatched compute  $y := \alpha * x + y$  over a set of batched vectors.

- Supported precisions in rocBLAS : h,s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x and y.
- **alpha** – [in] specifies the scalar alpha.
- **x** – [in] pointer storing vector x on the GPU.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [out] pointer storing vector y on the GPU.
- **incy** – [inout] [int] specifies the increment for the elements of y.
- **batchCount** – [in] [int] number of instances in the batch

*hipblasStatus\_t* **hipblasHaxpyStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasHalf* \*alpha, const *hipblasHalf* \*x, int incx, *hipblasStride* stridex, *hipblasHalf* \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasSaxpyStridedBatched**(*hipblasHandle\_t* handle, int n, const float \*alpha, const float \*x, int incx, *hipblasStride* stridex, float \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasDaxpyStridedBatched**(*hipblasHandle\_t* handle, int n, const double \*alpha, const double \*x, int incx, *hipblasStride* stridex, double \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasCaxpyStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, *hipblasComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasZaxpyStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

BLAS Level 1 API.

axpyStridedBatched compute  $y := \alpha * x + y$  over a set of strided batched vectors.

- Supported precisions in rocBLAS : h,s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int]
- **alpha** – [in] specifies the scalar alpha.
- **x** – [in] pointer storing vector x on the GPU.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **stridex** – [in] [hipblasStride] specifies the increment between vectors of x.
- **y** – [out] pointer storing vector y on the GPU.
- **incy** – [inout] [int] specifies the increment for the elements of y.
- **stridey** – [in] [hipblasStride] specifies the increment between vectors of y.
- **batchCount** – [in] [int] number of instances in the batch

#### 1.8.1.5 hipblasXcopy + Batched, StridedBatched

*hipblasStatus\_t* **hipblasScopy**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, float \*y, int incy)

*hipblasStatus\_t* **hipblasDcopy**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, double \*y, int incy)

*hipblasStatus\_t* **hipblasCcopy**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, *hipblasComplex* \*y, int incy)

*hipblasStatus\_t* **hipblasZcopy**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*y, int incy)

BLAS Level 1 API.

copy copies each element  $x[i]$  into  $y[i]$ , for  $i = 1, \dots, n$

$y := x,$
-----------

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x to be copied to y.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.

- **y** – [out] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.

*hipblasStatus\_t* **hipblasScopyBatched**(*hipblasHandle\_t* handle, int n, const float \*const x[], int incx, float \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasDcopyBatched**(*hipblasHandle\_t* handle, int n, const double \*const x[], int incx, double \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasCcopyBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*const x[], int incx, *hipblasComplex* \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasZcopyBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*const x[], int incx, *hipblasDoubleComplex* \*const y[], int incy, int batchCount)

BLAS Level 1 API.

copyBatched copies each element  $x_i[j]$  into  $y_i[j]$ , for  $j = 1, \dots, n$ ;  $i = 1, \dots, \text{batchCount}$

$y_i := x_i,$

where  $(x_i, y_i)$  is the  $i$ -th instance of the batch.  $x_i$  and  $y_i$  are vectors.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x_i$  to be copied to  $y_i$ .
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each vector  $x_i$ .
- **y** – [out] device array of device pointers storing each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of each vector  $y_i$ .
- **batchCount** – [in] [int] number of instances in the batch

*hipblasStatus\_t* **hipblasScopyStridedBatched**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, *hipblasStride* stridex, float \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasDcopyStridedBatched**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, *hipblasStride* stridex, double \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasCcopyStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, *hipblasComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasZcopyStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

BLAS Level 1 API.

copyStridedBatched copies each element  $x\_i[j]$  into  $y\_i[j]$ , for  $j = 1, \dots, n$ ;  $i = 1, \dots, \text{batchCount}$

$y\_i := x\_i,$

where  $(x\_i, y\_i)$  is the  $i$ -th instance of the batch.  $x\_i$  and  $y\_i$  are vectors.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x\_i$  to be copied to  $y\_i$ .
- **x** – [in] device pointer to the first vector ( $x\_1$ ) in the batch.
- **incx** – [in] [int] specifies the increments for the elements of vectors  $x\_i$ .
- **stridex** – [in] [*hipblasStride*] stride from the start of one vector ( $x\_i$ ) and the next one ( $x_{i+1}$ ). There are no restrictions placed on stride\_x, however the user should take care to ensure that stride\_x is of appropriate size, for a typical case this means stride\_x  $\geq n * \text{incx}$ .
- **y** – [out] device pointer to the first vector ( $y\_1$ ) in the batch.
- **incy** – [in] [int] specifies the increment for the elements of vectors  $y\_i$ .
- **stridey** – [in] [*hipblasStride*] stride from the start of one vector ( $y\_i$ ) and the next one ( $y_{i+1}$ ). There are no restrictions placed on stride\_y, however the user should take care to ensure that stride\_y is of appropriate size, for a typical case this means stride\_y  $\geq n * \text{incy}$ . stridey should be non zero.
- **batchCount** – [in] [int] number of instances in the batch

#### 1.8.1.6 hipblasXdot + Batched, StridedBatched

*hipblasStatus\_t* **hipblasHdot**(*hipblasHandle\_t* handle, int n, const *hipblasHalf* \*x, int incx, const *hipblasHalf* \*y, int incy, *hipblasHalf* \*result)

*hipblasStatus\_t* **hipblasBfdot**(*hipblasHandle\_t* handle, int n, const *hipblasBfloat16* \*x, int incx, const *hipblasBfloat16* \*y, int incy, *hipblasBfloat16* \*result)

*hipblasStatus\_t* **hipblasSdot**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, const float \*y, int incy, float \*result)

*hipblasStatus\_t* **hipblasDdot**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, const double \*y, int incy, double \*result)

*hipblasStatus\_t* **hipblasCdotc**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*y, int incy, *hipblasComplex* \*result)

*hipblasStatus\_t* **hipblasCdotu**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*y, int incy, *hipblasComplex* \*result)

*hipblasStatus\_t* **hipblasZdotc**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*y, int incy, *hipblasDoubleComplex* \*result)

*hipblasStatus\_t* **hipblasZdotu**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*y, int incy, *hipblasDoubleComplex* \*result)

BLAS Level 1 API.

dot(u) performs the dot product of vectors x and y

```
result = x * y;
```

dotc performs the dot product of the conjugate of complex vector x and complex vector y

```
result = conjugate (x) * y;
```

- Supported precisions in rocBLAS : h,bf,s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x and y.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of y.
- **y** – [in] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **result** – [inout] device pointer or host pointer to store the dot product. return is 0.0 if n <= 0.

*hipblasStatus\_t* **hipblasHdotBatched**(*hipblasHandle\_t* handle, int n, const *hipblasHalf* \*const x[], int incx, const *hipblasHalf* \*const y[], int incy, int batchSize, *hipblasHalf* \*result)

*hipblasStatus\_t* **hipblasBfdotBatched**(*hipblasHandle\_t* handle, int n, const *hipblasBfloat16* \*const x[], int incx, const *hipblasBfloat16* \*const y[], int incy, int batchSize, *hipblasBfloat16* \*result)

*hipblasStatus\_t* **hipblasSdotBatched**(*hipblasHandle\_t* handle, int n, const float \*const x[], int incx, const float \*const y[], int incy, int batchSize, float \*result)

*hipblasStatus\_t* **hipblasDdotBatched**(*hipblasHandle\_t* handle, int n, const double \*const x[], int incx, const double \*const y[], int incy, int batchCount, double \*result)

*hipblasStatus\_t* **hipblasCdotcBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*const y[], int incy, int batchCount, *hipblasComplex* \*result)

*hipblasStatus\_t* **hipblasCdotuBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*const y[], int incy, int batchCount, *hipblasComplex* \*result)

*hipblasStatus\_t* **hipblasZdotcBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*const y[], int incy, int batchCount, *hipblasDoubleComplex* \*result)

*hipblasStatus\_t* **hipblasZdotuBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*const y[], int incy, int batchCount, *hipblasDoubleComplex* \*result)

BLAS Level 1 API.

dotBatched(u) performs a batch of dot products of vectors x and y

```
result_i = x_i * y_i;
```

dotcBatched performs a batch of dot products of the conjugate of complex vector x and complex vector y

```
result_i = conjugate (x_i) * y_i;
```

where (x\_i, y\_i) is the i-th instance of the batch. x\_i and y\_i are vectors, for i = 1, ..., batchCount

- Supported precisions in rocBLAS : h,bf,s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each x\_i and y\_i.
- **x** – [in] device array of device pointers storing each vector x\_i.
- **incx** – [in] [int] specifies the increment for the elements of each x\_i.
- **y** – [in] device array of device pointers storing each vector y\_i.
- **incy** – [in] [int] specifies the increment for the elements of each y\_i.
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [inout] device array or host array of batchCount size to store the dot products of each batch. return 0.0 for each element if n <= 0.

*hipblasStatus\_t* **hipblasHdotStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasHalf* \*x, int incx, *hipblasStride* stridex, const *hipblasHalf* \*y, int incy, *hipblasStride* stridey, int batchCount, *hipblasHalf* \*result)

*hipblasStatus\_t* **hipblasBfdotStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasBfloat16* \*x, int incx, *hipblasStride* stridex, const *hipblasBfloat16* \*y, int incy, *hipblasStride* stridey, int batchCount, *hipblasBfloat16* \*result)

*hipblasStatus\_t* **hipblasSdotStridedBatched**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, *hipblasStride* stridex, const float \*y, int incy, *hipblasStride* stridey, int batchCount, float \*result)

*hipblasStatus\_t* **hipblasDdotStridedBatched**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, *hipblasStride* stridex, const double \*y, int incy, *hipblasStride* stridey, int batchCount, double \*result)

*hipblasStatus\_t* **hipblasCdotcStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*y, int incy, *hipblasStride* stridey, int batchCount, *hipblasComplex* \*result)

*hipblasStatus\_t* **hipblasCdotuStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*y, int incy, *hipblasStride* stridey, int batchCount, *hipblasComplex* \*result)

*hipblasStatus\_t* **hipblasZdotcStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, int batchCount, *hipblasDoubleComplex* \*result)

*hipblasStatus\_t* **hipblasZdotuStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, int batchCount, *hipblasDoubleComplex* \*result)

BLAS Level 1 API.

dotStridedBatched(u) performs a batch of dot products of vectors x and y

```
result_i = x_i * y_i;
```

dotcStridedBatched performs a batch of dot products of the conjugate of complex vector x and complex vector y

```
result_i = conjugate (x_i) * y_i;
```

where (x\_i, y\_i) is the i-th instance of the batch. x\_i and y\_i are vectors, for i = 1, ..., batchCount

- Supported precisions in rocBLAS : h,bf,s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters



- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x_i$  and  $y_i$ .
- **x** – [in] device pointer to the first vector ( $x_1$ ) in the batch.
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ )
- **y** – [in] device pointer to the first vector ( $y_1$ ) in the batch.
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **stridey** – [in] [hipblasStride] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [inout] device array or host array of batchCount size to store the dot products of each batch. return 0.0 for each element if  $n \leq 0$ .

### 1.8.1.7 hipblasXnrm2 + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSnrm2**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, float \*result)

*hipblasStatus\_t* **hipblasDnrm2**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, double \*result)

*hipblasStatus\_t* **hipblasScnrm2**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, float \*result)

*hipblasStatus\_t* **hipblasDznrm2**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, double \*result)

BLAS Level 1 API.

nrm2 computes the euclidean norm of a real or complex vector

```
result := sqrt( x'*x ) for real vectors
result := sqrt( x**H*x ) for complex vectors
```

- Supported precisions in rocBLAS : s,d,c,z,sc,dz
- Supported precisions in cuBLAS : s,d,sc,dz

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of y.
- **result** – [inout] device pointer or host pointer to store the nrm2 product. return is 0.0 if n, incx<=0.

*hipblasStatus\_t* **hipblasSnrm2Batched**(*hipblasHandle\_t* handle, int n, const float \*const x[], int incx, int batchCount, float \*result)

*hipblasStatus\_t* **hipblasDnrm2Batched**(*hipblasHandle\_t* handle, int n, const double \*const x[], int incx, int batchCount, double \*result)

*hipblasStatus\_t* **hipblasScnrm2Batched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*const x[], int incx, int batchCount, float \*result)

*hipblasStatus\_t* **hipblasDznrm2Batched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*const x[], int incx, int batchCount, double \*result)

BLAS Level 1 API.

nrm2Batched computes the euclidean norm over a batch of real or complex vectors

```
result := sqrt( x_i'*x_i ) for real vectors x, for i = 1, ..., batchCount
result := sqrt( x_i**H*x_i ) for complex vectors x, for i = 1, ..., batchCount
```

- Supported precisions in rocBLAS : s,d,c,z,sc,dz
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each  $x_i$ .
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ . incx must be > 0.
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [out] device pointer or host pointer to array of batchCount size for nrm2 results. return is 0.0 for each element if  $n \leq 0$ ,  $incx \leq 0$ .

*hipblasStatus\_t* **hipblasSnrm2StridedBatched**(*hipblasHandle\_t* handle, int n, const float \*x, int incx, *hipblasStride* stridex, int batchCount, float \*result)

*hipblasStatus\_t* **hipblasDnrm2StridedBatched**(*hipblasHandle\_t* handle, int n, const double \*x, int incx, *hipblasStride* stridex, int batchCount, double \*result)

*hipblasStatus\_t* **hipblasScnrm2StridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount, float \*result)

*hipblasStatus\_t* **hipblasDznrm2StridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount, double \*result)

BLAS Level 1 API.

nrm2StridedBatched computes the euclidean norm over a batch of real or complex vectors

```
:= sqrt( x_i'*x_i ) for real vectors x, for i = 1, ..., batchCount
:= sqrt( x_i**H*x_i ) for complex vectors, for i = 1, ..., batchCount
```

- Supported precisions in rocBLAS : s,d,c,z,sc,dz
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each  $x_i$ .
- **x** – [in] device pointer to the first vector  $x_1$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ . incx must be  $> 0$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ). There are no restrictions placed on stride\_x, however the user should take care to ensure that stride\_x is of appropriate size, for a typical case this means  $\text{stride}_x \geq n * \text{incx}$ .
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [out] device pointer or host pointer to array for storing contiguous batchCount results. return is 0.0 for each element if  $n \leq 0$ ,  $\text{incx} \leq 0$ .

#### 1.8.1.8 hipblasXrot + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSrot**(*hipblasHandle\_t* handle, int n, float \*x, int incx, float \*y, int incy, const float \*c, const float \*s)

*hipblasStatus\_t* **hipblasDrot**(*hipblasHandle\_t* handle, int n, double \*x, int incx, double \*y, int incy, const double \*c, const double \*s)

*hipblasStatus\_t* **hipblasCrot**(*hipblasHandle\_t* handle, int n, *hipblasComplex* \*x, int incx, *hipblasComplex* \*y, int incy, const float \*c, const *hipblasComplex* \*s)

*hipblasStatus\_t* **hipblasCsrot**(*hipblasHandle\_t* handle, int n, *hipblasComplex* \*x, int incx, *hipblasComplex* \*y, int incy, const float \*c, const float \*s)

*hipblasStatus\_t* **hipblasZrot**(*hipblasHandle\_t* handle, int n, *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*y, int incy, const double \*c, const *hipblasDoubleComplex* \*s)

*hipblasStatus\_t* **hipblasZdrot**(*hipblasHandle\_t* handle, int n, *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*y, int incy, const double \*c, const double \*s)

BLAS Level 1 API.

rot applies the Givens rotation matrix defined by  $c=\cos(\alpha)$  and  $s=\sin(\alpha)$  to vectors x and y. Scalars c and s may be stored in either host or device memory, location is specified by calling hipblasSetPointerMode.

- Supported precisions in rocBLAS : s,d,c,z,sc,dz
- Supported precisions in cuBLAS : s,d,c,z,cs,zd

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in the x and y vectors.
- **x** – [inout] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment between elements of x.
- **y** – [inout] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment between elements of y.
- **c** – [in] device pointer or host pointer storing scalar cosine component of the rotation matrix.
- **s** – [in] device pointer or host pointer storing scalar sine component of the rotation matrix.

*hipblasStatus\_t* **hipblasSrotBatched**(*hipblasHandle\_t* handle, int n, float \*const x[], int incx, float \*const y[], int incy, const float \*c, const float \*s, int batchCount)

*hipblasStatus\_t* **hipblasDrotBatched**(*hipblasHandle\_t* handle, int n, double \*const x[], int incx, double \*const y[], int incy, const double \*c, const double \*s, int batchCount)

*hipblasStatus\_t* **hipblasCrotBatched**(*hipblasHandle\_t* handle, int n, *hipblasComplex* \*const x[], int incx, *hipblasComplex* \*const y[], int incy, const float \*c, const *hipblasComplex* \*s, int batchCount)

*hipblasStatus\_t* **hipblasCsrotBatched**(*hipblasHandle\_t* handle, int n, *hipblasComplex* \*const x[], int incx, *hipblasComplex* \*const y[], int incy, const float \*c, const float \*s, int batchCount)

*hipblasStatus\_t* **hipblasZrotBatched**(*hipblasHandle\_t* handle, int n, *hipblasDoubleComplex* \*const x[], int incx, *hipblasDoubleComplex* \*const y[], int incy, const double \*c, const *hipblasDoubleComplex* \*s, int batchCount)

*hipblasStatus\_t* **hipblasZdrotBatched**(*hipblasHandle\_t* handle, int n, *hipblasDoubleComplex* \*const x[], int incx, *hipblasDoubleComplex* \*const y[], int incy, const double \*c, const double \*s, int batchCount)

BLAS Level 1 API.

rotBatched applies the Givens rotation matrix defined by  $c=\cos(\alpha)$  and  $s=\sin(\alpha)$  to batched vectors  $x_i$  and  $y_i$ , for  $i = 1, \dots, \text{batchCount}$ . Scalars  $c$  and  $s$  may be stored in either host or device memory, location is specified by calling `hipblasSetPointerMode`.

- Supported precisions in rocBLAS : s,d,sc,dz
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each  $x_i$  and  $y_i$  vectors.
- **x** – [inout] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment between elements of each  $x_i$ .

- **y** – [inout] device array of device pointers storing each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment between elements of each  $y_i$ .
- **c** – [in] device pointer or host pointer to scalar cosine component of the rotation matrix.
- **s** – [in] device pointer or host pointer to scalar sine component of the rotation matrix.
- **batchCount** – [in] [int] the number of x and y arrays, i.e. the number of batches.

*hipblasStatus\_t* **hipblasSrotStridedBatched**(*hipblasHandle\_t* handle, int n, float \*x, int incx, *hipblasStride* stridex, float \*y, int incy, *hipblasStride* stridey, const float \*c, const float \*s, int batchCount)

*hipblasStatus\_t* **hipblasDrotStridedBatched**(*hipblasHandle\_t* handle, int n, double \*x, int incx, *hipblasStride* stridex, double \*y, int incy, *hipblasStride* stridey, const double \*c, const double \*s, int batchCount)

*hipblasStatus\_t* **hipblasCrotStridedBatched**(*hipblasHandle\_t* handle, int n, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, *hipblasComplex* \*y, int incy, *hipblasStride* stridey, const float \*c, const *hipblasComplex* \*s, int batchCount)

*hipblasStatus\_t* **hipblasCsrotStridedBatched**(*hipblasHandle\_t* handle, int n, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, *hipblasComplex* \*y, int incy, *hipblasStride* stridey, const float \*c, const float \*s, int batchCount)

*hipblasStatus\_t* **hipblasZrotStridedBatched**(*hipblasHandle\_t* handle, int n, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, const double \*c, const *hipblasDoubleComplex* \*s, int batchCount)

*hipblasStatus\_t* **hipblasZdrotStridedBatched**(*hipblasHandle\_t* handle, int n, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, const double \*c, const double \*s, int batchCount)

BLAS Level 1 API.

rotStridedBatched applies the Givens rotation matrix defined by  $c=\cos(\alpha)$  and  $s=\sin(\alpha)$  to strided batched vectors  $x_i$  and  $y_i$ , for  $i = 1, \dots, \text{batchCount}$ . Scalars  $c$  and  $s$  may be stored in either host or device memory, location is specified by calling `hipblasSetPointerMode`.

- Supported precisions in rocBLAS : s,d,sc,dz
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each  $x_i$  and  $y_i$  vectors.
- **x** – [inout] device pointer to the first vector  $x_1$ .
- **incx** – [in] [int] specifies the increment between elements of each  $x_i$ .

- **stridex** – [in] [hipblasStride] specifies the increment from the beginning of  $x_i$  to the beginning of  $x_{i+1}$
- **y** – [inout] device pointer to the first vector  $y_1$ .
- **incy** – [in] [int] specifies the increment between elements of each  $y_i$ .
- **stridey** – [in] [hipblasStride] specifies the increment from the beginning of  $y_i$  to the beginning of  $y_{i+1}$
- **c** – [in] device pointer or host pointer to scalar cosine component of the rotation matrix.
- **s** – [in] device pointer or host pointer to scalar sine component of the rotation matrix.
- **batchCount** – [in] [int] the number of x and y arrays, i.e. the number of batches.

### 1.8.1.9 hipblasXrotg + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSrotg**(*hipblasHandle\_t* handle, float \*a, float \*b, float \*c, float \*s)

*hipblasStatus\_t* **hipblasDrotg**(*hipblasHandle\_t* handle, double \*a, double \*b, double \*c, double \*s)

*hipblasStatus\_t* **hipblasCrotg**(*hipblasHandle\_t* handle, *hipblasComplex* \*a, *hipblasComplex* \*b, float \*c, *hipblasComplex* \*s)

*hipblasStatus\_t* **hipblasZrotg**(*hipblasHandle\_t* handle, *hipblasDoubleComplex* \*a, *hipblasDoubleComplex* \*b, double \*c, *hipblasDoubleComplex* \*s)

BLAS Level 1 API.

rotg creates the Givens rotation matrix for the vector (a b). Scalars c and s and arrays a and b may be stored in either host or device memory, location is specified by calling `hipblasSetPointerMode`. If the pointer mode is set to `HIPBLAS_POINTER_MODE_HOST`, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to `HIPBLAS_POINTER_MODE_DEVICE`, this function returns immediately and synchronization is required to read the results.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **a** – [inout] device pointer or host pointer to input vector element, overwritten with r.
- **b** – [inout] device pointer or host pointer to input vector element, overwritten with z.
- **c** – [inout] device pointer or host pointer to cosine element of Givens rotation.
- **s** – [inout] device pointer or host pointer sine element of Givens rotation.

*hipblasStatus\_t* **hipblasSrotgBatched**(*hipblasHandle\_t* handle, float \*const a[], float \*const b[], float \*const c[], float \*const s[], int batchCount)

*hipblasStatus\_t* **hipblasDrotgBatched**(*hipblasHandle\_t* handle, double \*const a[], double \*const b[], double \*const c[], double \*const s[], int batchCount)

*hipblasStatus\_t* **hipblasCrotgBatched**(*hipblasHandle\_t* handle, *hipblasComplex* \*const a[], *hipblasComplex* \*const b[], float \*const c[], *hipblasComplex* \*const s[], int batchCount)

*hipblasStatus\_t* **hipblasZrotgBatched**(*hipblasHandle\_t* handle, *hipblasDoubleComplex* \*const a[], *hipblasDoubleComplex* \*const b[], double \*const c[], *hipblasDoubleComplex* \*const s[], int batchCount)

BLAS Level 1 API.

rotgBatched creates the Givens rotation matrix for the batched vectors (a\_i b\_i), for i = 1, ..., batchCount. a, b, c, and s may be stored in either host or device memory, location is specified by calling hipblasSetPointerMode. If the pointer mode is set to HIPBLAS\_POINTER\_MODE\_HOST, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to HIPBLAS\_POINTER\_MODE\_DEVICE, this function returns immediately and synchronization is required to read the results.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **a** – [inout] device array of device pointers storing each single input vector element a\_i, overwritten with r\_i.
- **b** – [inout] device array of device pointers storing each single input vector element b\_i, overwritten with z\_i.
- **c** – [inout] device array of device pointers storing each cosine element of Givens rotation for the batch.
- **s** – [inout] device array of device pointers storing each sine element of Givens rotation for the batch.
- **batchCount** – [in] [int] number of batches (length of arrays a, b, c, and s).

*hipblasStatus\_t* **hipblasSrotgStridedBatched**(*hipblasHandle\_t* handle, float \*a, *hipblasStride* stridea, float \*b, *hipblasStride* strideb, float \*c, *hipblasStride* stridec, float \*s, *hipblasStride* strides, int batchCount)

*hipblasStatus\_t* **hipblasDrotgStridedBatched**(*hipblasHandle\_t* handle, double \*a, *hipblasStride* stridea, double \*b, *hipblasStride* strideb, double \*c, *hipblasStride* stridec, double \*s, *hipblasStride* strides, int batchCount)

*hipblasStatus\_t* **hipblasCrotgStridedBatched**(*hipblasHandle\_t* handle, *hipblasComplex* \*a, *hipblasStride* stridea, *hipblasComplex* \*b, *hipblasStride* strideb, float \*c, *hipblasStride* stridec, *hipblasComplex* \*s, *hipblasStride* strides, int batchCount)

*hipblasStatus\_t* **hipblasZrotgStridedBatched**(*hipblasHandle\_t* handle, *hipblasDoubleComplex* \*a, *hipblasStride* stridea, *hipblasDoubleComplex* \*b, *hipblasStride* strideb, double \*c, *hipblasStride* stridec, *hipblasDoubleComplex* \*s, *hipblasStride* strides, int batchCount)

BLAS Level 1 API.

rotgStridedBatched creates the Givens rotation matrix for the strided batched vectors ( $a_i$   $b_i$ ), for  $i = 1, \dots, \text{batchCount}$ .  $a$ ,  $b$ ,  $c$ , and  $s$  may be stored in either host or device memory, location is specified by calling `hipblasSetPointerMode`. If the pointer mode is set to `HIPBLAS_POINTER_MODE_HOST`, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to `HIPBLAS_POINTER_MODE_DEVICE`, this function returns immediately and synchronization is required to read the results.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **a** – [inout] device strided\_batched pointer or host strided\_batched pointer to first single input vector element  $a_1$ , overwritten with  $r$ .
- **stridea** – [in] [hipblasStride] distance between elements of  $a$  in batch (distance between  $a_i$  and  $a_{(i+1)}$ )
- **b** – [inout] device strided\_batched pointer or host strided\_batched pointer to first single input vector element  $b_1$ , overwritten with  $z$ .
- **strideb** – [in] [hipblasStride] distance between elements of  $b$  in batch (distance between  $b_i$  and  $b_{(i+1)}$ )
- **c** – [inout] device strided\_batched pointer or host strided\_batched pointer to first cosine element of Givens rotations  $c_1$ .
- **stridec** – [in] [hipblasStride] distance between elements of  $c$  in batch (distance between  $c_i$  and  $c_{(i+1)}$ )
- **s** – [inout] device strided\_batched pointer or host strided\_batched pointer to sine element of Givens rotations  $s_1$ .
- **strides** – [in] [hipblasStride] distance between elements of  $s$  in batch (distance between  $s_i$  and  $s_{(i+1)}$ )
- **batchCount** – [in] [int] number of batches (length of arrays  $a$ ,  $b$ ,  $c$ , and  $s$ ).

#### 1.8.1.10 hipblasXrotm + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSrotm**(*hipblasHandle\_t* handle, int n, float \*x, int incx, float \*y, int incy, const float \*param)

*hipblasStatus\_t* **hipblasDrotm**(*hipblasHandle\_t* handle, int n, double \*x, int incx, double \*y, int incy, const double \*param)

BLAS Level 1 API.

rotm applies the modified Givens rotation matrix defined by param to vectors  $x$  and  $y$ .

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : s,d

#### Parameters



- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in the x and y vectors.
- **x** – [inout] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment between elements of x.
- **y** – [inout] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment between elements of y.
- **param** – [in] device vector or host vector of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 =>  $H = \begin{pmatrix} H11 & H12 & H21 & H22 \end{pmatrix}$  flag = 0 =>  $H = \begin{pmatrix} 1.0 & H12 & H21 & 1.0 \end{pmatrix}$  flag = 1 =>  $H = \begin{pmatrix} H11 & 1.0 & -1.0 & H22 \end{pmatrix}$  flag = -2 =>  $H = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$  param may be stored in either host or device memory, location is specified by calling hipblasSetPointerMode.

*hipblasStatus\_t* **hipblasSrotmBatched**(*hipblasHandle\_t* handle, int n, float \*const x[], int incx, float \*const y[], int incy, const float \*const param[], int batchCount)

*hipblasStatus\_t* **hipblasDrotmBatched**(*hipblasHandle\_t* handle, int n, double \*const x[], int incx, double \*const y[], int incy, const double \*const param[], int batchCount)

BLAS Level 1 API.

rotmBatched applies the modified Givens rotation matrix defined by param\_i to batched vectors x\_i and y\_i, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in the x and y vectors.
- **x** – [inout] device array of device pointers storing each vector x\_i.
- **incx** – [in] [int] specifies the increment between elements of each x\_i.
- **y** – [inout] device array of device pointers storing each vector y\_i.
- **incy** – [in] [int] specifies the increment between elements of each y\_i.
- **param** – [in] device array of device vectors of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 =>  $H = \begin{pmatrix} H11 & H12 & H21 & H22 \end{pmatrix}$  flag = 0 =>  $H = \begin{pmatrix} 1.0 & H12 & H21 & 1.0 \end{pmatrix}$  flag = 1 =>  $H = \begin{pmatrix} H11 & 1.0 & -1.0 & H22 \end{pmatrix}$  flag = -2 =>  $H = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$  param may ONLY be stored on the device for the batched version of this function.
- **batchCount** – [in] [int] the number of x and y arrays, i.e. the number of batches.

*hipblasStatus\_t* **hipblasSrotmStridedBatched**(*hipblasHandle\_t* handle, int n, float \*x, int incx, *hipblasStride* stridex, float \*y, int incy, *hipblasStride* stridey, const float \*param, *hipblasStride* strideParam, int batchCount)

*hipblasStatus\_t* **hipblasDrotmStridedBatched**(*hipblasHandle\_t* handle, int n, double \*x, int incx, *hipblasStride* stridex, double \*y, int incy, *hipblasStride* stridey, const double \*param, *hipblasStride* strideParam, int batchCount)

BLAS Level 1 API.

rotmStridedBatched applies the modified Givens rotation matrix defined by param\_i to strided batched vectors x\_i and y\_i, for i = 1, ..., batchCount

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in the x and y vectors.
- **x** – [inout] device pointer pointing to first strided batched vector x\_1.
- **incx** – [in] [int] specifies the increment between elements of each x\_i.
- **stridex** – [in] [hipblasStride] specifies the increment between the beginning of x\_i and x\_(i + 1)
- **y** – [inout] device pointer pointing to first strided batched vector y\_1.
- **incy** – [in] [int] specifies the increment between elements of each y\_i.
- **stridey** – [in] [hipblasStride] specifies the increment between the beginning of y\_i and y\_(i + 1)
- **param** – [in] device pointer pointing to first array of 5 elements defining the rotation (param\_1). param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may ONLY be stored on the device for the strided\_batched version of this function.
- **strideParam** – [in] [hipblasStride] specifies the increment between the beginning of param\_i and param\_(i + 1)
- **batchCount** – [in] [int] the number of x and y arrays, i.e. the number of batches.

#### 1.8.1.11 hipblasXrotmg + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSrotmg**(*hipblasHandle\_t* handle, float \*d1, float \*d2, float \*x1, const float \*y1, float \*param)

*hipblasStatus\_t* **hipblasDrotmg**(*hipblasHandle\_t* handle, double \*d1, double \*d2, double \*x1, const double \*y1, double \*param)

BLAS Level 1 API.

rotmg creates the modified Givens rotation matrix for the vector (d1 \* x1, d2 \* y1). Parameters may be stored in either host or device memory, location is specified by calling hipblasSetPointerMode. If the pointer mode is set to HIPBLAS\_POINTER\_MODE\_HOST, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to HIPBLAS\_POINTER\_MODE\_DEVICE, this function returns immediately and synchronization is required to read the results.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : s,d

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **d1** – [inout] device pointer or host pointer to input scalar that is overwritten.
- **d2** – [inout] device pointer or host pointer to input scalar that is overwritten.
- **x1** – [inout] device pointer or host pointer to input scalar that is overwritten.
- **y1** – [in] device pointer or host pointer to input scalar.
- **param** – [out] device vector or host vector of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may be stored in either host or device memory, location is specified by calling hipblasSetPointerMode.

*hipblasStatus\_t* **hipblasSrotmgBatched**(*hipblasHandle\_t* handle, float \*const d1[], float \*const d2[], float \*const x1[], const float \*const y1[], float \*const param[], int batchCount)

*hipblasStatus\_t* **hipblasDrotmgBatched**(*hipblasHandle\_t* handle, double \*const d1[], double \*const d2[], double \*const x1[], const double \*const y1[], double \*const param[], int batchCount)

BLAS Level 1 API.

rotmgBatched creates the modified Givens rotation matrix for the batched vectors ( $d1_i * x1_i$ ,  $d2_i * y1_i$ ), for  $i = 1, \dots, \text{batchCount}$ . Parameters may be stored in either host or device memory, location is specified by calling hipblasSetPointerMode. If the pointer mode is set to HIPBLAS\_POINTER\_MODE\_HOST, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to HIPBLAS\_POINTER\_MODE\_DEVICE, this function returns immediately and synchronization is required to read the results.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **d1** – [inout] device batched array or host batched array of input scalars that is overwritten.
- **d2** – [inout] device batched array or host batched array of input scalars that is overwritten.
- **x1** – [inout] device batched array or host batched array of input scalars that is overwritten.
- **y1** – [in] device batched array or host batched array of input scalars.
- **param** – [out] device batched array or host batched array of vectors of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0

0.0 0.0 1.0 ) param may be stored in either host or device memory, location is specified by calling hipblasSetPointerMode.

- **batchCount** – [in] [int] the number of instances in the batch.

*hipblasStatus\_t* **hipblasSrotmgStridedBatched**(*hipblasHandle\_t* handle, float \*d1, *hipblasStride* strided1, float \*d2, *hipblasStride* strided2, float \*x1, *hipblasStride* strided1, const float \*y1, *hipblasStride* stridey1, float \*param, *hipblasStride* strideParam, int batchCount)

*hipblasStatus\_t* **hipblasDrotmgStridedBatched**(*hipblasHandle\_t* handle, double \*d1, *hipblasStride* strided1, double \*d2, *hipblasStride* strided2, double \*x1, *hipblasStride* strided1, const double \*y1, *hipblasStride* stridey1, double \*param, *hipblasStride* strideParam, int batchCount)

BLAS Level 1 API.

rotmgStridedBatched creates the modified Givens rotation matrix for the strided batched vectors ( $d1_i * x1_i$ ,  $d2_i * y1_i$ ), for  $i = 1, \dots, \text{batchCount}$ . Parameters may be stored in either host or device memory, location is specified by calling hipblasSetPointerMode. If the pointer mode is set to HIPBLAS\_POINTER\_MODE\_HOST, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to HIPBLAS\_POINTER\_MODE\_DEVICE, this function returns immediately and synchronization is required to read the results.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **d1** – [inout] device strided\_batched array or host strided\_batched array of input scalars that is overwritten.
- **strided1** – [in] [hipblasStride] specifies the increment between the beginning of  $d1_i$  and  $d1_{i+1}$
- **d2** – [inout] device strided\_batched array or host strided\_batched array of input scalars that is overwritten.
- **strided2** – [in] [hipblasStride] specifies the increment between the beginning of  $d2_i$  and  $d2_{i+1}$
- **x1** – [inout] device strided\_batched array or host strided\_batched array of input scalars that is overwritten.
- **strided1** – [in] [hipblasStride] specifies the increment between the beginning of  $x1_i$  and  $x1_{i+1}$
- **y1** – [in] device strided\_batched array or host strided\_batched array of input scalars.
- **stridey1** – [in] [hipblasStride] specifies the increment between the beginning of  $y1_i$  and  $y1_{i+1}$
- **param** – [out] device stridedBatched array or host stridedBatched array of vectors of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 =>  $H = \begin{pmatrix} H11 & H12 \\ H21 & H22 \end{pmatrix}$  flag = 0 =>  $H = \begin{pmatrix} 1.0 & H12 & H21 & 1.0 \end{pmatrix}$  flag = 1 =>  $H = \begin{pmatrix} H11 & 1.0 & -1.0 & H22 \end{pmatrix}$  flag =

-2 =>  $H = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$  param may be stored in either host or device memory, location is specified by calling `hipblasSetPointerMode`.

- **strideParam** – [in] [hipblasStride] specifies the increment between the beginning of param\_i and param\_(i + 1)
- **batchCount** – [in] [int] the number of instances in the batch.

### 1.8.1.12 hipblasXscal + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSscal**(*hipblasHandle\_t* handle, int n, const float \*alpha, float \*x, int incx)

*hipblasStatus\_t* **hipblasDscal**(*hipblasHandle\_t* handle, int n, const double \*alpha, double \*x, int incx)

*hipblasStatus\_t* **hipblasCscal**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*alpha, *hipblasComplex* \*x, int incx)

*hipblasStatus\_t* **hipblasCsscal**(*hipblasHandle\_t* handle, int n, const float \*alpha, *hipblasComplex* \*x, int incx)

*hipblasStatus\_t* **hipblasZscal**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*alpha, *hipblasDoubleComplex* \*x, int incx)

*hipblasStatus\_t* **hipblasZdscal**(*hipblasHandle\_t* handle, int n, const double \*alpha, *hipblasDoubleComplex* \*x, int incx)

BLAS Level 1 API.

scal scales each element of vector x with scalar alpha.

$$x := \alpha * x$$

- Supported precisions in rocBLAS : s,d,c,z,cs,zd
- Supported precisions in cuBLAS : s,d,c,z,cs,zd

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x.
- **alpha** – [in] device pointer or host pointer for the scalar alpha.
- **x** – [inout] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.

*hipblasStatus\_t* **hipblasSscalBatched**(*hipblasHandle\_t* handle, int n, const float \*alpha, float \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasDscalBatched**(*hipblasHandle\_t* handle, int n, const double \*alpha, double \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasCscalBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*alpha, *hipblasComplex* \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasZscalBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*alpha, *hipblasDoubleComplex* \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasCsscalBatched**(*hipblasHandle\_t* handle, int n, const float \*alpha, *hipblasComplex* \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasZdscalBatched**(*hipblasHandle\_t* handle, int n, const double \*alpha, *hipblasDoubleComplex* \*const x[], int incx, int batchCount)

BLAS Level 1 API.

scalBatched scales each element of vector  $x_i$  with scalar alpha, for  $i = 1, \dots, \text{batchCount}$ .

$$x_i := \alpha * x_i$$

where  $(x_i)$  is the  $i$ -th instance of the batch.

- Supported precisions in rocBLAS : s,d,c,z,cs,zd
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x_i$ .
- **alpha** – [in] host pointer or device pointer for the scalar alpha.
- **x** – [inout] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **batchCount** – [in] [int] specifies the number of batches in  $x$ .

*hipblasStatus\_t* **hipblasSscalStridedBatched**(*hipblasHandle\_t* handle, int n, const float \*alpha, float \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasDscalStridedBatched**(*hipblasHandle\_t* handle, int n, const double \*alpha, double \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasCscalStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasComplex* \*alpha, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasZscalStridedBatched**(*hipblasHandle\_t* handle, int n, const *hipblasDoubleComplex* \*alpha, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasCsscalStridedBatched**(*hipblasHandle\_t* handle, int n, const float \*alpha, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasZdscalStridedBatched**(*hipblasHandle\_t* handle, int n, const double \*alpha, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

BLAS Level 1 API.

scalStridedBatched scales each element of vector  $x_i$  with scalar alpha, for  $i = 1, \dots, \text{batchCount}$ .

$$x_i := \text{alpha} * x_i,$$

where  $(x_i)$  is the  $i$ -th instance of the batch.

- Supported precisions in rocBLAS : s,d,c,z,cs,zd
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x_i$ .
- **alpha** – [in] host pointer or device pointer for the scalar alpha.
- **x** – [inout] device pointer to the first vector ( $x_1$ ) in the batch.
- **incx** – [in] [int] specifies the increment for the elements of  $x$ .
- **stridex** – [in] [*hipblasStride*] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ). There are no restrictions placed on stride\_x, however the user should take care to ensure that stride\_x is of appropriate size, for a typical case this means stride\_x  $\geq n * \text{incx}$ .
- **batchCount** – [in] [int] specifies the number of batches in  $x$ .

#### 1.8.1.13 hipblasXswap + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSswap**(*hipblasHandle\_t* handle, int n, float \*x, int incx, float \*y, int incy)

*hipblasStatus\_t* **hipblasDswap**(*hipblasHandle\_t* handle, int n, double \*x, int incx, double \*y, int incy)

*hipblasStatus\_t* **hipblasCswap**(*hipblasHandle\_t* handle, int n, *hipblasComplex* \*x, int incx, *hipblasComplex* \*y, int incy)

*hipblasStatus\_t* **hipblasZswap**(*hipblasHandle\_t* handle, int n, *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*y, int incy)

BLAS Level 1 API.

swap interchanges vectors  $x$  and  $y$ .

$$y := x; x := y$$

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

**Parameters**

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x and y.
- **x** – [inout] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [inout] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.

*hipblasStatus\_t* **hipblasSswapBatched**(*hipblasHandle\_t* handle, int n, float \*x[], int incx, float \*y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasDswapBatched**(*hipblasHandle\_t* handle, int n, double \*x[], int incx, double \*y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasCswapBatched**(*hipblasHandle\_t* handle, int n, *hipblasComplex* \*x[], int incx, *hipblasComplex* \*y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasZswapBatched**(*hipblasHandle\_t* handle, int n, *hipblasDoubleComplex* \*x[], int incx, *hipblasDoubleComplex* \*y[], int incy, int batchCount)

BLAS Level 1 API.

swapBatched interchanges vectors x<sub>i</sub> and y<sub>i</sub>, for i = 1, ..., batchCount

$$y_i := x_i; x_i := y_i$$

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

**Parameters**

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each x<sub>i</sub> and y<sub>i</sub>.
- **x** – [inout] device array of device pointers storing each vector x<sub>i</sub>.
- **incx** – [in] [int] specifies the increment for the elements of each x<sub>i</sub>.
- **y** – [inout] device array of device pointers storing each vector y<sub>i</sub>.
- **incy** – [in] [int] specifies the increment for the elements of each y<sub>i</sub>.
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasSswapStridedBatched**(*hipblasHandle\_t* handle, int n, float \*x, int incx, *hipblasStride* stridex, float \*y, int incy, *hipblasStride* stridey, int batchCount)



*hipblasStatus\_t* **hipblasDswapStridedBatched**(*hipblasHandle\_t* handle, int n, double \*x, int incx, *hipblasStride* stridex, double \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasCswapStridedBatched**(*hipblasHandle\_t* handle, int n, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, *hipblasComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasZswapStridedBatched**(*hipblasHandle\_t* handle, int n, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

BLAS Level 1 API.

swapStridedBatched interchanges vectors  $x_i$  and  $y_i$ , for  $i = 1, \dots, \text{batchCount}$

$y_i := x_i; x_i := y_i$
--------------------------

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x_i$  and  $y_i$ .
- **x** – [inout] device pointer to the first vector  $x_1$ .
- **incx** – [in] [int] specifies the increment for the elements of x.
- **stridex** – [in] [*hipblasStride*] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ). There are no restrictions placed on stride\_x, however the user should take care to ensure that stride\_x is of appropriate size, for a typical case this means stride\_x  $\geq n * \text{incx}$ .
- **y** – [inout] device pointer to the first vector  $y_1$ .
- **incy** – [in] [int] specifies the increment for the elements of y.
- **stridey** – [in] [*hipblasStride*] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ ). There are no restrictions placed on stride\_y, however the user should take care to ensure that stride\_y is of appropriate size, for a typical case this means stride\_y  $\geq n * \text{incy}$ . stridey should be non zero.
- **batchCount** – [in] [int] number of instances in the batch.

## 1.8.2 Level 2 BLAS

#### List of Level-2 BLAS Functions

- *hipblasXgbmv + Batched, StridedBatched*
- *hipblasXgemv + Batched, StridedBatched*
- *hipblasXger + Batched, StridedBatched*
- *hipblasXhbmvm + Batched, StridedBatched*

- *hipblasXhemv + Batched, StridedBatched*
- *hipblasXher + Batched, StridedBatched*
- *hipblasXher2 + Batched, StridedBatched*
- *hipblasXhpmv + Batched, StridedBatched*
- *hipblasXhpr + Batched, StridedBatched*
- *hipblasXhpr2 + Batched, StridedBatched*
- *hipblasXsbmv + Batched, StridedBatched*
- *hipblasXspmv + Batched, StridedBatched*
- *hipblasXspr + Batched, StridedBatched*
- *hipblasXspr2 + Batched, StridedBatched*
- *hipblasXsymv + Batched, StridedBatched*
- *hipblasXsyr + Batched, StridedBatched*
- *hipblasXsyr2 + Batched, StridedBatched*
- *hipblasXtbmv + Batched, StridedBatched*
- *hipblasXtbsv + Batched, StridedBatched*
- *hipblasXtpmv + Batched, StridedBatched*
- *hipblasXtpsv + Batched, StridedBatched*
- *hipblasXtrmv + Batched, StridedBatched*
- *hipblasXtrsv + Batched, StridedBatched*

### 1.8.2.1 hipblasXgbmv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSgbmv**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const float \*alpha, const float \*AP, int lda, const float \*x, int incx, const float \*beta, float \*y, int incy)

*hipblasStatus\_t* **hipblasDgbmv**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const double \*alpha, const double \*AP, int lda, const double \*x, int incx, const double \*beta, double \*y, int incy)

*hipblasStatus\_t* **hipblasCgbmv**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy)

*hipblasStatus\_t* **hipblasZgbmv**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*y, int incy)

BLAS Level 2 API.

gbmv performs one of the matrix-vector operations

```

y := alpha*A*x    + beta*y,    or
y := alpha*A**T*x + beta*y,    or
y := alpha*A**H*x + beta*y,

```

where alpha and beta are scalars, x and y are vectors and A is an m by n banded matrix with kl sub-diagonals and ku super-diagonals.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **trans** – [in] [hipblasOperation\_t] indicates whether matrix A is tranposed (conjugated) or not
- **m** – [in] [int] number of rows of matrix A
- **n** – [in] [int] number of columns of matrix A
- **kl** – [in] [int] number of sub-diagonals of A
- **ku** – [in] [int] number of super-diagonals of A
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device pointer storing banded matrix A. Leading (kl + ku + 1) by n part of the matrix contains the coefficients of the banded matrix. The leading diagonal resides in row (ku + 1) with the first super-diagonal above on the RHS of row ku. The first sub-diagonal resides below on the LHS of row ku + 2. This propogates up and down across sub/super-diagonals. Ex: (m = n = 7; ku = 2, kl = 2) 1 2 3 0 0 0 0 0 3 3 3 3 3 4 1 2 3 0 0 0 2 2 2 2 2 5 4 1 2 3 0 0 -> 1 1 1 1 1 1 1 0 5 4 1 2 3 0 4 4 4 4 4 0 0 0 5 4 1 2 0 5 5 5 5 5 0 0 0 0 5 4 1 2 0 0 0 0 0 0 0 0 0 0 5 4 1 0 0 0 0 0 0 Note that the empty elements which don't correspond to data will not be referenced.
- **lda** – [in] [int] specifies the leading dimension of A. Must be >= (kl + ku + 1)
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.

*hipblasStatus\_t* **hipblasSgbmvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const float \*alpha, const float \*const AP[], int lda, const float \*const x[], int incx, const float \*beta, float \*const y[], int incy, int batchSize)

*hipblasStatus\_t* **hipblasDgbmvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const double \*alpha, const double \*const AP[], int lda, const double \*const x[], int incx, const double \*beta, double \*const y[], int incy, int batchSize)

*hipblasStatus\_t* **hipblasCgbmvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasZgbmvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const y[], int incy, int batchCount)

BLAS Level 2 API.

gbmvBatched performs one of the matrix-vector operations

$y_i := \alpha A_i x_i + \beta y_i,$ <b>or</b> $y_i := \alpha A_i^T x_i + \beta y_i,$ <b>or</b> $y_i := \alpha A_i^H x_i + \beta y_i,$
--

where ( $A_i, x_i, y_i$ ) is the  $i$ -th instance of the batch.  $\alpha$  and  $\beta$  are scalars,  $x_i$  and  $y_i$  are vectors and  $A_i$  is an  $m$  by  $n$  banded matrix with  $kl$  sub-diagonals and  $ku$  super-diagonals, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **trans** – [in] [*hipblasOperation\_t*] indicates whether matrix  $A$  is tranposed (conjugated) or not
- **m** – [in] [int] number of rows of each matrix  $A_i$
- **n** – [in] [int] number of columns of each matrix  $A_i$
- **kl** – [in] [int] number of sub-diagonals of each  $A_i$
- **ku** – [in] [int] number of super-diagonals of each  $A_i$
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$ .
- **AP** – [in] device array of device pointers storing each banded matrix  $A_i$ . Leading  $(kl + ku + 1)$  by  $n$  part of the matrix contains the coefficients of the banded matrix. The leading diagonal resides in row  $(ku + 1)$  with the first super-diagonal above on the RHS of row  $ku$ . The first sub-diagonal resides below on the LHS of row  $ku + 2$ . This propagates up and down across sub/super-diagonals. Ex: ( $m = n = 7$ ;  $ku = 2, kl = 2$ )  
1 2 3 0 0 0 0 0 0 3 3 3 3 4 1 2  
3 0 0 0 2 2 2 2 2 5 4 1 2 3 0 0 -> 1 1 1 1 1 1 1 0 5 4 1 2 3 0 4 4 4 4 4 0 0 0 5 4 1 2 0 5 5  
5 5 5 0 0 0 0 0 5 4 1 2 0 0 0 0 0 0 0 0 0 0 5 4 1 0 0 0 0 0 0 0 Note that the empty elements which don't correspond to data will not be referenced.
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ . Must be  $\geq (kl + ku + 1)$
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **beta** – [in] device pointer or host pointer to scalar  $\beta$ .
- **y** – [inout] device array of device pointers storing each vector  $y_i$ .

- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **batchCount** – [in] [int] specifies the number of instances in the batch.

*hipblasStatus\_t* **hipblasSgbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const float \*alpha, const float \*AP, int lda, *hipblasStride* strideA, const float \*x, int incx, *hipblasStride* stridex, const float \*beta, float \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasDgbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const double \*alpha, const double \*AP, int lda, *hipblasStride* strideA, const double \*x, int incx, *hipblasStride* stridex, const double \*beta, double \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasCgbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasZgbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, int kl, int ku, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

BLAS Level 2 API.

gbmvStridedBatched performs one of the matrix-vector operations

$y_i := \alpha A_i x_i + \beta y_i, \quad \text{or}$ $y_i := \alpha A_i^T x_i + \beta y_i, \quad \text{or}$ $y_i := \alpha A_i^H x_i + \beta y_i,$
--

where  $(A_i, x_i, y_i)$  is the  $i$ -th instance of the batch.  $\alpha$  and  $\beta$  are scalars,  $x_i$  and  $y_i$  are vectors and  $A_i$  is an  $m$  by  $n$  banded matrix with  $kl$  sub-diagonals and  $ku$  super-diagonals, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **trans** – [in] [*hipblasOperation\_t*] indicates whether matrix  $A$  is transposed (conjugated) or not
- **m** – [in] [int] number of rows of matrix  $A$
- **n** – [in] [int] number of columns of matrix  $A$
- **kl** – [in] [int] number of sub-diagonals of  $A$

- **ku** – [in] [int] number of super-diagonals of A
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device pointer to first banded matrix (A<sub>1</sub>). Leading (kl + ku + 1) by n part of the matrix contains the coefficients of the banded matrix. The leading diagonal resides in row (ku + 1) with the first super-diagonal above on the RHS of row ku. The first sub-diagonal resides below on the LHS of row ku + 2. This propagates up and down across sub/super-diagonals. Ex: (m = n = 7; ku = 2, kl = 2) 1 2 3 0 0 0 0 0 3 3 3 3 3 4 1 2 3 0 0 0 0 2 2 2 2 2 5 4 1 2 3 0 0 -> 1 1 1 1 1 1 1 0 5 4 1 2 3 0 4 4 4 4 4 0 0 0 5 4 1 2 0 5 5 5 5 5 0 0 0 0 5 4 1 2 0 0 0 0 0 0 0 0 0 0 5 4 1 0 0 0 0 0 0 Note that the empty elements which don't correspond to data will not be referenced.
- **lda** – [in] [int] specifies the leading dimension of A. Must be >= (kl + ku + 1)
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A<sub>i</sub>) and the next one (A<sub>i+1</sub>)
- **x** – [in] device pointer to first vector (x<sub>1</sub>).
- **incx** – [in] [int] specifies the increment for the elements of x.
- **stridex** – [in] [hipblasStride] stride from the start of one vector (x<sub>i</sub>) and the next one (x<sub>i+1</sub>)
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device pointer to first vector (y<sub>1</sub>).
- **incy** – [in] [int] specifies the increment for the elements of y.
- **stridey** – [in] [hipblasStride] stride from the start of one vector (y<sub>i</sub>) and the next one (y<sub>i+1</sub>)
- **batchCount** – [in] [int] specifies the number of instances in the batch.

### 1.8.2.2 hipblasXgemv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSgemv**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, const float \*alpha, const float \*AP, int lda, const float \*x, int incx, const float \*beta, float \*y, int incy)

*hipblasStatus\_t* **hipblasDgemv**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, const double \*alpha, const double \*AP, int lda, const double \*x, int incx, const double \*beta, double \*y, int incy)

*hipblasStatus\_t* **hipblasCgemv**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy)

*hipblasStatus\_t* **hipblasZgemv**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*y, int incy)

BLAS Level 2 API.

gemv performs one of the matrix-vector operations

```

y := alpha*A*x      + beta*y,   or
y := alpha*A**T*x  + beta*y,   or
y := alpha*A**H*x  + beta*y,

```

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **trans** – [in] [hipblasOperation\_t] indicates whether matrix A is tranposed (conjugated) or not
- **m** – [in] [int] number of rows of matrix A
- **n** – [in] [int] number of columns of matrix A
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device pointer storing matrix A.
- **lda** – [in] [int] specifies the leading dimension of A.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.

*hipblasStatus\_t* **hipblasSgemvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, const float \*alpha, const float \*const AP[], int lda, const float \*const x[], int incx, const float \*beta, float \*const y[], int incy, int batchSize)

*hipblasStatus\_t* **hipblasDgemvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, const double \*alpha, const double \*const AP[], int lda, const double \*const x[], int incx, const double \*beta, double \*const y[], int incy, int batchSize)

*hipblasStatus\_t* **hipblasCgemvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*const y[], int incy, int batchSize)

*hipblasStatus\_t* **hipblasZgemvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* trans, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const y[], int incy, int batchSize)

BLAS Level 2 API.

gemvBatched performs a batch of matrix-vector operations

```

y_i := alpha*A_i*x_i    + beta*y_i,   or
y_i := alpha*A_i**T*x_i + beta*y_i,   or
y_i := alpha*A_i**H*x_i + beta*y_i,

```

where (A\_i, x\_i, y\_i) is the i-th instance of the batch. alpha and beta are scalars, x\_i and y\_i are vectors and A\_i is an m by n matrix, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **trans** – [in] [hipblasOperation\_t] indicates whether matrices A\_i are tranposed (conjugated) or not
- **m** – [in] [int] number of rows of each matrix A\_i
- **n** – [in] [int] number of columns of each matrix A\_i
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device array of device pointers storing each matrix A\_i.
- **lda** – [in] [int] specifies the leading dimension of each matrix A\_i.
- **x** – [in] device array of device pointers storing each vector x\_i.
- **incx** – [in] [int] specifies the increment for the elements of each vector x\_i.
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device array of device pointers storing each vector y\_i.
- **incy** – [in] [int] specifies the increment for the elements of each vector y\_i.
- **batchCount** – [in] [int] number of instances in the batch

*hipblasStatus\_t* **hipblasSgemvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, int m, int n, const float \*alpha, const float \*AP, int lda, *hipblasStride* strideA, const float \*x, int incx, *hipblasStride* stridex, const float \*beta, float \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasDgemvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, int m, int n, const double \*alpha, const double \*AP, int lda, *hipblasStride* strideA, const double \*x, int incx, *hipblasStride* stridex, const double \*beta, double \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasCgemvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)



```
hipblasStatus_t hipblasZgemvStridedBatched(hipblasHandle_t handle, hipblasOperation_t transA, int m, int n,
const hipblasDoubleComplex *alpha, const
hipblasDoubleComplex *AP, int lda, hipblasStride strideA, const
hipblasDoubleComplex *x, int incx, hipblasStride stridex, const
hipblasDoubleComplex *beta, hipblasDoubleComplex *y, int
incy, hipblasStride stridey, int batchCount)
```

BLAS Level 2 API.

gemvStridedBatched performs a batch of matrix-vector operations

```
y_i := alpha*A_i*x_i + beta*y_i,   or
y_i := alpha*A_i**T*x_i + beta*y_i, or
y_i := alpha*A_i**H*x_i + beta*y_i,
```

where (A\_i, x\_i, y\_i) is the i-th instance of the batch. alpha and beta are scalars, x\_i and y\_i are vectors and A\_i is an m by n matrix, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **transA** – [in] [hipblasOperation\_t] indicates whether matrices A\_i are tranposed (conjugated) or not
- **m** – [in] [int] number of rows of matrices A\_i
- **n** – [in] [int] number of columns of matrices A\_i
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device pointer to the first matrix (A\_1) in the batch.
- **lda** – [in] [int] specifies the leading dimension of matrices A\_i.
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A\_i) and the next one (A\_i+1)
- **x** – [in] device pointer to the first vector (x\_1) in the batch.
- **incx** – [in] [int] specifies the increment for the elements of vectors x\_i.
- **stridex** – [in] [hipblasStride] stride from the start of one vector (x\_i) and the next one (x\_i+1). There are no restrictions placed on stridex, however the user should take care to ensure that stridex is of appropriate size. When trans equals HIPBLAS\_OP\_N this typically means stridex >= n \* incx, otherwise stridex >= m \* incx.
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device pointer to the first vector (y\_1) in the batch.
- **incy** – [in] [int] specifies the increment for the elements of vectors y\_i.
- **stridey** – [in] [hipblasStride] stride from the start of one vector (y\_i) and the next one (y\_i+1). There are no restrictions placed on stridey, however the user should take care to ensure that stridey is of appropriate size. When trans equals HIPBLAS\_OP\_N this typically means stridey >= m \* incy, otherwise stridey >= n \* incy. stridey should be non zero.
- **batchCount** – [in] [int] number of instances in the batch

### 1.8.2.3 hipblasXger + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSger**(*hipblasHandle\_t* handle, int m, int n, const float \*alpha, const float \*x, int incx, const float \*y, int incy, float \*AP, int lda)

*hipblasStatus\_t* **hipblasDger**(*hipblasHandle\_t* handle, int m, int n, const double \*alpha, const double \*x, int incx, const double \*y, int incy, double \*AP, int lda)

*hipblasStatus\_t* **hipblasCgeru**(*hipblasHandle\_t* handle, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*y, int incy, *hipblasComplex* \*AP, int lda)

*hipblasStatus\_t* **hipblasCgerc**(*hipblasHandle\_t* handle, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*y, int incy, *hipblasComplex* \*AP, int lda)

*hipblasStatus\_t* **hipblasZgeru**(*hipblasHandle\_t* handle, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*y, int incy, *hipblasDoubleComplex* \*AP, int lda)

*hipblasStatus\_t* **hipblasZgerc**(*hipblasHandle\_t* handle, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*y, int incy, *hipblasDoubleComplex* \*AP, int lda)

BLAS Level 2 API.

ger,geru,gerc performs the matrix-vector operations

$A := A + \alpha x y^{**T}$  , OR  
 $A := A + \alpha x y^{**H}$  **for** gerc

where alpha is a scalar, x and y are vectors, and A is an m by n matrix.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **m** – [in] [int] the number of rows of the matrix A.
- **n** – [in] [int] the number of columns of the matrix A.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [in] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **AP** – [inout] device pointer storing matrix A.
- **lda** – [in] [int] specifies the leading dimension of A.

*hipblasStatus\_t* **hipblasSgerBatched**(*hipblasHandle\_t* handle, int m, int n, const float \*alpha, const float \*const x[], int incx, const float \*const y[], int incy, float \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasDgerBatched**(*hipblasHandle\_t* handle, int m, int n, const double \*alpha, const double \*const x[], int incx, const double \*const y[], int incy, double \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasCgeruBatched**(*hipblasHandle\_t* handle, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*const y[], int incy, *hipblasComplex* \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasCgercBatched**(*hipblasHandle\_t* handle, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*const y[], int incy, *hipblasComplex* \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasZgeruBatched**(*hipblasHandle\_t* handle, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*const y[], int incy, *hipblasDoubleComplex* \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasZgercBatched**(*hipblasHandle\_t* handle, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*const y[], int incy, *hipblasDoubleComplex* \*const AP[], int lda, int batchCount)

BLAS Level 2 API.

gerBatched,geruBatched,gercBatched performs a batch of the matrix-vector operations

$A := A + \alpha * x * y^{**T}$  , OR  
 $A := A + \alpha * x * y^{**H}$  **for** gerc

where (A<sub>i</sub>, x<sub>i</sub>, y<sub>i</sub>) is the i-th instance of the batch. alpha is a scalar, x<sub>i</sub> and y<sub>i</sub> are vectors and A<sub>i</sub> is an m by n matrix, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **m** – [in] [int] the number of rows of each matrix A<sub>i</sub>.
- **n** – [in] [int] the number of columns of each matrix A<sub>i</sub>.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device array of device pointers storing each vector x<sub>i</sub>.
- **incx** – [in] [int] specifies the increment for the elements of each vector x<sub>i</sub>.
- **y** – [in] device array of device pointers storing each vector y<sub>i</sub>.
- **incy** – [in] [int] specifies the increment for the elements of each vector y<sub>i</sub>.

- **AP** – [inout] device array of device pointers storing each matrix  $A_i$ .
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .
- **batchCount** – [in] [int] number of instances in the batch

*hipblasStatus\_t* **hipblasSgerStridedBatched**(*hipblasHandle\_t* handle, int m, int n, const float \*alpha, const float \*x, int incx, *hipblasStride* stridex, const float \*y, int incy, *hipblasStride* stridey, float \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasDgerStridedBatched**(*hipblasHandle\_t* handle, int m, int n, const double \*alpha, const double \*x, int incx, *hipblasStride* stridex, const double \*y, int incy, *hipblasStride* stridey, double \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasCgeruStridedBatched**(*hipblasHandle\_t* handle, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*y, int incy, *hipblasStride* stridey, *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasCgercStridedBatched**(*hipblasHandle\_t* handle, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*y, int incy, *hipblasStride* stridey, *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasZgeruStridedBatched**(*hipblasHandle\_t* handle, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasZgercStridedBatched**(*hipblasHandle\_t* handle, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

BLAS Level 2 API.

gerStridedBatched,geruStridedBatched,gercStridedBatched performs the matrix-vector operations

$A_i := A_i + \alpha x_i y_i^{**T}$ , OR  
 $A_i := A_i + \alpha x_i y_i^{**H}$  **for** gerc

where  $(A_i, x_i, y_i)$  is the  $i$ -th instance of the batch.  $\alpha$  is a scalar,  $x_i$  and  $y_i$  are vectors and  $A_i$  is an  $m$  by  $n$  matrix, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

**Parameters**

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **m** – [in] [int] the number of rows of each matrix  $A_i$ .
- **n** – [in] [int] the number of columns of each matrix  $A_i$ .
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer to the first vector ( $x_1$ ) in the batch.
- **incx** – [in] [int] specifies the increments for the elements of each vector  $x_i$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ). There are no restrictions placed on stridex, however the user should take care to ensure that stridex is of appropriate size, for a typical case this means  $\text{stridex} \geq m * \text{incx}$ .
- **y** – [inout] device pointer to the first vector ( $y_1$ ) in the batch.
- **incy** – [in] [int] specifies the increment for the elements of each vector  $y_i$ .
- **stridey** – [in] [hipblasStride] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ ). There are no restrictions placed on stridey, however the user should take care to ensure that stridey is of appropriate size, for a typical case this means  $\text{stridey} \geq n * \text{incy}$ .
- **AP** – [inout] device pointer to the first matrix ( $A_1$ ) in the batch.
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .
- **strideA** – [in] [hipblasStride] stride from the start of one matrix ( $A_i$ ) and the next one ( $A_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch

**1.8.2.4 hipblasXhbmvm + Batched, StridedBatched**

*hipblasStatus\_t* **hipblasChbmvm**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy)

*hipblasStatus\_t* **hipblasZhbmvm**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*y, int incy)

BLAS Level 2 API.

hbmvm performs the matrix-vector operations

$$y := \alpha * A * x + \beta * y$$

where alpha and beta are scalars, x and y are n element vectors and A is an n by n Hermitian band matrix, with k super-diagonals.

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

if `uplo == HIPBLAS_FILL_MODE_LOWER`: The leading  $(k + 1)$  by  $n$  part of  $A$  must contain the lower triangular band part of the Hermitian matrix, with the leading diagonal in row (1), the first sub-diagonal on the LHS of row 2, etc. The bottom right  $k$  by  $k$  triangle of  $A$  will not be referenced. Ex (lower,  $lda = 2$ ,  $n = 4$ ,  $k = 1$ ):  $A$  Represented matrix (1,0) (2,0) (3,0) (4,0) (1, 0) (5,-9) (0, 0) (0, 0) (5,9) (6,8) (7,7) (0,0) (5, 9) (2, 0) (6,-8) (0, 0) (0, 0) (6, 8) (3, 0) (7,-7) (0, 0) (0, 0) (7, 7) (4, 0)

As a Hermitian matrix, the imaginary part of the main diagonal of  $A$  will not be referenced and is assumed to be  $== 0$ .

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of  $A$  is being supplied. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of  $A$  is being supplied.
- **n** – [in] [int] the order of the matrix  $A$ .
- **k** – [in] [int] the number of super-diagonals of the matrix  $A$ . Must be  $\geq 0$ .
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$ .
- **AP** – [in] device pointer storing matrix  $A$ . Of dimension  $(lda, n)$ . if `uplo == HIPBLAS_FILL_MODE_UPPER`: The leading  $(k + 1)$  by  $n$  part of  $A$  must contain the upper triangular band part of the Hermitian matrix, with the leading diagonal in row  $(k + 1)$ , the first super-diagonal on the RHS of row  $k$ , etc. The top left  $k$  by  $x$  triangle of  $A$  will not be referenced. Ex (upper,  $lda = n = 4$ ,  $k = 1$ ):  $A$  Represented matrix (0,0) (5,9) (6,8) (7,7) (1, 0) (5, 9) (0, 0) (0, 0) (1,0) (2,0) (3,0) (4,0) (5,-9) (2, 0) (6, 8) (0, 0) (0,0) (0,0) (0,0) (0,0) (0, 0) (6,-8) (3, 0) (7, 7) (0,0) (0,0) (0,0) (0,0) (0, 0) (0, 0) (7,-7) (4, 0)
- **lda** – [in] [int] specifies the leading dimension of  $A$ . must be  $\geq k + 1$
- **x** – [in] device pointer storing vector  $x$ .
- **incx** – [in] [int] specifies the increment for the elements of  $x$ .
- **beta** – [in] device pointer or host pointer to scalar  $\beta$ .
- **y** – [inout] device pointer storing vector  $y$ .
- **incy** – [in] [int] specifies the increment for the elements of  $y$ .

*hipblasStatus\_t* **hipblasChbmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*const y[], int incy, int batchSize)

*hipblasStatus\_t* **hipblasZhbmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const y[], int incy, int batchSize)

BLAS Level 2 API.

hbmvmBatched performs one of the matrix-vector operations

$$y_i := \alpha * A_{i,i} * x_i + \beta * y_i$$

where alpha and beta are scalars,  $x_i$  and  $y_i$  are  $n$  element vectors and  $A_i$  is an  $n$  by  $n$  Hermitian band matrix with  $k$  super-diagonals, for each batch in  $i = [1, \text{batchCount}]$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

if `uplo == HIPBLAS_FILL_MODE_LOWER`: The leading  $(k + 1)$  by  $n$  part of each  $A_i$  must contain the lower triangular band part of the Hermitian matrix, with the leading diagonal in row (1), the first sub-diagonal on the LHS of row 2, etc. The bottom right  $k$  by  $k$  triangle of each  $A_i$  will not be referenced. Ex (lower,  $\text{lda} = 2$ ,  $n = 4$ ,  $k = 1$ ): A Represented matrix (1,0) (2,0) (3,0) (4,0) (1, 0) (5,-9) (0, 0) (0, 0) (5,9) (6,8) (7,7) (0,0) (5, 9) (2, 0) (6,-8) (0, 0) (0, 0) (6, 8) (3, 0) (7,-7) (0, 0) (0, 0) (7, 7) (4, 0)

As a Hermitian matrix, the imaginary part of the main diagonal of each  $A_i$  will not be referenced and is assumed to be == 0.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is being supplied. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is being supplied.
- **n** – [in] [int] the order of each matrix  $A_i$ .
- **k** – [in] [int] the number of super-diagonals of each matrix  $A_i$ . Must be  $\geq 0$ .
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device array of device pointers storing each matrix  $A_i$  of dimension  $(\text{lda}, n)$ . if `uplo == HIPBLAS_FILL_MODE_UPPER`: The leading  $(k + 1)$  by  $n$  part of each  $A_i$  must contain the upper triangular band part of the Hermitian matrix, with the leading diagonal in row  $(k + 1)$ , the first super-diagonal on the RHS of row  $k$ , etc. The top left  $k$  by  $x$  triangle of each  $A_i$  will not be referenced. Ex (upper,  $\text{lda} = n = 4$ ,  $k = 1$ ): A Represented matrix (0,0) (5,9) (6,8) (7,7) (1, 0) (5, 9) (0, 0) (0, 0) (1,0) (2,0) (3,0) (4,0) (5,-9) (2, 0) (6, 8) (0, 0) (0,0) (0,0) (0,0) (0, 0) (6,-8) (3, 0) (7, 7) (0,0) (0,0) (0,0) (0,0) (0, 0) (0, 0) (7,-7) (4, 0)
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ . must be  $\geq \max(1, n)$
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device array of device pointers storing each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of  $y$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasChbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

```

hipblasStatus_t hipblasZhbmvStridedBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, int k,
const hipblasDoubleComplex *alpha, const
hipblasDoubleComplex *AP, int lda, hipblasStride strideA, const
hipblasDoubleComplex *x, int incx, hipblasStride stridex, const
hipblasDoubleComplex *beta, hipblasDoubleComplex *y, int
incy, hipblasStride stridey, int batchCount)

```

BLAS Level 2 API.

hbmvStridedBatched performs one of the matrix-vector operations

$$y_i := \alpha * A_i * x_i + \beta * y_i$$

where alpha and beta are scalars,  $x_i$  and  $y_i$  are  $n$  element vectors and  $A_i$  is an  $n$  by  $n$  Hermitian band matrix with  $k$  super-diagonals, for each batch in  $i = [1, \text{batchCount}]$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The leading  $(k + 1)$  by  $n$  part of each  $A_i$  must contain the lower triangular band part of the Hermitian matrix, with the leading diagonal in row (1), the first sub-diagonal on the LHS of row 2, etc. The bottom right  $k$  by  $k$  triangle of each  $A_i$  will not be referenced. Ex (lower, lda = 2, n = 4, k = 1): A Represented matrix (1,0) (2,0) (3,0) (4,0) (1, 0) (5,-9) (0, 0) (0, 0) (5,9) (6,8) (7,7) (0,0) (5, 9) (2, 0) (6,-8) (0, 0) (0, 0) (6, 8) (3, 0) (7,-7) (0, 0) (0, 0) (7, 7) (4, 0)

As a Hermitian matrix, the imaginary part of the main diagonal of each  $A_i$  will not be referenced and is assumed to be == 0.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is being supplied. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is being supplied.
- **n** – [in] [int] the order of each matrix  $A_i$ .
- **k** – [in] [int] the number of super-diagonals of each matrix  $A_i$ . Must be  $\geq 0$ .
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device array pointing to the first matrix  $A_1$ . Each  $A_i$  is of dimension (lda, n). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The leading  $(k + 1)$  by  $n$  part of each  $A_i$  must contain the upper triangular band part of the Hermitian matrix, with the leading diagonal in row  $(k + 1)$ , the first super-diagonal on the RHS of row  $k$ , etc. The top left  $k$  by  $x$  triangle of each  $A_i$  will not be referenced. Ex (upper, lda = n = 4, k = 1): A Represented matrix (0,0) (5,9) (6,8) (7,7) (1, 0) (5, 9) (0, 0) (0, 0) (1,0) (2,0) (3,0) (4,0) (5,-9) (2, 0) (6, 8) (0, 0) (0,0) (0,0) (0,0) (0, 0) (6,-8) (3, 0) (7, 7) (0,0) (0,0) (0,0) (0,0) (0, 0) (0, 0) (7,-7) (4, 0)
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ . must be  $\geq \max(1, n)$
- **strideA** – [in] [hipblasStride] stride from the start of one matrix ( $A_i$ ) and the next one ( $A_{i+1}$ )
- **x** – [in] device array pointing to the first vector  $y_1$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .



- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ )
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device array pointing to the first vector  $y_1$ .
- **incy** – [in] [int] specifies the increment for the elements of y.
- **stridey** – [in] [hipblasStride] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.2.5 hipblasXhemv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasChemv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy)

*hipblasStatus\_t* **hipblasZhemv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*y, int incy)

BLAS Level 2 API.

hemv performs one of the matrix-vector operations

$$y := \alpha * A * x + \beta * y$$

where alpha and beta are scalars, x and y are n element vectors and A is an n by n Hermitian matrix.

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: the upper triangular part of the Hermitian matrix A is supplied. HIPBLAS\_FILL\_MODE\_LOWER: the lower triangular part of the Hermitian matrix A is supplied.
- **n** – [in] [int] the order of the matrix A.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device pointer storing matrix A. Of dimension (lda, n). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of A must contain the upper triangular part of a Hermitian matrix. The lower triangular part of A will not be referenced. if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of A must contain the lower triangular part of a Hermitian matrix. The upper triangular part of A will not be referenced. As a Hermitian matrix, the imaginary part of the main diagonal of A will not be referenced and is assumed to be == 0.
- **lda** – [in] [int] specifies the leading dimension of A. must be  $\geq \max(1, n)$
- **x** – [in] device pointer storing vector x.

- **incx** – [in] [int] specifies the increment for the elements of x.
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.

*hipblasStatus\_t* **hipblasChemvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasZhemvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const y[], int incy, int batchCount)

BLAS Level 2 API.

hemvBatched performs one of the matrix-vector operations

$$y_i := \alpha * A_i * x_i + \beta * y_i$$

where alpha and beta are scalars,  $x_i$  and  $y_i$  are n element vectors and  $A_i$  is an n by n Hermitian matrix, for each batch in  $i = [1, \text{batchCount}]$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER: the upper triangular part of the Hermitian matrix A is supplied. HIPBLAS\_FILL\_MODE\_LOWER: the lower triangular part of the Hermitian matrix A is supplied.
- **n** – [in] [int] the order of each matrix  $A_i$ .
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device array of device pointers storing each matrix  $A_i$  of dimension (lda, n). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  must contain the upper triangular part of a Hermitian matrix. The lower triangular part of each  $A_i$  will not be referenced. if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  must contain the lower triangular part of a Hermitian matrix. The upper triangular part of each  $A_i$  will not be referenced. As a Hermitian matrix, the imaginary part of the main diagonal of each  $A_i$  will not be referenced and is assumed to be == 0.
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ . must be  $\geq \max(1, n)$
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device array of device pointers storing each vector  $y_i$ .

- **incy** – [in] [int] specifies the increment for the elements of y.
- **batchCount** – [in] [int] number of instances in the batch.

```
hipblasStatus_t hipblasChemvStridedBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
hipblasComplex *alpha, const hipblasComplex *AP, int lda,
hipblasStride strideA, const hipblasComplex *x, int incx,
hipblasStride stridex, const hipblasComplex *beta,
hipblasComplex *y, int incy, hipblasStride stridey, int
batchCount)
```

```
hipblasStatus_t hipblasZhemvStridedBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
hipblasDoubleComplex *alpha, const hipblasDoubleComplex
*AP, int lda, hipblasStride strideA, const hipblasDoubleComplex
*x, int incx, hipblasStride stridex, const hipblasDoubleComplex
*beta, hipblasDoubleComplex *y, int incy, hipblasStride stridey,
int batchCount)
```

BLAS Level 2 API.

hemvStridedBatched performs one of the matrix-vector operations

$$y_i := \alpha * A_i * x_i + \beta * y_i$$

where alpha and beta are scalars,  $x_i$  and  $y_i$  are n element vectors and  $A_i$  is an n by n Hermitian matrix, for each batch in  $i = [1, \text{batchCount}]$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER: the upper triangular part of the Hermitian matrix A is supplied. HIPBLAS\_FILL\_MODE\_LOWER: the lower triangular part of the Hermitian matrix A is supplied.
- **n** – [in] [int] the order of each matrix  $A_i$ .
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device array of device pointers storing each matrix  $A_i$  of dimension (lda, n). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  must contain the upper triangular part of a Hermitian matrix. The lower triangular part of each  $A_i$  will not be referenced. if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  must contain the lower triangular part of a Hermitian matrix. The upper triangular part of each  $A_i$  will not be referenced. As a Hermitian matrix, the imaginary part of the main diagonal of each  $A_i$  will not be referenced and is assumed to be == 0.
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ . must be  $\geq \max(1, n)$
- **strideA** – [in] [*hipblasStride*] stride from the start of one ( $A_i$ ) to the next ( $A_{i+1}$ )
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .

- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ).
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device array of device pointers storing each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of  $y$ .
- **stridey** – [in] [hipblasStride] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ ).
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.2.6 hipblasXher + Batched, StridedBatched

*hipblasStatus\_t* **hipblasCher**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const *hipblasComplex* \*x, int incx, *hipblasComplex* \*AP, int lda)

*hipblasStatus\_t* **hipblasZher**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*AP, int lda)

BLAS Level 2 API.

her performs the matrix-vector operations

$$A := A + \alpha * x * x^* * H$$

where alpha is a real scalar, x is a vector, and A is an n by n Hermitian matrix.

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of A is supplied in A. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of A is supplied in A.
- **n** – [in] [int] the number of rows and columns of matrix A, must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **AP** – [inout] device pointer storing the specified triangular portion of the Hermitian matrix A. Of size (lda \* n). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of the Hermitian matrix A is supplied. The lower triangular portion will not be touched. if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of the Hermitian matrix A is supplied. The upper triangular portion will not be touched. Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **lda** – [in] [int] specifies the leading dimension of A. Must be at least max(1, n).

*hipblasStatus\_t* **hipblasCherBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const *hipblasComplex* \*const x[], int incx, *hipblasComplex* \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasZherBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const *hipblasDoubleComplex* \*const x[], int incx, *hipblasDoubleComplex* \*const AP[], int lda, int batchCount)

BLAS Level 2 API.

herBatched performs the matrix-vector operations

$$A_i := A_i + \alpha x_i x_i^* H$$

where alpha is a real scalar,  $x_i$  is a vector, and  $A_i$  is an  $n$  by  $n$  symmetric matrix, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is supplied in A. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is supplied in A.
- **n** – [in] [int] the number of rows and columns of each matrix  $A_i$ , must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **AP** – [inout] device array of device pointers storing the specified triangular portion of each Hermitian matrix  $A_i$  of at least size  $((n * (n + 1)) / 2)$ . Array is of at least size batchCount. if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The lower triangular portion of each  $A_i$  will not be touched. if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The upper triangular portion of each  $A_i$  will not be touched. Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ . Must be at least  $\max(1, n)$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasCherStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasZherStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

BLAS Level 2 API.

herStridedBatched performs the matrix-vector operations

$$A_i := A_i + \alpha x_i x_i^* H$$

where  $\alpha$  is a real scalar,  $x_i$  is a vector, and  $A_i$  is an  $n$  by  $n$  Hermitian matrix, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is supplied in A. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is supplied in A.
- **n** – [in] [int] the number of rows and columns of each matrix  $A_i$ , must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$ .
- **x** – [in] device pointer pointing to the first vector ( $x_1$ ).
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ).
- **AP** – [inout] device array of device pointers storing the specified triangular portion of each Hermitian matrix  $A_i$ . Points to the first matrix ( $A_1$ ). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The lower triangular portion of each  $A_i$  will not be touched. if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The upper triangular portion of each  $A_i$  will not be touched. Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .
- **strideA** – [in] [hipblasStride] stride from the start of one ( $A_i$ ) and the next ( $A_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch.

#### 1.8.2.7 hipblasXher2 + Batched, StridedBatched

*hipblasStatus\_t* **hipblasCher2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*y, int incy, *hipblasComplex* \*AP, int lda)

*hipblasStatus\_t* **hipblasZher2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*y, int incy, *hipblasDoubleComplex* \*AP, int lda)

BLAS Level 2 API.

her2 performs the matrix-vector operations

$$A := A + \alpha x y^* H + \text{conj}(\alpha) y x^* H$$

where alpha is a complex scalar, x and y are vectors, and A is an n by n Hermitian matrix.

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of A is supplied. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of A is supplied.
- **n** – [in] [int] the number of rows and columns of matrix A, must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [in] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **AP** – [inout] device pointer storing the specified triangular portion of the Hermitian matrix A. Of size (lda, n). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of the Hermitian matrix A is supplied. The lower triangular portion of A will not be touched. if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of the Hermitian matrix A is supplied. The upper triangular portion of A will not be touched. Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **lda** – [in] [int] specifies the leading dimension of A. Must be at least max(lda, 1).

*hipblasStatus\_t* **hipblasCher2Batched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*const y[], int incy, *hipblasComplex* \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasZher2Batched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*const y[], int incy, *hipblasDoubleComplex* \*const AP[], int lda, int batchCount)

BLAS Level 2 API.

her2Batched performs the matrix-vector operations

$$A_i := A_i + \alpha x_i y_i^* H + \text{conj}(\alpha) y_i x_i^* H$$

where alpha is a complex scalar, x<sub>i</sub> and y<sub>i</sub> are vectors, and A<sub>i</sub> is an n by n Hermitian matrix for each batch in i = [1, batchCount].

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is supplied. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is supplied.
- **n** – [in] [int] the number of rows and columns of each matrix  $A_i$ , must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of  $x$ .
- **y** – [in] device array of device pointers storing each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **AP** – [inout] device array of device pointers storing the specified triangular portion of each Hermitian matrix  $A_i$  of size (lda, n). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The lower triangular portion of each  $A_i$  will not be touched. if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The upper triangular portion of each  $A_i$  will not be touched. Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ . Must be at least  $\max(\text{lda}, 1)$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasCher2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*y, int incy, *hipblasStride* stridey, *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasZher2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

BLAS Level 2 API.

her2StridedBatched performs the matrix-vector operations

$$A_i := A_i + \alpha x_i y_i^H + \text{conj}(\alpha) y_i x_i^H$$

where alpha is a complex scalar,  $x_i$  and  $y_i$  are vectors, and  $A_i$  is an  $n$  by  $n$  Hermitian matrix for each batch in  $i = [1, \text{batchCount}]$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.



- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is supplied. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is supplied.
- **n** – [in] [int] the number of rows and columns of each matrix  $A_i$ , must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer pointing to the first vector  $x_1$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] specifies the stride between the beginning of one vector ( $x_i$ ) and the next ( $x_{i+1}$ ).
- **y** – [in] device pointer pointing to the first vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **stridey** – [in] [hipblasStride] specifies the stride between the beginning of one vector ( $y_i$ ) and the next ( $y_{i+1}$ ).
- **AP** – [inout] device pointer pointing to the first matrix ( $A_1$ ). Stores the specified triangular portion of each Hermitian matrix  $A_i$ . if `uplo == HIPBLAS_FILL_MODE_UPPER`: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The lower triangular portion of each  $A_i$  will not be touched. if `uplo == HIPBLAS_FILL_MODE_LOWER`: The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The upper triangular portion of each  $A_i$  will not be touched. Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ . Must be at least  $\max(\text{lda}, 1)$ .
- **strideA** – [in] [hipblasStride] specifies the stride between the beginning of one matrix ( $A_i$ ) and the next ( $A_{i+1}$ ).
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.2.8 hipblasXhpmv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasChpmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy)

*hipblasStatus\_t* **hipblasZhpmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*y, int incy)

BLAS Level 2 API.

hpmv performs the matrix-vector operation

$$y := \alpha * A * x + \beta * y$$

where alpha and beta are scalars, x and y are n element vectors and A is an n by n Hermitian matrix, supplied in packed form (see description below).

- Supported precisions in rocBLAS : c,z

- Supported precisions in cuBLAS : c,z

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: the upper triangular part of the Hermitian matrix A is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: the lower triangular part of the Hermitian matrix A is supplied in AP.
- **n** – [in] [int] the order of the matrix A, must be  $\geq 0$ .
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device pointer storing the packed version of the specified triangular portion of the Hermitian matrix A. Of at least size  $((n * (n + 1)) / 2)$ . if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of the Hermitian matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 3) (1, 0) (2, 1) (3, 2) (2,-1) (4, 0) (5,-1)  $\rightarrow$  [(1,0), (2,1), (4,0), (3,2), (5,-1), (6,0)] (3,-2) (5, 1) (6, 0) if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of the Hermitian matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(2) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 3) (1, 0) (2, 1) (3, 2) (2,-1) (4, 0) (5,-1)  $\rightarrow$  [(1,0), (2,-1), (3,-2), (4,0), (5,1), (6,0)] (3,-2) (5, 1) (6, 0) Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.

*hipblasStatus\_t* **hipblasChpmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*const y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasZhpmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const y[], int incy, int batchCount)

BLAS Level 2 API.

hpmvBatched performs the matrix-vector operation

$$y_i := \alpha * A_i * x_i + \beta * y_i$$

where alpha and beta are scalars, x\_i and y\_i are n element vectors and A\_i is an n by n Hermitian matrix, supplied in packed form (see description below), for each batch in  $i = [1, \text{batchCount}]$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

## Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: the upper triangular part of each Hermitian matrix  $A_i$  is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: the lower triangular part of each Hermitian matrix  $A_i$  is supplied in AP.
- **n** – [in] [int] the order of each matrix  $A_i$ .
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device pointer of device pointers storing the packed version of the specified triangular portion of each Hermitian matrix  $A_i$ . Each  $A_i$  is of at least size  $((n * (n + 1)) / 2)$ . if `uplo == HIPBLAS_FILL_MODE_UPPER`: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that each  $AP_i$  contains the triangular portion column-by-column so that:  $AP(0) = A(0,0)$   $AP(1) = A(0,1)$   $AP(2) = A(1,1)$ , etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER;  $n = 3$ ) (1, 0) (2, 1) (3, 2) (2,-1) (4, 0) (5,-1)  $\rightarrow$  [(1,0), (2,1), (4,0), (3,2), (5,-1), (6,0)] (3,-2) (5, 1) (6, 0) if `uplo == HIPBLAS_FILL_MODE_LOWER`: The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that each  $AP_i$  contains the triangular portion column-by-column so that:  $AP(0) = A(0,0)$   $AP(1) = A(1,0)$   $AP(2) = A(2,1)$ , etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER;  $n = 3$ ) (1, 0) (2, 1) (3, 2) (2,-1) (4, 0) (5,-1)  $\rightarrow$  [(1,0), (2,-1), (3,-2), (4,0), (5,1), (6,0)] (3,-2) (5, 1) (6, 0) Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device array of device pointers storing each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of  $y$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasChpmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, *hipblasStride* strideA, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasZhpmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, *hipblasStride* strideA, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, int batchCount)

BLAS Level 2 API.

hpmvStridedBatched performs the matrix-vector operation

$$y_i := \alpha A_i x_i + \beta y_i$$

where alpha and beta are scalars,  $x_i$  and  $y_i$  are  $n$  element vectors and  $A_i$  is an  $n$  by  $n$  Hermitian matrix, supplied in packed form (see description below), for each batch in  $i = [1, \text{batchCount}]$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: the upper triangular part of each Hermitian matrix  $A_i$  is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: the lower triangular part of each Hermitian matrix  $A_i$  is supplied in AP.
- **n** – [in] [int] the order of each matrix  $A_i$ .
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **AP** – [in] device pointer pointing to the beginning of the first matrix (AP\_1). Stores the packed version of the specified triangular portion of each Hermitian matrix AP\_i of size  $((n * (n + 1)) / 2)$ . if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that each AP\_i contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 3) (1, 0) (2, 1) (3, 2) (2,-1) (4, 0) (5,-1) → [(1,0), (2,1), (4,0), (3,2), (5,-1), (6,0)] (3,-2) (5, 1) (6, 0) if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that each AP\_i contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(2) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 3) (1, 0) (2, 1) (3, 2) (2,-1) (4, 0) (5,-1) → [(1,0), (2,-1), (3,-2), (4,0), (5,1), (6,0)] (3,-2) (5, 1) (6, 0) Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (AP\_i) and the next one (AP\_i+1).
- **x** – [in] device array pointing to the beginning of the first vector (x\_1).
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector (x\_i) and the next one (x\_i+1).
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device array pointing to the beginning of the first vector (y\_1).
- **incy** – [in] [int] specifies the increment for the elements of y.
- **stridey** – [in] [hipblasStride] stride from the start of one vector (y\_i) and the next one (y\_i+1).
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.2.9 hipblasXhpr + Batched, StridedBatched

*hipblasStatus\_t* **hipblasChpr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const *hipblasComplex* \*x, int incx, *hipblasComplex* \*AP)

*hipblasStatus\_t* **hipblasZhpr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*AP)

BLAS Level 2 API.

hpr performs the matrix-vector operations

$$A := A + \alpha * x * x^* H$$

where alpha is a real scalar, x is a vector, and A is an n by n Hermitian matrix, supplied in packed form.

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of A is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of A is supplied in AP.
- **n** – [in] [int] the number of rows and columns of matrix A, must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **AP** – [inout] device pointer storing the packed version of the specified triangular portion of the Hermitian matrix A. Of at least size  $((n * (n + 1)) / 2)$ . if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of the Hermitian matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 3) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3) → [(1,0), (2,1), (3,0), (4,9), (5,3), (6,0)] (4,-9) (5,-3) (6,0) if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of the Hermitian matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(2) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 3) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3) → [(1,0), (2,-1), (4,-9), (3,0), (5,-3), (6,0)] (4,-9) (5,-3) (6,0) Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.

*hipblasStatus\_t* **hipblasChprBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const *hipblasComplex* \*const x[], int incx, *hipblasComplex* \*const AP[], int batchSize)

*hipblasStatus\_t* **hipblasZhprBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const *hipblasDoubleComplex* \*const x[], int incx, *hipblasDoubleComplex* \*const AP[], int batchCount)

BLAS Level 2 API.

hprBatched performs the matrix-vector operations

$$A_i := A_i + \alpha x_i x_i^* H$$

where alpha is a real scalar,  $x_i$  is a vector, and  $A_i$  is an  $n$  by  $n$  symmetric matrix, supplied in packed form, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is supplied in AP.
- **n** – [in] [int] the number of rows and columns of each matrix  $A_i$ , must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **AP** – [inout] device array of device pointers storing the packed version of the specified triangular portion of each Hermitian matrix  $A_i$  of at least size  $((n * (n + 1)) / 2)$ . Array is of at least size batchCount. if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 3) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3) → [(1,0), (2,1), (3,0), (4,9), (5,3), (6,0)] (4,-9) (5,-3) (6,0) if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(2) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 3) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3) → [(1,0), (2,-1), (4,-9), (3,0), (5,-3), (6,0)] (4,-9) (5,-3) (6,0) Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasChprStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, *hipblasComplex* \*AP, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasZhprStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, *hipblasDoubleComplex* \*AP, *hipblasStride* strideA, int batchCount)

BLAS Level 2 API.

hprStridedBatched performs the matrix-vector operations

$$A_i := A_i + \alpha x_i x_i^* H$$

where alpha is a real scalar,  $x_i$  is a vector, and  $A_i$  is an  $n$  by  $n$  symmetric matrix, supplied in packed form, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is supplied in AP.
- **n** – [in] [int] the number of rows and columns of each matrix  $A_i$ , must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer pointing to the first vector ( $x_1$ ).
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [*hipblasStride*] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ).
- **AP** – [inout] device array of device pointers storing the packed version of the specified triangular portion of each Hermitian matrix  $A_i$ . Points to the first matrix ( $A_1$ ). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that:  $AP(0) = A(0,0)$   $AP(1) = A(0,1)$   $AP(2) = A(1,1)$ , etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER;  $n = 3$ ) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3)  $\rightarrow$  [(1,0), (2,1), (3,0), (4,9), (5,3), (6,0)] (4,-9) (5,-3) (6,0) if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that:  $AP(0) = A(0,0)$   $AP(1) = A(1,0)$   $AP(2) = A(2,1)$ , etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER;  $n = 3$ ) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3)  $\rightarrow$  [(1,0), (2,-1), (4,-9), (3,0), (5,-3), (6,0)] (4,-9) (5,-3) (6,0) Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **strideA** – [in] [*hipblasStride*] stride from the start of one ( $A_i$ ) and the next ( $A_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.2.10 hipblasXhpr2 + Batched, StridedBatched

*hipblasStatus\_t* **hipblasChpr2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*y, int incy, *hipblasComplex* \*AP)

*hipblasStatus\_t* **hipblasZhpr2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*y, int incy, *hipblasDoubleComplex* \*AP)

BLAS Level 2 API.

hpr2 performs the matrix-vector operations

$$A := A + \alpha * x * y^{**H} + \text{conj}(\alpha) * y * x^{**H}$$

where alpha is a complex scalar, x and y are vectors, and A is an n by n Hermitian matrix, supplied in packed form.

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of A is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of A is supplied in AP.
- **n** – [in] [int] the number of rows and columns of matrix A, must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [in] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **AP** – [inout] device pointer storing the packed version of the specified triangular portion of the Hermitian matrix A. Of at least size  $((n * (n + 1)) / 2)$ . if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of the Hermitian matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 3) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3) → [(1,0), (2,1), (3,0), (4,9), (5,3), (6,0)] (4,-9) (5,-3) (6,0) if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of the Hermitian matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(2) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 3) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3) → [(1,0), (2,-1), (4,-9), (3,0), (5,-3), (6,0)] (4,-9) (5,-3) (6,0) Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.



```
hipblasStatus_t hipblasChpr2Batched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
hipblasComplex *alpha, const hipblasComplex *const x[], int incx, const
hipblasComplex *const y[], int incy, hipblasComplex *const AP[], int
batchCount)
```

```
hipblasStatus_t hipblasZhpr2Batched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
hipblasDoubleComplex *alpha, const hipblasDoubleComplex *const x[],
int incx, const hipblasDoubleComplex *const y[], int incy,
hipblasDoubleComplex *const AP[], int batchCount)
```

BLAS Level 2 API.

hpr2Batched performs the matrix-vector operations

$$A_i := A_i + \alpha x_i y_i^* H + \text{conj}(\alpha) y_i x_i^* H$$

where  $\alpha$  is a complex scalar,  $x_i$  and  $y_i$  are vectors, and  $A_i$  is an  $n$  by  $n$  symmetric matrix, supplied in packed form, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is supplied in AP.
- **n** – [in] [int] the number of rows and columns of each matrix  $A_i$ , must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$ .
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **y** – [in] device array of device pointers storing each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **AP** – [inout] device array of device pointers storing the packed version of the specified triangular portion of each Hermitian matrix  $A_i$  of at least size  $((n * (n + 1)) / 2)$ . Array is of at least size batchCount. if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 3) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3) → [(1,0), (2,1), (3,0), (4,9), (5,3), (6,0)] (4,-9) (5,-3) (6,0) if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(2) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 3) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3) → [(1,0), (2,-1), (4,-9), (3,0), (5,-3), (6,0)] (4,-9) (5,-3) (6,0) Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasChpr2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*y, int incy, *hipblasStride* stridey, *hipblasComplex* \*AP, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasZhpr2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, *hipblasDoubleComplex* \*AP, *hipblasStride* strideA, int batchCount)

BLAS Level 2 API.

hpr2StridedBatched performs the matrix-vector operations

$$A_i := A_i + \alpha * x_i * y_i^{*H} + \text{conj}(\alpha) * y_i * x_i^{*H}$$

where alpha is a complex scalar,  $x_i$  and  $y_i$  are vectors, and  $A_i$  is an  $n$  by  $n$  symmetric matrix, supplied in packed form, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is supplied in AP.
- **n** – [in] [int] the number of rows and columns of each matrix  $A_i$ , must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer pointing to the first vector ( $x_1$ ).
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [*hipblasStride*] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ).
- **y** – [in] device pointer pointing to the first vector ( $y_1$ ).
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **stridey** – [in] [*hipblasStride*] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ ).
- **AP** – [inout] device array of device pointers storing the packed version of the specified triangular portion of each Hermitian matrix  $A_i$ . Points to the first matrix ( $A_1$ ). if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that:  $AP(0) = A(0,0)$   $AP(1) = A(0,1)$   $AP(2) = A(1,1)$ , etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER;  $n = 3$ ) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3) → [(1,0), (2,1), (3,0), (4,9), (5,3), (6,0)] (4,-9) (5,-3) (6,0) if uplo == HIPBLAS\_FILL\_MODE\_LOWER:

The lower triangular portion of each Hermitian matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that:  $AP(0) = A(0,0)$   $AP(1) = A(1,0)$   $AP(2) = A(2,1)$ , etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER;  $n = 3$ ) (1, 0) (2, 1) (4,9) (2,-1) (3, 0) (5,3)  $\rightarrow$  [(1,0), (2,-1), (4,-9), (3,0), (5,-3), (6,0)] (4,-9) (5,-3) (6,0) Note that the imaginary part of the diagonal elements are not accessed and are assumed to be 0.

- **strideA** – [in] [hipblasStride] stride from the start of one ( $A_i$ ) and the next ( $A_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.2.11 hipblasXsbmv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSsbmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const float \*alpha, const float \*AP, int lda, const float \*x, int incx, const float \*beta, float \*y, int incy)

*hipblasStatus\_t* **hipblasDsbmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const double \*alpha, const double \*AP, int lda, const double \*x, int incx, const double \*beta, double \*y, int incy)

BLAS Level 2 API.

sbmv performs the matrix-vector operation:

$$y := \alpha * A * x + \beta * y,$$

where alpha and beta are scalars, x and y are n element vectors and A should contain an upper or lower triangular n by n symmetric banded matrix.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : s,d

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int]
- **k** – [in] [int] specifies the number of sub- and super-diagonals
- **alpha** – [in] specifies the scalar alpha
- **AP** – [in] pointer storing matrix A on the GPU
- **lda** – [in] [int] specifies the leading dimension of matrix A
- **x** – [in] pointer storing vector x on the GPU
- **incx** – [in] [int] specifies the increment for the elements of x
- **beta** – [in] specifies the scalar beta
- **y** – [out] pointer storing vector y on the GPU
- **incy** – [in] [int] specifies the increment for the elements of y

*hipblasStatus\_t* **hipblasSsbmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const float \*alpha, const float \*const AP[], int lda, const float \*const x[], int incx, const float \*beta, float \*y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasDsbmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const double \*alpha, const double \*const AP[], int lda, const double \*const x[], int incx, const double \*beta, double \*y[], int incy, int batchCount)

BLAS Level 2 API.

sbmvBatched performs the matrix-vector operation:

$$y_i := \alpha A_i x_i + \beta y_i,$$

where  $(A_i, x_i, y_i)$  is the  $i$ -th instance of the batch.  $\alpha$  and  $\beta$  are scalars,  $x_i$  and  $y_i$  are vectors and  $A_i$  is an  $n$  by  $n$  symmetric banded matrix, for  $i = 1, \dots, \text{batchCount}$ .  $A$  should contain an upper or lower triangular  $n$  by  $n$  symmetric banded matrix.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of  $A$  is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of  $A$  is not referenced
- **n** – [in] [int] number of rows and columns of each matrix  $A_i$
- **k** – [in] [int] specifies the number of sub- and super-diagonals
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$
- **AP** – [in] device array of device pointers storing each matrix  $A_i$
- **lda** – [in] [int] specifies the leading dimension of each matrix  $A_i$
- **x** – [in] device array of device pointers storing each vector  $x_i$
- **incx** – [in] [int] specifies the increment for the elements of each vector  $x_i$
- **beta** – [in] device pointer or host pointer to scalar  $\beta$
- **y** – [out] device array of device pointers storing each vector  $y_i$
- **incy** – [in] [int] specifies the increment for the elements of each vector  $y_i$
- **batchCount** – [in] [int] number of instances in the batch

*hipblasStatus\_t* **hipblasSsbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const float \*alpha, const float \*AP, int lda, *hipblasStride* strideA, const float \*x, int incx, *hipblasStride* stridex, const float \*beta, float \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasDsbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, int k, const double \*alpha, const double \*AP, int lda, *hipblasStride* strideA, const double \*x, int incx, *hipblasStride* stridex, const double \*beta, double \*y, int incy, *hipblasStride* stridey, int batchCount)

BLAS Level 2 API.

sbmvStridedBatched performs the matrix-vector operation:

$$y_i := \alpha * A_i * x_i + \beta * y_i,$$

where  $(A_i, x_i, y_i)$  is the  $i$ -th instance of the batch.  $\alpha$  and  $\beta$  are scalars,  $x_i$  and  $y_i$  are vectors and  $A_i$  is an  $n$  by  $n$  symmetric banded matrix, for  $i = 1, \dots, \text{batchCount}$ .  $A$  should contain an upper or lower triangular  $n$  by  $n$  symmetric banded matrix.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of  $A$  is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of  $A$  is not referenced
- **n** – [in] [int] number of rows and columns of each matrix  $A_i$
- **k** – [in] [int] specifies the number of sub- and super-diagonals
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$
- **AP** – [in] Device pointer to the first matrix  $A_1$  on the GPU
- **lda** – [in] [int] specifies the leading dimension of each matrix  $A_i$
- **strideA** – [in] [*hipblasStride*] stride from the start of one matrix ( $A_i$ ) and the next one ( $A_{i+1}$ )
- **x** – [in] Device pointer to the first vector  $x_1$  on the GPU
- **incx** – [in] [int] specifies the increment for the elements of each vector  $x_i$
- **stridex** – [in] [*hipblasStride*] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ). There are no restrictions placed on stridex, however the user should take care to ensure that stridex is of appropriate size. This typically means  $\text{stridex} \geq n * \text{incx}$ . stridex should be non zero.
- **beta** – [in] device pointer or host pointer to scalar  $\beta$
- **y** – [out] Device pointer to the first vector  $y_1$  on the GPU
- **incy** – [in] [int] specifies the increment for the elements of each vector  $y_i$
- **stridey** – [in] [*hipblasStride*] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ ). There are no restrictions placed on stridey, however the user should take care to ensure that stridey is of appropriate size. This typically means  $\text{stridey} \geq n * \text{incy}$ . stridey should be non zero.
- **batchCount** – [in] [int] number of instances in the batch

### 1.8.2.12 hipblasXspmv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSspmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*AP, const float \*x, int incx, const float \*beta, float \*y, int incy)

*hipblasStatus\_t* **hipblasDspmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*AP, const double \*x, int incx, const double \*beta, double \*y, int incy)

BLAS Level 2 API.

spmv performs the matrix-vector operation:

$$y := \alpha * A * x + \beta * y,$$

where alpha and beta are scalars, x and y are n element vectors and A should contain an upper or lower triangular n by n packed symmetric matrix.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : s,d

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int]
- **alpha** – [in] specifies the scalar alpha
- **AP** – [in] pointer storing matrix A on the GPU
- **x** – [in] pointer storing vector x on the GPU
- **incx** – [in] [int] specifies the increment for the elements of x
- **beta** – [in] specifies the scalar beta
- **y** – [out] pointer storing vector y on the GPU
- **incy** – [in] [int] specifies the increment for the elements of y

*hipblasStatus\_t* **hipblasSspmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*const AP[], const float \*const x[], int incx, const float \*beta, float \*y[], int incy, int batchCount)

*hipblasStatus\_t* **hipblasDspmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*const AP[], const double \*const x[], int incx, const double \*beta, double \*y[], int incy, int batchCount)

BLAS Level 2 API.

spmvBatched performs the matrix-vector operation:

$$y\_i := \alpha * AP\_i * x\_i + \beta * y\_i,$$

where ( $A_i$ ,  $x_i$ ,  $y_i$ ) is the  $i$ -th instance of the batch.  $\alpha$  and  $\beta$  are scalars,  $x_i$  and  $y_i$  are vectors and  $A_i$  is an  $n$  by  $n$  symmetric matrix, for  $i = 1, \dots, \text{batchCount}$ .  $A$  should contain an upper or lower triangular  $n$  by  $n$  packed symmetric matrix.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of  $A$  is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of  $A$  is not referenced
- **n** – [in] [int] number of rows and columns of each matrix  $A_i$
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$
- **AP** – [in] device array of device pointers storing each matrix  $A_i$
- **x** – [in] device array of device pointers storing each vector  $x_i$
- **incx** – [in] [int] specifies the increment for the elements of each vector  $x_i$
- **beta** – [in] device pointer or host pointer to scalar  $\beta$
- **y** – [out] device array of device pointers storing each vector  $y_i$
- **incy** – [in] [int] specifies the increment for the elements of each vector  $y_i$
- **batchCount** – [in] [int] number of instances in the batch

*hipblasStatus\_t* **hipblasSspmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*AP, *hipblasStride* strideA, const float \*x, int incx, *hipblasStride* stridex, const float \*beta, float \*y, int incy, *hipblasStride* stridey, int batchCount)

*hipblasStatus\_t* **hipblasDspmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*AP, *hipblasStride* strideA, const double \*x, int incx, *hipblasStride* stridex, const double \*beta, double \*y, int incy, *hipblasStride* stridey, int batchCount)

BLAS Level 2 API.

spmvStridedBatched performs the matrix-vector operation:

$$y_i := \alpha A_i x_i + \beta y_i,$$

where ( $A_i$ ,  $x_i$ ,  $y_i$ ) is the  $i$ -th instance of the batch.  $\alpha$  and  $\beta$  are scalars,  $x_i$  and  $y_i$  are vectors and  $A_i$  is an  $n$  by  $n$  symmetric matrix, for  $i = 1, \dots, \text{batchCount}$ .  $A$  should contain an upper or lower triangular  $n$  by  $n$  packed symmetric matrix.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int] number of rows and columns of each matrix A<sub>i</sub>
- **alpha** – [in] device pointer or host pointer to scalar alpha
- **AP** – [in] Device pointer to the first matrix A<sub>1</sub> on the GPU
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A<sub>i</sub>) and the next one (A<sub>i+1</sub>)
- **x** – [in] Device pointer to the first vector x<sub>1</sub> on the GPU
- **incx** – [in] [int] specifies the increment for the elements of each vector x<sub>i</sub>
- **stridex** – [in] [hipblasStride] stride from the start of one vector (x<sub>i</sub>) and the next one (x<sub>i+1</sub>). There are no restrictions placed on stridex, however the user should take care to ensure that stridex is of appropriate size. This typically means stridex ≥ n \* incx. stridex should be non zero.
- **beta** – [in] device pointer or host pointer to scalar beta
- **y** – [out] Device pointer to the first vector y<sub>1</sub> on the GPU
- **incy** – [in] [int] specifies the increment for the elements of each vector y<sub>i</sub>
- **stridey** – [in] [hipblasStride] stride from the start of one vector (y<sub>i</sub>) and the next one (y<sub>i+1</sub>). There are no restrictions placed on stridey, however the user should take care to ensure that stridey is of appropriate size. This typically means stridey ≥ n \* incy. stridey should be non zero.
- **batchCount** – [in] [int] number of instances in the batch

### 1.8.2.13 hipblasXspr + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSspr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*x, int incx, float \*AP)

*hipblasStatus\_t* **hipblasDspr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*x, int incx, double \*AP)

*hipblasStatus\_t* **hipblasCspr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasComplex* \*AP)

*hipblasStatus\_t* **hipblasZspr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*AP)

BLAS Level 2 API.

spr performs the matrix-vector operations

$$A := A + \alpha * x * x^T$$



where alpha is a scalar, x is a vector, and A is an n by n symmetric matrix, supplied in packed form.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of A is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of A is supplied in AP.
- **n** – [in] [int] the number of rows and columns of matrix A, must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **AP** – [inout] device pointer storing the packed version of the specified triangular portion of the symmetric matrix A. Of at least size  $((n * (n + 1)) / 2)$ . if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of the symmetric matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 4) 1 2 4 7 2 3 5 8 → [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 4 5 6 9 7 8 9 0 if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of the symmetric matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(2) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 4) 1 2 3 4 2 5 6 7 → [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 3 6 8 9 4 7 9 0

*hipblasStatus\_t* **hipblasSsprBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*const x[], int incx, float \*const AP[], int batchCount)

*hipblasStatus\_t* **hipblasDsprBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*const x[], int incx, double \*const AP[], int batchCount)

*hipblasStatus\_t* **hipblasCsprBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const x[], int incx, *hipblasComplex* \*const AP[], int batchCount)

*hipblasStatus\_t* **hipblasZsprBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const x[], int incx, *hipblasDoubleComplex* \*const AP[], int batchCount)

BLAS Level 2 API.

sprBatched performs the matrix-vector operations

$$A_i := A_i + \alpha x_i x_i^T$$

where alpha is a scalar, x<sub>i</sub> is a vector, and A<sub>i</sub> is an n by n symmetric matrix, supplied in packed form, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each A<sub>i</sub> is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each A<sub>i</sub> is supplied in AP.
- **n** – [in] [int] the number of rows and columns of each matrix A<sub>i</sub>, must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device array of device pointers storing each vector x<sub>i</sub>.
- **incx** – [in] [int] specifies the increment for the elements of each x<sub>i</sub>.
- **AP** – [inout] device array of device pointers storing the packed version of the specified triangular portion of each symmetric matrix A<sub>i</sub> of at least size  $((n * (n + 1)) / 2)$ . Array is of at least size batchCount. if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each symmetric matrix A<sub>i</sub> is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 4) 1 2 4 7 2 3 5 8 → [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 4 5 6 9 7 8 9 0 if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each symmetric matrix A<sub>i</sub> is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(2) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 4) 1 2 3 4 2 5 6 7 → [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 3 6 8 9 4 7 9 0
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasSsprStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*x, int incx, *hipblasStride* stridex, float \*AP, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasDsprStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*x, int incx, *hipblasStride* stridex, double \*AP, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasCsprStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, *hipblasComplex* \*AP, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasZsprStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, *hipblasDoubleComplex* \*AP, *hipblasStride* strideA, int batchCount)

BLAS Level 2 API.

sprStridedBatched performs the matrix-vector operations

$$A_i := A_i + \alpha x_i x_i^T$$

where  $\alpha$  is a scalar,  $x_i$  is a vector, and  $A_i$  is an  $n$  by  $n$  symmetric matrix, supplied in packed form, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each  $A_i$  is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each  $A_i$  is supplied in AP.
- **n** – [in] [int] the number of rows and columns of each matrix  $A_i$ , must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$ .
- **x** – [in] device pointer pointing to the first vector ( $x_1$ ).
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ).
- **AP** – [inout] device pointer storing the packed version of the specified triangular portion of each symmetric matrix  $A_i$ . Points to the first  $A_1$ . if `uplo == HIPBLAS_FILL_MODE_UPPER`: The upper triangular portion of each symmetric matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that:  $AP(0) = A(0,0)$   $AP(1) = A(0,1)$   $AP(2) = A(1,1)$ , etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER;  $n = 4$ ) 1 2 4 7 2 3 5 8  $\rightarrow$  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 4 5 6 9 7 8 9 0 if `uplo == HIPBLAS_FILL_MODE_LOWER`: The lower triangular portion of each symmetric matrix  $A_i$  is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that:  $AP(0) = A(0,0)$   $AP(1) = A(1,0)$   $AP(2) = A(2,1)$ , etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER;  $n = 4$ ) 1 2 3 4 2 5 6 7  $\rightarrow$  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 3 6 8 9 4 7 9 0
- **strideA** – [in] [hipblasStride] stride from the start of one ( $A_i$ ) and the next ( $A_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch.

#### 1.8.2.14 hipblasXspr2 + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSspr2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*x, int incx, const float \*y, int incy, float \*AP)

*hipblasStatus\_t* **hipblasDspr2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*x, int incx, const double \*y, int incy, double \*AP)

BLAS Level 2 API.

spr2 performs the matrix-vector operation

$$A := A + \alpha * x * y^{**T} + \alpha * y * x^{**T}$$

where alpha is a scalar, x and y are vectors, and A is an n by n symmetric matrix, supplied in packed form.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : s,d

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of A is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of A is supplied in AP.
- **n** – [in] [int] the number of rows and columns of matrix A, must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [in] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **AP** – [inout] device pointer storing the packed version of the specified triangular portion of the symmetric matrix A. Of at least size  $((n * (n + 1)) / 2)$ . if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of the symmetric matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 4) 1 2 4 7 2 3 5 8 -> [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 4 5 6 9 7 8 9 0 if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of the symmetric matrix A is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(n) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 4) 1 2 3 4 2 5 6 7 -> [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 3 6 8 9 4 7 9 0

*hipblasStatus\_t* **hipblasSspr2Batched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*const x[], int incx, const float \*const y[], int incy, float \*const AP[], int batchCount)

*hipblasStatus\_t* **hipblasDspr2Batched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*const x[], int incx, const double \*const y[], int incy, double \*const AP[], int batchCount)

BLAS Level 2 API.

spr2Batched performs the matrix-vector operation

$$A_i := A_i + \alpha * x_i * y_i^{**T} + \alpha * y_i * x_i^{**T}$$

where alpha is a scalar, x\_i and y\_i are vectors, and A\_i is an n by n symmetric matrix, supplied in packed form, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d
- Supported precisions in cuBLAS : No support

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each A<sub>i</sub> is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each A<sub>i</sub> is supplied in AP.
- **n** – [in] [int] the number of rows and columns of each matrix A<sub>i</sub>, must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device array of device pointers storing each vector x<sub>i</sub>.
- **incx** – [in] [int] specifies the increment for the elements of each x<sub>i</sub>.
- **y** – [in] device array of device pointers storing each vector y<sub>i</sub>.
- **incy** – [in] [int] specifies the increment for the elements of each y<sub>i</sub>.
- **AP** – [inout] device array of device pointers storing the packed version of the specified triangular portion of each symmetric matrix A<sub>i</sub> of at least size  $((n * (n + 1)) / 2)$ . Array is of at least size batchCount. if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each symmetric matrix A<sub>i</sub> is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 4) 1 2 4 7 2 3 5 8 → [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 4 5 6 9 7 8 9 0 if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each symmetric matrix A<sub>i</sub> is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(n) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 4) 1 2 3 4 2 5 6 7 → [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 3 6 8 9 4 7 9 0
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasSspr2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*x, int incx, *hipblasStride* stridex, const float \*y, int incy, *hipblasStride* stridey, float \*AP, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasDspr2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*x, int incx, *hipblasStride* stridex, const double \*y, int incy, *hipblasStride* stridey, double \*AP, *hipblasStride* strideA, int batchCount)

BLAS Level 2 API.

spr2StridedBatched performs the matrix-vector operation

$$A_i := A_i + \alpha x_i y_i^{**T} + \alpha y_i x_i^{**T}$$

where alpha is a scalar, x<sub>i</sub> and y<sub>i</sub> are vectors, and A<sub>i</sub> is an n by n symmetric matrix, supplied in packed form, for i = 1, ..., batchCount.

- Supported precisions in rocBLAS : s,d

- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular part of each A<sub>i</sub> is supplied in AP. HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular part of each A<sub>i</sub> is supplied in AP.
- **n** – [in] [int] the number of rows and columns of each matrix A<sub>i</sub>, must be at least 0.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer pointing to the first vector (x<sub>1</sub>).
- **incx** – [in] [int] specifies the increment for the elements of each x<sub>i</sub>.
- **stridex** – [in] [hipblasStride] stride from the start of one vector (x<sub>i</sub>) and the next one (x<sub>i</sub>+1).
- **y** – [in] device pointer pointing to the first vector (y<sub>1</sub>).
- **incy** – [in] [int] specifies the increment for the elements of each y<sub>i</sub>.
- **stridey** – [in] [hipblasStride] stride from the start of one vector (y<sub>i</sub>) and the next one (y<sub>i</sub>+1).
- **AP** – [inout] device pointer storing the packed version of the specified triangular portion of each symmetric matrix A<sub>i</sub>. Points to the first A<sub>1</sub>. if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The upper triangular portion of each symmetric matrix A<sub>i</sub> is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(0,1) AP(2) = A(1,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; n = 4) 1 2 4 7 2 3 5 8 → [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 4 5 6 9 7 8 9 0 if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The lower triangular portion of each symmetric matrix A<sub>i</sub> is supplied. The matrix is compacted so that AP contains the triangular portion column-by-column so that: AP(0) = A(0,0) AP(1) = A(1,0) AP(n) = A(2,1), etc. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; n = 4) 1 2 3 4 2 5 6 7 → [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] 3 6 8 9 4 7 9 0
- **strideA** – [in] [hipblasStride] stride from the start of one (A<sub>i</sub>) and the next (A<sub>i</sub>+1)
- **batchCount** – [in] [int] number of instances in the batch.

#### 1.8.2.15 hipblasXsymv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSsymv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*AP, int lda, const float \*x, int incx, const float \*beta, float \*y, int incy)

*hipblasStatus\_t* **hipblasDsymv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*AP, int lda, const double \*x, int incx, const double \*beta, double \*y, int incy)

*hipblasStatus\_t* **hipblasCsymv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*beta, *hipblasComplex* \*y, int incy)

```
hipblasStatus_t hipblasZsymv(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
                             hipblasDoubleComplex *alpha, const hipblasDoubleComplex *AP, int lda, const
                             hipblasDoubleComplex *x, int incx, const hipblasDoubleComplex *beta,
                             hipblasDoubleComplex *y, int incy)
```

BLAS Level 2 API.

symv performs the matrix-vector operation:

$$y := \alpha * A * x + \beta * y,$$

where alpha and beta are scalars, x and y are n element vectors and A should contain an upper or lower triangular n by n symmetric matrix.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int]
- **alpha** – [in] specifies the scalar alpha
- **AP** – [in] pointer storing matrix A on the GPU
- **lda** – [in] [int] specifies the leading dimension of A
- **x** – [in] pointer storing vector x on the GPU
- **incx** – [in] [int] specifies the increment for the elements of x
- **beta** – [in] specifies the scalar beta
- **y** – [out] pointer storing vector y on the GPU
- **incy** – [in] [int] specifies the increment for the elements of y

```
hipblasStatus_t hipblasSsymvBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const float *alpha,
                                     const float *const AP[], int lda, const float *const x[], int incx, const float
                                     *beta, float *y[], int incy, int batchSize)
```

```
hipblasStatus_t hipblasDsymvBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const double
                                     *alpha, const double *const AP[], int lda, const double *const x[], int incx,
                                     const double *beta, double *y[], int incy, int batchSize)
```

```
hipblasStatus_t hipblasCsymvBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
                                     hipblasComplex *alpha, const hipblasComplex *const AP[], int lda, const
                                     hipblasComplex *const x[], int incx, const hipblasComplex *beta,
                                     hipblasComplex *y[], int incy, int batchSize)
```



```
hipblasStatus_t hipblasZsymvBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
                                     hipblasDoubleComplex *alpha, const hipblasDoubleComplex *const AP[],
                                     int lda, const hipblasDoubleComplex *const x[], int incx, const
                                     hipblasDoubleComplex *beta, hipblasDoubleComplex *y[], int incy, int
                                     batchCount)
```

BLAS Level 2 API.

symvBatched performs the matrix-vector operation:

$$y_i := \alpha * A_i * x_i + \beta * y_i,$$

where  $(A_i, x_i, y_i)$  is the  $i$ -th instance of the batch.  $\alpha$  and  $\beta$  are scalars,  $x_i$  and  $y_i$  are vectors and  $A_i$  is an  $n$  by  $n$  symmetric matrix, for  $i = 1, \dots, \text{batchCount}$ .  $A$  should contain an upper or lower triangular symmetric matrix and the opposing triangular part of  $A$  is not referenced

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of  $A$  is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of  $A$  is not referenced
- **n** – [in] [int] number of rows and columns of each matrix  $A_i$
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$
- **AP** – [in] device array of device pointers storing each matrix  $A_i$
- **lda** – [in] [int] specifies the leading dimension of each matrix  $A_i$
- **x** – [in] device array of device pointers storing each vector  $x_i$
- **incx** – [in] [int] specifies the increment for the elements of each vector  $x_i$
- **beta** – [in] device pointer or host pointer to scalar  $\beta$
- **y** – [out] device array of device pointers storing each vector  $y_i$
- **incy** – [in] [int] specifies the increment for the elements of each vector  $y_i$
- **batchCount** – [in] [int] number of instances in the batch

```
hipblasStatus_t hipblasSsymvStridedBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
                                             float *alpha, const float *AP, int lda, hipblasStride strideA, const
                                             float *x, int incx, hipblasStride stridex, const float *beta, float *y,
                                             int incy, hipblasStride stridey, int batchCount)
```

```
hipblasStatus_t hipblasDsymvStridedBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
                                             double *alpha, const double *AP, int lda, hipblasStride strideA,
                                             const double *x, int incx, hipblasStride stridex, const double
                                             *beta, double *y, int incy, hipblasStride stridey, int batchCount)
```



```
hipblasStatus_t hipblasCsymvStridedBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
hipblasComplex *alpha, const hipblasComplex *AP, int lda,
hipblasStride strideA, const hipblasComplex *x, int incx,
hipblasStride stridex, const hipblasComplex *beta,
hipblasComplex *y, int incy, hipblasStride stridey, int
batchCount)
```

```
hipblasStatus_t hipblasZsymvStridedBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, int n, const
hipblasDoubleComplex *alpha, const hipblasDoubleComplex
*AP, int lda, hipblasStride strideA, const hipblasDoubleComplex
*x, int incx, hipblasStride stridex, const hipblasDoubleComplex
*beta, hipblasDoubleComplex *y, int incy, hipblasStride stridey,
int batchCount)
```

BLAS Level 2 API.

symvStridedBatched performs the matrix-vector operation:

$$y_i := \alpha * A_i * x_i + \beta * y_i,$$

where  $(A_i, x_i, y_i)$  is the  $i$ -th instance of the batch.  $\alpha$  and  $\beta$  are scalars,  $x_i$  and  $y_i$  are vectors and  $A_i$  is an  $n$  by  $n$  symmetric matrix, for  $i = 1, \dots, \text{batchCount}$ .  $A$  should contain an upper or lower triangular symmetric matrix and the opposing triangular part of  $A$  is not referenced

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of  $A$  is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of  $A$  is not referenced
- **n** – [in] [int] number of rows and columns of each matrix  $A_i$
- **alpha** – [in] device pointer or host pointer to scalar  $\alpha$
- **AP** – [in] Device pointer to the first matrix  $A_1$  on the GPU
- **lda** – [in] [int] specifies the leading dimension of each matrix  $A_i$
- **strideA** – [in] [hipblasStride] stride from the start of one matrix ( $A_i$ ) and the next one ( $A_{i+1}$ )
- **x** – [in] Device pointer to the first vector  $x_1$  on the GPU
- **incx** – [in] [int] specifies the increment for the elements of each vector  $x_i$
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ). There are no restrictions placed on stridex, however the user should take care to ensure that stridex is of appropriate size. This typically means  $\text{stridex} \geq n * \text{incx}$ . stridex should be non zero.
- **beta** – [in] device pointer or host pointer to scalar  $\beta$
- **y** – [out] Device pointer to the first vector  $y_1$  on the GPU

- **incy** – [in] [int] specifies the increment for the elements of each vector  $y_i$
- **stridey** – [in] [hipblasStride] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ ). There are no restrictions placed on stridey, however the user should take care to ensure that stridey is of appropriate size. This typically means  $\text{stridey} \geq n * \text{incy}$ . stridey should be non zero.
- **batchCount** – [in] [int] number of instances in the batch

#### 1.8.2.16 hipblasXsyr + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSsyr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*x, int incx, float \*AP, int lda)

*hipblasStatus\_t* **hipblasDsyr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*x, int incx, double \*AP, int lda)

*hipblasStatus\_t* **hipblasCsyr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasComplex* \*AP, int lda)

*hipblasStatus\_t* **hipblasZsyr**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*AP, int lda)

BLAS Level 2 API.

syr performs the matrix-vector operations

$$A := A + \alpha * x * x^T$$

where alpha is a scalar, x is a vector, and A is an n by n symmetric matrix.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

##### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int] the number of rows and columns of matrix A.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **AP** – [inout] device pointer storing matrix A.
- **lda** – [in] [int] specifies the leading dimension of A.

*hipblasStatus\_t* **hipblasSsyrBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*const x[], int incx, float \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasDsyrBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*const x[], int incx, double \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasCsyrBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const x[], int incx, *hipblasComplex* \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasZsyrBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const x[], int incx, *hipblasDoubleComplex* \*const AP[], int lda, int batchCount)

BLAS Level 2 API.

syrBatched performs a batch of matrix-vector operations

$$A[i] := A[i] + \alpha * x[i] * x[i]^T$$

where alpha is a scalar, x is an array of vectors, and A is an array of n by n symmetric matrices, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int] the number of rows and columns of matrix A.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **AP** – [inout] device array of device pointers storing each matrix  $A_i$ .
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .
- **batchCount** – [in] [int] number of instances in the batch

*hipblasStatus\_t* **hipblasSsyrStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*x, int incx, *hipblasStride* stridex, float \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasDsyrStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*x, int incx, *hipblasStride* stridex, double \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasCsyrStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasZsyrStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

BLAS Level 2 API.

syrStridedBatched performs the matrix-vector operations

$$A[i] := A[i] + \alpha * x[i] * x[i]**T$$

where alpha is a scalar, vectors, and A is an array of n by n symmetric matrices, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int] the number of rows and columns of each matrix A.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer to the first vector  $x_1$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [*hipblasStride*] specifies the pointer increment between vectors ( $x_i$ ) and ( $x_{i+1}$ ).
- **AP** – [inout] device pointer to the first matrix  $A_1$ .
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .
- **strideA** – [in] [*hipblasStride*] stride from the start of one matrix ( $A_i$ ) and the next one ( $A_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch

### 1.8.2.17 hipblasXsyr2 + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSsyr2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*x, int incx, const float \*y, int incy, float \*AP, int lda)

*hipblasStatus\_t* **hipblasDsyr2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*x, int incx, const double \*y, int incy, double \*AP, int lda)

*hipblasStatus\_t* **hipblasCsyr2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, const *hipblasComplex* \*y, int incy, *hipblasComplex* \*AP, int lda)

*hipblasStatus\_t* **hipblasZsyr2**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, const *hipblasDoubleComplex* \*y, int incy, *hipblasDoubleComplex* \*AP, int lda)

BLAS Level 2 API.

syr2 performs the matrix-vector operations

$$A := A + \alpha x y^{**T} + \alpha y x^{**T}$$

where alpha is a scalar, x and y are vectors, and A is an n by n symmetric matrix.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int] the number of rows and columns of matrix A.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [in] device pointer storing vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **AP** – [inout] device pointer storing matrix A.
- **lda** – [in] [int] specifies the leading dimension of A.

*hipblasStatus\_t* **hipblasSsyr2Batched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*const x[], int incx, const float \*const y[], int incy, float \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasDsyrr2Batched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*const x[], int incx, const double \*const y[], int incy, double \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasCsyrr2Batched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const x[], int incx, const *hipblasComplex* \*const y[], int incy, *hipblasComplex* \*const AP[], int lda, int batchCount)

*hipblasStatus\_t* **hipblasZsyrr2Batched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const x[], int incx, const *hipblasDoubleComplex* \*const y[], int incy, *hipblasDoubleComplex* \*const AP[], int lda, int batchCount)

BLAS Level 2 API.

syrr2Batched performs a batch of matrix-vector operations

$$A[i] := A[i] + \alpha * x[i] * y[i]^T + \alpha * y[i] * x[i]^T$$

where alpha is a scalar, x[i] and y[i] are vectors, and A[i] is a n by n symmetric matrix, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int] the number of rows and columns of matrix A.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device array of device pointers storing each vector x<sub>i</sub>.
- **incx** – [in] [int] specifies the increment for the elements of each x<sub>i</sub>.
- **y** – [in] device array of device pointers storing each vector y<sub>i</sub>.
- **incy** – [in] [int] specifies the increment for the elements of each y<sub>i</sub>.
- **AP** – [inout] device array of device pointers storing each matrix A<sub>i</sub>.
- **lda** – [in] [int] specifies the leading dimension of each A<sub>i</sub>.
- **batchCount** – [in] [int] number of instances in the batch

*hipblasStatus\_t* **hipblasSsyrr2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const float \*alpha, const float \*x, int incx, *hipblasStride* stridex, const float \*y, int incy, *hipblasStride* stridey, float \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasDsyrr2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const double \*alpha, const double \*x, int incx, *hipblasStride* stridex, const double \*y, int incy, *hipblasStride* stridey, double \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasCsyrr2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasComplex* \*y, int incy, *hipblasStride* stridey, *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

*hipblasStatus\_t* **hipblasZsyrr2StridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, const *hipblasDoubleComplex* \*y, int incy, *hipblasStride* stridey, *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, int batchCount)

BLAS Level 2 API.

syr2StridedBatched the matrix-vector operations

$$A[i] := A[i] + \alpha * x[i] * y[i]**T + \alpha * y[i] * x[i]**T$$

where alpha is a scalar, x[i] and y[i] are vectors, and A[i] is a n by n symmetric matrices, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **n** – [in] [int] the number of rows and columns of each matrix A.
- **alpha** – [in] device pointer or host pointer to scalar alpha.
- **x** – [in] device pointer to the first vector x\_1.
- **incx** – [in] [int] specifies the increment for the elements of each x\_i.
- **stridex** – [in] [*hipblasStride*] specifies the pointer increment between vectors (x\_i) and (x\_i+1).
- **y** – [in] device pointer to the first vector y\_1.
- **incy** – [in] [int] specifies the increment for the elements of each y\_i.
- **stridey** – [in] [*hipblasStride*] specifies the pointer increment between vectors (y\_i) and (y\_i+1).
- **AP** – [inout] device pointer to the first matrix A\_1.
- **lda** – [in] [int] specifies the leading dimension of each A\_i.

- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A<sub>i</sub>) and the next one (A<sub>i+1</sub>)
- **batchCount** – [in] [int] number of instances in the batch

### 1.8.2.18 hipblasXtbmv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasStbmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const float \*AP, int lda, float \*x, int incx)

*hipblasStatus\_t* **hipblasDtbmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const double \*AP, int lda, double \*x, int incx)

*hipblasStatus\_t* **hipblasCtbmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const *hipblasComplex* \*AP, int lda, *hipblasComplex* \*x, int incx)

*hipblasStatus\_t* **hipblasZtbmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const *hipblasDoubleComplex* \*AP, int lda, *hipblasDoubleComplex* \*x, int incx)

BLAS Level 2 API.

tbmv performs one of the matrix-vector operations

$x := A * x$	or
$x := A^{**T} * x$	or
$x := A^{**H} * x,$	

x is a vectors and A is a banded m by m matrix (see description below).

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper banded triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: A is a lower banded triangular matrix.
- **transA** – [in] [hipblasOperation\_t] indicates whether matrix A is tranposed (conjugated) or not.
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: The main diagonal of A is assumed to consist of only 1's and is not referenced. HIPBLAS\_DIAG\_NON\_UNIT: No assumptions are made of A's main diagonal.
- **m** – [in] [int] the number of rows and columns of the matrix represented by A.
- **k** – [in] [int] if uplo == HIPBLAS\_FILL\_MODE\_UPPER, k specifies the number of super-diagonals of the matrix A. if uplo == HIPBLAS\_FILL\_MODE\_LOWER, k specifies the number of sub-diagonals of the matrix A. k must satisfy  $k > 0 \ \&\& \ k < \text{lda}$ .



- **AP** – [in] device pointer storing banded triangular matrix A. if `uplo == HIPBLAS_FILL_MODE_UPPER`: The matrix represented is an upper banded triangular matrix with the main diagonal and k super-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the k'th row, the first super diagonal resides on the RHS of the k-1'th row, etc, with the k'th diagonal on the RHS of the 0'th row. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; m = 5; k = 2) 1 6 9 0 0 0 0 9 8 7 0 2 7 8 0 0 6 7 8 9 0 0 3 8 7 -> 1 2 3 4 5 0 0 0 4 9 0 0 0 0 0 0 0 0 5 0 0 0 0 0 if `uplo == HIPBLAS_FILL_MODE_LOWER`: The matrix represented is a lower banded triangular matrix with the main diagonal and k sub-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the 0'th row, working up to the k'th diagonal residing on the LHS of the k'th row. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; m = 5; k = 2) 1 0 0 0 0 1 2 3 4 5 6 2 0 0 0 6 7 8 9 0 9 7 3 0 0 -> 9 8 7 0 0 0 8 8 4 0 0 0 0 0 0 0 7 9 5 0 0 0 0 0
- **lda** – [in] [int] specifies the leading dimension of A. lda must satisfy  $lda > k$ .
- **x** – [inout] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.

*hipblasStatus\_t* **hipblasStbmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const float \*const AP[], int lda, float \*const x[], int incx, int batchSize)

*hipblasStatus\_t* **hipblasDtbmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const double \*const AP[], int lda, double \*const x[], int incx, int batchSize)

*hipblasStatus\_t* **hipblasCtbmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const *hipblasComplex* \*const AP[], int lda, *hipblasComplex* \*const x[], int incx, int batchSize)

*hipblasStatus\_t* **hipblasZtbmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const *hipblasDoubleComplex* \*const AP[], int lda, *hipblasDoubleComplex* \*const x[], int incx, int batchSize)

BLAS Level 2 API.

tbmvBatched performs one of the matrix-vector operations

$x_i := A_i * x_i$	<b>or</b>
$x_i := A_i * T * x_i$	<b>or</b>
$x_i := A_i * H * x_i,$	

where ( $A_i, x_i$ ) is the i-th instance of the batch.  $x_i$  is a vector and  $A_i$  is an m by m matrix, for  $i = 1, \dots, batchSize$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.

- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: each  $A_i$  is an upper banded triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: each  $A_i$  is a lower banded triangular matrix.
- **transA** – [in] [hipblasOperation\_t] indicates whether each matrix  $A_i$  is transposed (conjugated) or not.
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: The main diagonal of each  $A_i$  is assumed to consist of only 1's and is not referenced. HIPBLAS\_DIAG\_NON\_UNIT: No assumptions are made of each  $A_i$ 's main diagonal.
- **m** – [in] [int] the number of rows and columns of the matrix represented by each  $A_i$ .
- **k** – [in] [int] if uplo == HIPBLAS\_FILL\_MODE\_UPPER, k specifies the number of super-diagonals of each matrix  $A_i$ . if uplo == HIPBLAS\_FILL\_MODE\_LOWER, k specifies the number of sub-diagonals of each matrix  $A_i$ . k must satisfy  $k > 0 \ \&\& \ k < \text{lda}$ .
- **AP** – [in] device array of device pointers storing each banded triangular matrix  $A_i$ . if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The matrix represented is an upper banded triangular matrix with the main diagonal and k super-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the k'th row, the first super diagonal resides on the RHS of the k-1'th row, etc, with the k'th diagonal on the RHS of the 0'th row. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; m = 5; k = 2) 1 6 9 0 0 0 0 9 8 7 0 2 7 8 0 0 6 7 8 9 0 0 3 8 7 -> 1 2 3 4 5 0 0 0 4 9 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The matrix represented is a lower banded triangular matrix with the main diagonal and k sub-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the 0'th row, working up to the k'th diagonal residing on the LHS of the k'th row. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; m = 5; k = 2) 1 0 0 0 0 1 2 3 4 5 6 2 0 0 0 6 7 8 9 0 9 7 3 0 0 -> 9 8 7 0 0 0 8 8 4 0 0 0 0 0 0 0 7 9 5 0 0 0 0 0
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ . lda must satisfy  $\text{lda} > k$ .
- **x** – [inout] device array of device pointer storing each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasStbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const float \*AP, int lda, *hipblasStride* strideA, float \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasDtbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const double \*AP, int lda, *hipblasStride* strideA, double \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasCtbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasZtbmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int k, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridx, int batchCount)

BLAS Level 2 API.

tbmvStridedBatched performs one of the matrix-vector operations

$x_i := A_i * x_i$	or
$x_i := A_i^T * x_i$	or
$x_i := A_i^H * x_i$ ,	

where  $(A_i, x_i)$  is the  $i$ -th instance of the batch.  $x_i$  is a vector and  $A_i$  is an  $m$  by  $m$  matrix, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER: each  $A_i$  is an upper banded triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: each  $A_i$  is a lower banded triangular matrix.
- transA** – [in] [*hipblasOperation\_t*] indicates whether each matrix  $A_i$  is transposed (conjugated) or not.
- diag** – [in] [*hipblasDiagType\_t*] HIPBLAS\_DIAG\_UNIT: The main diagonal of each  $A_i$  is assumed to consist of only 1's and is not referenced. HIPBLAS\_DIAG\_NON\_UNIT: No assumptions are made of each  $A_i$ 's main diagonal.
- m** – [in] [int] the number of rows and columns of the matrix represented by each  $A_i$ .
- k** – [in] [int] if uplo == HIPBLAS\_FILL\_MODE\_UPPER, k specifies the number of super-diagonals of each matrix  $A_i$ . if uplo == HIPBLAS\_FILL\_MODE\_LOWER, k specifies the number of sub-diagonals of each matrix  $A_i$ . k must satisfy  $k > 0 \ \&\& \ k < \text{lda}$ .
- AP** – [in] device array to the first matrix  $A_i$  of the batch. Stores each banded triangular matrix  $A_i$ . if uplo == HIPBLAS\_FILL\_MODE\_UPPER: The matrix represented is an upper banded triangular matrix with the main diagonal and k super-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the k'th row, the first super diagonal resides on the RHS of the k-1'th row, etc, with the k'th diagonal on the RHS of the 0'th row. Ex: (HIPBLAS\_FILL\_MODE\_UPPER; m = 5; k = 2) 1 6 9 0 0 0 0 9 8 7 0 2 7 8 0 0 6 7 8 9 0 0 3 8 7 -> 1 2 3 4 5 0 0 0 4 9 0 0 0 0 0 0 0 5 0 0 0 0 0 if uplo == HIPBLAS\_FILL\_MODE\_LOWER: The matrix represented is a lower banded triangular matrix with the main diagonal and k sub-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the 0'th row, working up to the k'th diagonal residing on the LHS of the k'th row. Ex: (HIPBLAS\_FILL\_MODE\_LOWER; m = 5; k = 2) 1 0 0 0 0 1 2 3 4 5 6 2 0 0 0 6 7 8 9 0 9 7 3 0 0 -> 9 8 7 0 0 0 8 8 4 0 0 0 0 0 0 0 7 9 5 0 0 0 0 0
- lda** – [in] [int] specifies the leading dimension of each  $A_i$ . lda must satisfy  $\text{lda} > k$ .
- strideA** – [in] [*hipblasStride*] stride from the start of one  $A_i$  matrix to the next  $A_{(i+1)}$ .

- **x** – [inout] device array to the first vector  $x_i$  of the batch.
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] stride from the start of one  $x_i$  matrix to the next  $x_{(i+1)}$ .
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.2.19 hipblasXtbsv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasStbsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const float \*AP, int lda, float \*x, int incx)

*hipblasStatus\_t* **hipblasDtbsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const double \*AP, int lda, double \*x, int incx)

*hipblasStatus\_t* **hipblasCtbsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const *hipblasComplex* \*AP, int lda, *hipblasComplex* \*x, int incx)

*hipblasStatus\_t* **hipblasZtbsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const *hipblasDoubleComplex* \*AP, int lda, *hipblasDoubleComplex* \*x, int incx)

BLAS Level 2 API.

tbsv solves

$$A^*x = b \text{ or } A^{**T}x = b \text{ or } A^{**H}x = b,$$

where  $x$  and  $b$  are vectors and  $A$  is a banded triangular matrix.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER:  $A$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER:  $A$  is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_N: Solves  $A^*x = b$  HIPBLAS\_OP\_T: Solves  $A^{**T}x = b$  HIPBLAS\_OP\_C: Solves  $A^{**H}x = b$
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT:  $A$  is assumed to be unit triangular (i.e. the diagonal elements of  $A$  are not used in computations). HIPBLAS\_DIAG\_NON\_UNIT:  $A$  is not assumed to be unit triangular.
- **n** – [in] [int]  $n$  specifies the number of rows of  $b$ .  $n \geq 0$ .
- **k** – [in] [int] if( $uplo == HIPBLAS_FILL_MODE_UPPER$ )  $k$  specifies the number of super-diagonals of  $A$ . if( $uplo == HIPBLAS_FILL_MODE_LOWER$ )  $k$  specifies the number of sub-diagonals of  $A$ .  $k \geq 0$ .
- **AP** – [in] device pointer storing the matrix  $A$  in banded format.

- **lda** – [in] [int] specifies the leading dimension of A.  $lda \geq (k + 1)$ .
- **x** – [inout] device pointer storing input vector b. Overwritten by the output vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.

*hipblasStatus\_t* **hipblasStbsvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const float \*const AP[], int lda, float \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasDtbsvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const double \*const AP[], int lda, double \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasCtbsvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const *hipblasComplex* \*const AP[], int lda, *hipblasComplex* \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasZtbsvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const *hipblasDoubleComplex* \*const AP[], int lda, *hipblasDoubleComplex* \*const x[], int incx, int batchCount)

BLAS Level 2 API.

tbsvBatched solves

$$A_i * x_i = b_i \text{ or } A_i ** T * x_i = b_i \text{ or } A_i ** H * x_i = b_i,$$

where  $x_i$  and  $b_i$  are vectors and  $A_i$  is a banded triangular matrix, for  $i = [1, \text{batchCount}]$ .

The input vectors  $b_i$  are overwritten by the output vectors  $x_i$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER:  $A_i$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER:  $A_i$  is a lower triangular matrix.
- **transA** – [in] [*hipblasOperation\_t*] HIPBLAS\_OP\_N: Solves  $A_i * x_i = b_i$  HIPBLAS\_OP\_T: Solves  $A_i ** T * x_i = b_i$  HIPBLAS\_OP\_C: Solves  $A_i ** H * x_i = b_i$
- **diag** – [in] [*hipblasDiagType\_t*] HIPBLAS\_DIAG\_UNIT: each  $A_i$  is assumed to be unit triangular (i.e. the diagonal elements of each  $A_i$  are not used in computations). HIPBLAS\_DIAG\_NON\_UNIT: each  $A_i$  is not assumed to be unit triangular.
- **n** – [in] [int] n specifies the number of rows of each  $b_i$ .  $n \geq 0$ .
- **k** – [in] [int] if(uplo == HIPBLAS\_FILL\_MODE\_UPPER) k specifies the number of super-diagonals of each  $A_i$ . if(uplo == HIPBLAS\_FILL\_MODE\_LOWER) k specifies the number of sub-diagonals of each  $A_i$ .  $k \geq 0$ .
- **AP** – [in] device vector of device pointers storing each matrix  $A_i$  in banded format.

- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .  $lda \geq (k + 1)$ .
- **x** – [inout] device vector of device pointers storing each input vector  $b_i$ . Overwritten by each output vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasStbsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const float \*AP, int lda, *hipblasStride* strideA, float \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasDtbsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const double \*AP, int lda, *hipblasStride* strideA, double \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasCtbsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasZtbsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int n, int k, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

BLAS Level 2 API.

tbsvStridedBatched solves

$$A_i * x_i = b_i \text{ or } A_i ** T * x_i = b_i \text{ or } A_i ** H * x_i = b_i,$$

where  $x_i$  and  $b_i$  are vectors and  $A_i$  is a banded triangular matrix, for  $i = [1, \text{batchCount}]$ .

The input vectors  $b_i$  are overwritten by the output vectors  $x_i$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER:  $A_i$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER:  $A_i$  is a lower triangular matrix.
- **transA** – [in] [*hipblasOperation\_t*] HIPBLAS\_OP\_N: Solves  $A_i * x_i = b_i$  HIPBLAS\_OP\_T: Solves  $A_i ** T * x_i = b_i$  HIPBLAS\_OP\_C: Solves  $A_i ** H * x_i = b_i$
- **diag** – [in] [*hipblasDiagType\_t*] HIPBLAS\_DIAG\_UNIT: each  $A_i$  is assumed to be unit triangular (i.e. the diagonal elements of each  $A_i$  are not used in computations). HIPBLAS\_DIAG\_NON\_UNIT: each  $A_i$  is not assumed to be unit triangular.

- **n** – [in] [int] n specifies the number of rows of each  $b_i$ .  $n \geq 0$ .
- **k** – [in] [int] if( $uplo == HIPBLAS\_FILL\_MODE\_UPPER$ ) k specifies the number of super-diagonals of each  $A_i$ . if( $uplo == HIPBLAS\_FILL\_MODE\_LOWER$ ) k specifies the number of sub-diagonals of each  $A_i$ .  $k \geq 0$ .
- **AP** – [in] device pointer pointing to the first banded matrix  $A_1$ .
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .  $lda \geq (k + 1)$ .
- **strideA** – [in] [hipblasStride] specifies the distance between the start of one matrix ( $A_i$ ) and the next ( $A_{i+1}$ ).
- **x** – [inout] device pointer pointing to the first input vector  $b_1$ . Overwritten by output vectors  $x$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] specifies the distance between the start of one vector ( $x_i$ ) and the next ( $x_{i+1}$ ).
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.2.20 hipblasXtpmv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasStpmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*AP, float \*x, int incx)

*hipblasStatus\_t* **hipblasDtpmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*AP, double \*x, int incx)

*hipblasStatus\_t* **hipblasCtpmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*AP, *hipblasComplex* \*x, int incx)

*hipblasStatus\_t* **hipblasZtpmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*AP, *hipblasDoubleComplex* \*x, int incx)

BLAS Level 2 API.

tpmv performs one of the matrix-vector operations

$$x = A * x \text{ or } x = A^{**T} * x,$$

where  $x$  is an  $n$  element vector and  $A$  is an  $n$  by  $n$  unit, or non-unit, upper or lower triangular matrix, supplied in the pack form.

The vector  $x$  is overwritten.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.



- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: A is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t]
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: A is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: A is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of A.  $m \geq 0$ .
- **AP** – [in] device pointer storing matrix A, of dimension at least  $(m * (m + 1) / 2)$ . Before entry with uplo = HIPBLAS\_FILL\_MODE\_UPPER, the array A must contain the upper triangular matrix packed sequentially, column by column, so that A[0] contains  $a_{\{0,0\}}$ , A[1] and A[2] contain  $a_{\{0,1\}}$  and  $a_{\{1,1\}}$  respectively, and so on. Before entry with uplo = HIPBLAS\_FILL\_MODE\_LOWER, the array A must contain the lower triangular matrix packed sequentially, column by column, so that A[0] contains  $a_{\{0,0\}}$ , A[1] and A[2] contain  $a_{\{1,0\}}$  and  $a_{\{2,0\}}$  respectively, and so on. Note that when DIAG = HIPBLAS\_DIAG\_UNIT, the diagonal elements of A are not referenced, but are assumed to be unity.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x. incx must not be zero.

*hipblasStatus\_t* **hipblasStpmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*const AP[], float \*const x[], int incx, int batchSize)

*hipblasStatus\_t* **hipblasDtpmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*const AP[], double \*const x[], int incx, int batchSize)

*hipblasStatus\_t* **hipblasCtpmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*const AP[], *hipblasComplex* \*const x[], int incx, int batchSize)

*hipblasStatus\_t* **hipblasZtpmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*const AP[], *hipblasDoubleComplex* \*const x[], int incx, int batchSize)

BLAS Level 2 API.

tpmvBatched performs one of the matrix-vector operations

$$x_i = A_i * x_i \text{ or } x_i = A_i^T * x_i, \quad 0 \leq i < \text{batchCount}$$

where  $x_i$  is an n element vector and  $A_i$  is an n by n (unit, or non-unit, upper or lower triangular matrix)

The vectors  $x_i$  are overwritten.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.



- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER:  $A_i$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER:  $A_i$  is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t]
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT:  $A_i$  is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT:  $A_i$  is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of matrices  $A_i$ .  $m \geq 0$ .
- **AP** – [in] device pointer storing pointer of matrices  $A_i$ , of dimension ( lda, m )
- **x** – [in] device pointer storing vectors  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of vectors  $x_i$ .
- **batchCount** – [in] [int] The number of batched matrices/vectors.

*hipblasStatus\_t* **hipblasStpmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*AP, *hipblasStride* strideA, float \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasDtpmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*AP, *hipblasStride* strideA, double \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasCtpmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*AP, *hipblasStride* strideA, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasZtpmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*AP, *hipblasStride* strideA, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

BLAS Level 2 API.

tpmvStridedBatched performs one of the matrix-vector operations

$$x_i = A_i * x_i \text{ or } x_i = A^* T * x_i, \quad 0 \leq i < \text{batchCount}$$

where  $x_i$  is an n element vector and  $A_i$  is an n by n (unit, or non-unit, upper or lower triangular matrix) with strides specifying how to retrieve  $x_i$  (resp.  $A_i$ ) from  $x_{i-1}$  (resp.  $A_i$ ).

The vectors  $x_i$  are overwritten.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.

- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A<sub>i</sub> is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: A<sub>i</sub> is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t]
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: A<sub>i</sub> is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: A<sub>i</sub> is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of matrices A<sub>i</sub>. m >= 0.
- **AP** – [in] device pointer of the matrix A<sub>0</sub>, of dimension (lda, m)
- **strideA** – [in] [hipblasStride] stride from the start of one A<sub>i</sub> matrix to the next A<sub>{i + 1}</sub>
- **x** – [in] device pointer storing the vector x<sub>0</sub>.
- **incx** – [in] [int] specifies the increment for the elements of one vector x.
- **stridx** – [in] [hipblasStride] stride from the start of one x<sub>i</sub> vector to the next x<sub>{i + 1}</sub>
- **batchCount** – [in] [int] The number of batched matrices/vectors.

### 1.8.2.21 hipblasXtpsv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasStpsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*AP, float \*x, int incx)

*hipblasStatus\_t* **hipblasDtpsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*AP, double \*x, int incx)

*hipblasStatus\_t* **hipblasCtpsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*AP, *hipblasComplex* \*x, int incx)

*hipblasStatus\_t* **hipblasZtpsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*AP, *hipblasDoubleComplex* \*x, int incx)

BLAS Level 2 API.

tpsv solves

$$A*x = b \text{ or } A^{**T}*x = b, \text{ or } A^{**H}*x = b,$$

where x and b are vectors and A is a triangular matrix stored in the packed format.

The input vector b is overwritten by the output vector x.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: A is a lower triangular matrix.

- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_N: Solves  $A * x = b$  HIPBLAS\_OP\_T: Solves  $A ** T * x = b$  HIPBLAS\_OP\_C: Solves  $A ** H * x = b$
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: A is assumed to be unit triangular (i.e. the diagonal elements of A are not used in computations). HIPBLAS\_DIAG\_NON\_UNIT: A is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of b.  $m \geq 0$ .
- **AP** – [in] device pointer storing the packed version of matrix A, of dimension  $\geq (n * (n + 1) / 2)$
- **x** – [inout] device pointer storing vector b on input, overwritten by x on output.
- **incx** – [in] [int] specifies the increment for the elements of x.

*hipblasStatus\_t* **hipblasStpsvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*const AP[], float \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasDtpsivBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*const AP[], double \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasCtpsivBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*const AP[], *hipblasComplex* \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasZtpsivBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*const AP[], *hipblasDoubleComplex* \*const x[], int incx, int batchCount)

BLAS Level 2 API.

tpsivBatched solves

$$A_i * x_i = b_i \text{ or } A_i ** T * x_i = b_i, \text{ or } A_i ** H * x_i = b_i,$$

where  $x_i$  and  $b_i$  are vectors and  $A_i$  is a triangular matrix stored in the packed format, for  $i$  in  $[1, \text{batchCount}]$ .

The input vectors  $b_i$  are overwritten by the output vectors  $x_i$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: each  $A_i$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: each  $A_i$  is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_N: Solves  $A * x = b$  HIPBLAS\_OP\_T: Solves  $A ** T * x = b$  HIPBLAS\_OP\_C: Solves  $A ** H * x = b$
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: each  $A_i$  is assumed to be unit triangular (i.e. the diagonal elements of each  $A_i$  are not used in computations). HIPBLAS\_DIAG\_NON\_UNIT: each  $A_i$  is not assumed to be unit triangular.

- **m** – [in] [int] m specifies the number of rows of each  $b_i$ .  $m \geq 0$ .
- **AP** – [in] device array of device pointers storing the packed versions of each matrix  $A_i$ , of dimension  $\geq (n * (n + 1) / 2)$
- **x** – [inout] device array of device pointers storing each input vector  $b_i$ , overwritten by  $x_i$  on output.
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **batchCount** – [in] [int] specifies the number of instances in the batch.

*hipblasStatus\_t* **hipblasStpsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*AP, *hipblasStride* strideA, float \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasDtpsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*AP, *hipblasStride* strideA, double \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasCtpsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*AP, *hipblasStride* strideA, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasZtpsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*AP, *hipblasStride* strideA, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

BLAS Level 2 API.

tpsvStridedBatched solves

$$A_i * x_i = b_i \text{ or } A_i ** T * x_i = b_i, \text{ or } A_i ** H * x_i = b_i,$$

where  $x_i$  and  $b_i$  are vectors and  $A_i$  is a triangular matrix stored in the packed format, for  $i$  in  $[1, \text{batchCount}]$ .

The input vectors  $b_i$  are overwritten by the output vectors  $x_i$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER: each  $A_i$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: each  $A_i$  is a lower triangular matrix.
- **transA** – [in] [*hipblasOperation\_t*] HIPBLAS\_OP\_N: Solves  $A * x = b$  HIPBLAS\_OP\_T: Solves  $A ** T * x = b$  HIPBLAS\_OP\_C: Solves  $A ** H * x = b$

- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: each  $A_i$  is assumed to be unit triangular (i.e. the diagonal elements of each  $A_i$  are not used in computations). HIPBLAS\_DIAG\_NON\_UNIT: each  $A_i$  is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of each  $b_i$ .  $m \geq 0$ .
- **AP** – [in] device pointer pointing to the first packed matrix  $A_1$ , of dimension  $\geq (n * (n + 1) / 2)$
- **strideA** – [in] [hipblasStride] stride from the beginning of one packed matrix ( $AP_i$ ) and the next ( $AP_{i+1}$ ).
- **x** – [inout] device pointer pointing to the first input vector  $b_1$ . Overwritten by each  $x_i$  on output.
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridx** – [in] [hipblasStride] stride from the beginning of one vector ( $x_i$ ) and the next ( $x_{i+1}$ ).
- **batchCount** – [in] [int] specifies the number of instances in the batch.

### 1.8.2.22 hipblasXtrmv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasStrmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*AP, int lda, float \*x, int incx)

*hipblasStatus\_t* **hipblasDtrmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*AP, int lda, double \*x, int incx)

*hipblasStatus\_t* **hipblasCtrmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*AP, int lda, *hipblasComplex* \*x, int incx)

*hipblasStatus\_t* **hipblasZtrmv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*AP, int lda, *hipblasDoubleComplex* \*x, int incx)

BLAS Level 2 API.

trmv performs one of the matrix-vector operations

$$x = A * x \text{ or } x = A^T * x,$$

where  $x$  is an  $n$  element vector and  $A$  is an  $n$  by  $n$  unit, or non-unit, upper or lower triangular matrix.

The vector  $x$  is overwritten.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER:  $A$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER:  $A$  is a lower triangular matrix.

- **transA** – [in] [hipblasOperation\_t]
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: A is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: A is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of A.  $m \geq 0$ .
- **AP** – [in] device pointer storing matrix A, of dimension ( lda, m )
- **lda** – [in] [int] specifies the leading dimension of A.  $lda = \max(1, m)$ .
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.

*hipblasStatus\_t* **hipblasStrmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*const AP[], int lda, float \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasDtrmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*const AP[], int lda, double \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasCtrmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*const AP[], int lda, *hipblasComplex* \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasZtrmvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*const AP[], int lda, *hipblasDoubleComplex* \*const x[], int incx, int batchCount)

BLAS Level 2 API.

trmvBatched performs one of the matrix-vector operations

$$x_i = A_i * x_i \text{ or } x_i = A_i^T * x_i, \quad 0 \leq i < \text{batchCount}$$

where  $x_i$  is an n element vector and  $A_i$  is an n by n (unit, or non-unit, upper or lower triangular matrix)

The vectors  $x_i$  are overwritten.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER:  $A_i$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER:  $A_i$  is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t]
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT:  $A_i$  is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT:  $A_i$  is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of matrices  $A_i$ .  $m \geq 0$ .

- **AP** – [in] device pointer storing pointer of matrices  $A_i$ , of dimension ( lda, m )
- **lda** – [in] [int] specifies the leading dimension of  $A_i$ .  $lda \geq \max(1, m)$ .
- **x** – [in] device pointer storing vectors  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of vectors  $x_i$ .
- **batchCount** – [in] [int] The number of batched matrices/vectors.

*hipblasStatus\_t* **hipblasStrmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*AP, int lda, *hipblasStride* strideA, float \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasDtrmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*AP, int lda, *hipblasStride* strideA, double \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasCtrmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasZtrmvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

BLAS Level 2 API.

trmvStridedBatched performs one of the matrix-vector operations

$$x_i = A_i * x_i \text{ or } x_i = A_i^T * x_i, \quad 0 \leq i < \text{batchCount}$$

where  $x_i$  is an  $n$  element vector and  $A_i$  is an  $n$  by  $n$  (unit, or non-unit, upper or lower triangular matrix) with strides specifying how to retrieve  $x_i$  (resp.  $A_i$ ) from  $x_{i-1}$  (resp.  $A_i$ ).

The vectors  $x_i$  are overwritten.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER:  $A_i$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER:  $A_i$  is a lower triangular matrix.
- **transA** – [in] [*hipblasOperation\_t*]
- **diag** – [in] [*hipblasDiagType\_t*] HIPBLAS\_DIAG\_UNIT:  $A_i$  is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT:  $A_i$  is not assumed to be unit triangular.

- **m** – [in] [int] m specifies the number of rows of matrices A<sub>i</sub>. m >= 0.
- **AP** – [in] device pointer of the matrix A<sub>0</sub>, of dimension ( lda, m )
- **lda** – [in] [int] specifies the leading dimension of A<sub>i</sub>. lda >= max( 1, m ).
- **strideA** – [in] [hipblasStride] stride from the start of one A<sub>i</sub> matrix to the next A<sub>{i + 1}</sub>
- **x** – [in] device pointer storing the vector x<sub>0</sub>.
- **incx** – [in] [int] specifies the increment for the elements of one vector x.
- **stridex** – [in] [hipblasStride] stride from the start of one x<sub>i</sub> vector to the next x<sub>{i + 1}</sub>
- **batchCount** – [in] [int] The number of batched matrices/vectors.

### 1.8.2.23 hipblasXtrsv + Batched, StridedBatched

*hipblasStatus\_t* **hipblasStrsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*AP, int lda, float \*x, int incx)

*hipblasStatus\_t* **hipblasDtrsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*AP, int lda, double \*x, int incx)

*hipblasStatus\_t* **hipblasCtrsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*AP, int lda, *hipblasComplex* \*x, int incx)

*hipblasStatus\_t* **hipblasZtrsv**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*AP, int lda, *hipblasDoubleComplex* \*x, int incx)

BLAS Level 2 API.

trsv solves

$$A * x = b \text{ or } A^{**T} * x = b,$$

where x and b are vectors and A is a triangular matrix.

The vector x is overwritten on b.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: A is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t]
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: A is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: A is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of b. m >= 0.



- **AP** – [in] device pointer storing matrix A, of dimension ( lda, m )
- **lda** – [in] [int] specifies the leading dimension of A.  $lda = \max(1, m)$ .
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.

*hipblasStatus\_t* **hipblasStrsvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*const AP[], int lda, float \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasDtrsvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*const AP[], int lda, double \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasCtrsvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*const AP[], int lda, *hipblasComplex* \*const x[], int incx, int batchCount)

*hipblasStatus\_t* **hipblasZtrsvBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*const AP[], int lda, *hipblasDoubleComplex* \*const x[], int incx, int batchCount)

BLAS Level 2 API.

trsvBatched solves

$$A_i * x_i = b_i \text{ or } A_i^{**T} * x_i = b_i,$$

where (A<sub>i</sub>, x<sub>i</sub>, b<sub>i</sub>) is the i-th instance of the batch. x<sub>i</sub> and b<sub>i</sub> are vectors and A<sub>i</sub> is an m by m triangular matrix.

The vector x is overwritten on b.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: A is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t]
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: A is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: A is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of b.  $m \geq 0$ .
- **AP** – [in] device array of device pointers storing each matrix A<sub>i</sub>.
- **lda** – [in] [int] specifies the leading dimension of each A<sub>i</sub>.  $lda = \max(1, m)$
- **x** – [in] device array of device pointers storing each vector x<sub>i</sub>.

- **incx** – [in] [int] specifies the increment for the elements of x.
- **batchCount** – [in] [int] number of instances in the batch

*hipblasStatus\_t* **hipblasStrsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const float \*AP, int lda, *hipblasStride* strideA, float \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasDtrsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const double \*AP, int lda, *hipblasStride* strideA, double \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasCtrsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

*hipblasStatus\_t* **hipblasZtrsvStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasDoubleComplex* \*x, int incx, *hipblasStride* stridex, int batchCount)

BLAS Level 2 API.

trsvStridedBatched solves

$$A_i * x_i = b_i \text{ or } A_i ** T * x_i = b_i,$$

where (A<sub>i</sub>, x<sub>i</sub>, b<sub>i</sub>) is the i-th instance of the batch. x<sub>i</sub> and b<sub>i</sub> are vectors and A<sub>i</sub> is an m by m triangular matrix, for i = 1, ..., batchCount.

The vector x is overwritten on b.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: A is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t]
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: A is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: A is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of each b<sub>i</sub>. m >= 0.
- **AP** – [in] device pointer to the first matrix (A<sub>1</sub>) in the batch, of dimension ( lda, m )
- **strideA** – [in] [hipblasStride] stride from the start of one A<sub>i</sub> matrix to the next A<sub>i</sub>(i + 1)

- **lda** – **[in]** [int] specifies the leading dimension of each  $A_i$ .  $lda = \max(1, m)$ .
- **x** – **[inout]** device pointer to the first vector ( $x_1$ ) in the batch.
- **stridex** – **[in]** [hipblasStride] stride from the start of one  $x_i$  vector to the next  $x_{(i+1)}$
- **incx** – **[in]** [int] specifies the increment for the elements of each  $x_i$ .
- **batchCount** – **[in]** [int] number of instances in the batch

### 1.8.3 Level 3 BLAS

#### List of Level-3 BLAS Functions

- *hipblasXgemm + Batched, StridedBatched*
- *hipblasXherk + Batched, StridedBatched*
- *hipblasXherkx + Batched, StridedBatched*
- *hipblasXher2k + Batched, StridedBatched*
- *hipblasXsymm + Batched, StridedBatched*
- *hipblasXsyrk + Batched, StridedBatched*
- *hipblasXsyr2k + Batched, StridedBatched*
- *hipblasXsyrkx + Batched, StridedBatched*
- *hipblasXgeam + Batched, StridedBatched*
- *hipblasXhemm + Batched, StridedBatched*
- *hipblasXtrmm + Batched, StridedBatched*
- *hipblasXtrsm + Batched, StridedBatched*
- *hipblasXtrtri + Batched, StridedBatched*
- *hipblasXdgmm + Batched, StridedBatched*

#### 1.8.3.1 hipblasXgemm + Batched, StridedBatched

*hipblasStatus\_t* **hipblasHgemm**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const *hipblasHalf* \*alpha, const *hipblasHalf* \*AP, int lda, const *hipblasHalf* \*BP, int ldb, const *hipblasHalf* \*beta, *hipblasHalf* \*CP, int ldc)

*hipblasStatus\_t* **hipblasSgemm**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const float \*alpha, const float \*AP, int lda, const float \*BP, int ldb, const float \*beta, float \*CP, int ldc)

*hipblasStatus\_t* **hipblasDgemm**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const double \*alpha, const double \*AP, int lda, const double \*BP, int ldb, const double \*beta, double \*CP, int ldc)

*hipblasStatus\_t* **hipblasCgemm**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*BP, int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZgemm**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*BP, int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

gemm performs one of the matrix-matrix operations

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

where  $\text{op}(X)$  is one of

$\text{op}(X) = X$             **or**  
 $\text{op}(X) = X^{**T}$        **or**  
 $\text{op}(X) = X^{**H}$ ,

alpha and beta are scalars, and A, B and C are matrices, with  $\text{op}(A)$  an m by k matrix,  $\text{op}(B)$  a k by n matrix and C an m by n matrix.

- Supported precisions in rocBLAS : h,s,d,c,z
- Supported precisions in cuBLAS : h,s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t]

.

- **transA** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(A)$
- **transB** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(B)$
- **m** – [in] [int] number of rows of matrices  $\text{op}(A)$  and C
- **n** – [in] [int] number of columns of matrices  $\text{op}(B)$  and C
- **k** – [in] [int] number of columns of matrix  $\text{op}(A)$  and number of rows of matrix  $\text{op}(B)$
- **alpha** – [in] device pointer or host pointer specifying the scalar alpha.
- **AP** – [in] device pointer storing matrix A.
- **lda** – [in] [int] specifies the leading dimension of A.
- **BP** – [in] device pointer storing matrix B.
- **ldb** – [in] [int] specifies the leading dimension of B.
- **beta** – [in] device pointer or host pointer specifying the scalar beta.
- **CP** – [inout] device pointer storing matrix C on the GPU.
- **ldc** – [in] [int] specifies the leading dimension of C.

*hipblasStatus\_t* **hipblasHgemvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const *hipblasHalf* \*alpha, const *hipblasHalf* \*const AP[], int lda, const *hipblasHalf* \*const BP[], int ldb, const *hipblasHalf* \*beta, *hipblasHalf* \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasSgemvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const float \*alpha, const float \*const AP[], int lda, const float \*const BP[], int ldb, const float \*beta, float \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasDgemvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const double \*alpha, const double \*const AP[], int lda, const double \*const BP[], int ldb, const double \*beta, double \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasCgemvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const BP[], int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasZgemvBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const BP[], int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const CP[], int ldc, int batchSize)

BLAS Level 3 API.

gemvBatched performs one of the batched matrix-matrix operations  $C_i = \alpha * op(A_i) * op(B_i) + \beta * C_i$ , for  $i = 1, \dots, batchSize$ . where  $op(X)$  is one of  $op(X) = X$  or  $op(X) = X^T$  or  $op(X) = X^H$ ,  $\alpha$  and  $\beta$  are scalars, and A, B and C are strided batched matrices, with  $op(A)$  an m by k by batchSize strided\_batched matrix,  $op(B)$  an k by n by batchSize strided\_batched matrix and C an m by n by batchSize strided\_batched matrix.

- Supported precisions in rocBLAS : h,s,d,c,z
- Supported precisions in cuBLAS : h,s,d,c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **transA** – [in] [*hipblasOperation\_t*] specifies the form of  $op(A)$
- **transB** – [in] [*hipblasOperation\_t*] specifies the form of  $op(B)$
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **k** – [in] [int] matrix dimension k.
- **alpha** – [in] device pointer or host pointer specifying the scalar alpha.
- **AP** – [in] device array of device pointers storing each matrix  $A_i$ .

- **lda** – [in] [int] specifies the leading dimension of each A<sub>i</sub>.
- **BP** – [in] device array of device pointers storing each matrix B<sub>i</sub>.
- **ldb** – [in] [int] specifies the leading dimension of each B<sub>i</sub>.
- **beta** – [in] device pointer or host pointer specifying the scalar beta.
- **CP** – [inout] device array of device pointers storing each matrix C<sub>i</sub>.
- **ldc** – [in] [int] specifies the leading dimension of each C<sub>i</sub>.
- **batchCount** – [in] [int] number of gemm operations in the batch

*hipblasStatus\_t* **hipblasHgemvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const *hipblasHalf* \*alpha, const *hipblasHalf* \*AP, int lda, long long strideA, const *hipblasHalf* \*BP, int ldb, long long strideB, const *hipblasHalf* \*beta, *hipblasHalf* \*CP, int ldc, long long strideC, int batchCount)

*hipblasStatus\_t* **hipblasSgemvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const float \*alpha, const float \*AP, int lda, long long strideA, const float \*BP, int ldb, long long strideB, const float \*beta, float \*CP, int ldc, long long strideC, int batchCount)

*hipblasStatus\_t* **hipblasDgemvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const double \*alpha, const double \*AP, int lda, long long strideA, const double \*BP, int ldb, long long strideB, const double \*beta, double \*CP, int ldc, long long strideC, int batchCount)

*hipblasStatus\_t* **hipblasCgemvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, long long strideA, const *hipblasComplex* \*BP, int ldb, long long strideB, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc, long long strideC, int batchCount)

*hipblasStatus\_t* **hipblasZgemvStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, long long strideA, const *hipblasDoubleComplex* \*BP, int ldb, long long strideB, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc, long long strideC, int batchCount)

BLAS Level 3 API.

gemvStridedBatched performs one of the strided batched matrix-matrix operations

$$C_i = \alpha * op(A_i) * op(B_i) + \beta * C_i, \text{ for } i = 1, \dots, \text{batchCount}.$$

where op( X ) is one of

$\text{op}(X) = X$       **or**  
 $\text{op}(X) = X^{**T}$     **or**  
 $\text{op}(X) = X^{**H}$ ,

alpha and beta are scalars, and A, B and C are strided batched matrices, with  $\text{op}(A)$  an m by k by batchCount strided\_batched matrix,  $\text{op}(B)$  an k by n by batchCount strided\_batched matrix and C an m by n by batchCount strided\_batched matrix.

- Supported precisions in rocBLAS : h,s,d,c,z
- Supported precisions in cuBLAS : h,s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **transA** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(A)$
- **transB** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(B)$
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **k** – [in] [int] matrix dimension k.
- **alpha** – [in] device pointer or host pointer specifying the scalar alpha.
- **AP** – [in] device pointer pointing to the first matrix  $A_1$ .
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .
- **strideA** – [in] [hipblasStride] stride from the start of one  $A_i$  matrix to the next  $A_{i+1}$ .
- **BP** – [in] device pointer pointing to the first matrix  $B_1$ .
- **ldb** – [in] [int] specifies the leading dimension of each  $B_i$ .
- **strideB** – [in] [hipblasStride] stride from the start of one  $B_i$  matrix to the next  $B_{i+1}$ .
- **beta** – [in] device pointer or host pointer specifying the scalar beta.
- **CP** – [inout] device pointer pointing to the first matrix  $C_1$ .
- **ldc** – [in] [int] specifies the leading dimension of each  $C_i$ .
- **strideC** – [in] [hipblasStride] stride from the start of one  $C_i$  matrix to the next  $C_{i+1}$ .
- **batchCount** – [in] [int] number of gemm operations in the batch

#### 1.8.3.2 hipblasXherk + Batched, StridedBatched

*hipblasStatus\_t* **hipblasCherk**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const *hipblasComplex* \*AP, int lda, const float \*beta, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZherk**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const double \*beta, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

herk performs one of the matrix-matrix operations for a Hermitian rank-k update

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

where alpha and beta are scalars, op(A) is an n by k matrix, and C is a n x n Hermitian matrix stored as either upper or lower.

```
op( A ) = A,  and A is n by k if transA == HIPBLAS_OP_N
op( A ) = A^H and A is k by n if transA == HIPBLAS_OP_C
```

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_C:  $\text{op}(A) = A^H$  HIPBLAS\_OP\_N:  $\text{op}(A) = A$
- **n** – [in] [int] n specifies the number of rows and columns of C.  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of op(A).  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] pointer storing matrix A on the GPU. Martrix dimension is (lda, k) when if transA = HIPBLAS\_OP\_N, otherwise (lda, n) only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if transA = HIPBLAS\_OP\_N,  $\text{lda} \geq \max(1, n)$ , otherwise  $\text{lda} \geq \max(1, k)$ .
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] pointer storing matrix C on the GPU. The imaginary component of the diagonal elements are not used but are set to zero unless quick return.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $\text{ldc} \geq \max(1, n)$ .

*hipblasStatus\_t* **hipblasCherkBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const *hipblasComplex* \*const AP[], int lda, const float \*beta, *hipblasComplex* \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasZherkBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const double \*beta, *hipblasDoubleComplex* \*const CP[], int ldc, int batchCount)

BLAS Level 3 API.

herkBatched performs a batch of the matrix-matrix operations for a Hermitian rank-k update

$$C_i := \alpha * \text{op}(A_i) * \text{op}(A_i)^H + \beta * C_i$$



where alpha and beta are scalars, op(A) is an n by k matrix, and C\_i is a n x n Hermitian matrix stored as either upper or lower.

```
op( A_i ) = A_i, and A_i is n by k if transA == HIPBLAS_OP_N
op( A_i ) = A_i^H and A_i is k by n if transA == HIPBLAS_OP_C
```

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C\_i is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C\_i is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_C:  $op(A) = A^H$  HIPBLAS\_OP\_N:  $op(A) = A$
- **n** – [in] [int] n specifies the number of rows and columns of C\_i.  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of op(A).  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] device array of device pointers storing each matrix\_i A of dimension (lda, k) when transA is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if transA = HIPBLAS\_OP\_N,  $lda \geq \max(1, n)$ , otherwise  $lda \geq \max(1, k)$ .
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] device array of device pointers storing each matrix C\_i on the GPU. The imaginary component of the diagonal elements are not used but are set to zero unless quick return.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $ldc \geq \max(1, n)$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasCherkStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const float \*beta, *hipblasComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasZherkStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const double \*beta, *hipblasDoubleComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

BLAS Level 3 API.

herkStridedBatched performs a batch of the matrix-matrix operations for a Hermitian rank-k update

$C_i := \alpha * op(A_i) * op(A_i)^H + \beta * C_i$

where alpha and beta are scalars, op(A) is an n by k matrix, and C\_i is a n x n Hermitian matrix stored as either upper or lower.

```
op( A_i ) = A_i, and A_i is n by k if transA == HIPBLAS_OP_N
op( A_i ) = A_i^H and A_i is k by n if transA == HIPBLAS_OP_C
```

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C\_i is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C\_i is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_C:  $op(A) = A^H$  HIPBLAS\_OP\_N:  $op(A) = A$
- **n** – [in] [int] n specifies the number of rows and columns of C\_i.  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of op(A).  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] Device pointer to the first matrix A\_1 on the GPU of dimension (lda, k) when transA is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if transA = HIPBLAS\_OP\_N,  $lda \geq \max(1, n)$ , otherwise  $lda \geq \max(1, k)$ .
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A\_i) and the next one (A\_{i+1})
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] Device pointer to the first matrix C\_1 on the GPU. The imaginary component of the diagonal elements are not used but are set to zero unless quick return.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $ldc \geq \max(1, n)$ .
- **strideC** – [inout] [hipblasStride] stride from the start of one matrix (C\_i) and the next one (C\_{i+1})
- **batchCount** – [in] [int] number of instances in the batch.

#### 1.8.3.3 hipblasXherkx + Batched, StridedBatched

*hipblasStatus\_t* **hipblasCherkx**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*BP, int ldb, const float \*beta, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZherkx**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*BP, int ldb, const double \*beta, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

herkx performs one of the matrix-matrix operations for a Hermitian rank-k update

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

where alpha and beta are scalars, op(A) and op(B) are n by k matrices, and C is a n x n Hermitian matrix stored as either upper or lower. This routine should only be used when the caller can guarantee that the result of op(A)\*op(B)^T will be Hermitian.

op( A ) = A, op( B ) = B, **and** A **and** B are n by k **if** trans == HIPBLAS\_OP\_N  
 op( A ) = A^H, op( B ) = B^H, **and** A **and** B are k by n **if** trans == HIPBLAS\_OP\_C

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_C: op( A ) = A^H, op( B ) = B^H HIPBLAS\_OP\_N: op( A ) = A, op( B ) = B
- **n** – [in] [int] n specifies the number of rows and columns of C. n >= 0.
- **k** – [in] [int] k specifies the number of columns of op(A). k >= 0.
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] pointer storing matrix A on the GPU. Martrix dimension is ( lda, k ) when if trans = HIPBLAS\_OP\_N, otherwise (lda, n) only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if trans = HIPBLAS\_OP\_N, lda >= max( 1, n ), otherwise lda >= max( 1, k ).
- **BP** – [in] pointer storing matrix B on the GPU. Martrix dimension is ( ldb, k ) when if trans = HIPBLAS\_OP\_N, otherwise (ldb, n) only the upper/lower triangular part is accessed.
- **ldb** – [in] [int] ldb specifies the first dimension of B. if trans = HIPBLAS\_OP\_N, ldb >= max( 1, n ), otherwise ldb >= max( 1, k ).
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] pointer storing matrix C on the GPU. The imaginary component of the diagonal elements are not used but are set to zero unless quick return.
- **ldc** – [in] [int] ldc specifies the first dimension of C. ldc >= max( 1, n ).

```
hipblasStatus_t hipblasCherkxBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, hipblasOperation_t
                                     transA, int n, int k, const hipblasComplex *alpha, const hipblasComplex
                                     *const AP[], int lda, const hipblasComplex *const BP[], int ldb, const
                                     float *beta, hipblasComplex *const CP[], int ldc, int batchCount)
```

```
hipblasStatus_t hipblasZherkxBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, hipblasOperation_t
                                     transA, int n, int k, const hipblasDoubleComplex *alpha, const
                                     hipblasDoubleComplex *const AP[], int lda, const
                                     hipblasDoubleComplex *const BP[], int ldb, const double *beta,
                                     hipblasDoubleComplex *const CP[], int ldc, int batchCount)
```

BLAS Level 3 API.

herkxBatched performs a batch of the matrix-matrix operations for a Hermitian rank-k update

$$C_i := \alpha * \text{op}(A_i) * \text{op}(B_i)^H + \beta * C_i$$

where alpha and beta are scalars, op(A\_i) and op(B\_i) are n by k matrices, and C\_i is a n x n Hermitian matrix stored as either upper or lower. This routine should only be used when the caller can guarantee that the result of op(A) \* op(B)^T will be Hermitian.

```
op( A_i ) = A_i, op( B_i ) = B_i, and A_i and B_i are n by k if trans == HIPBLAS_OP_
↪N
op( A_i ) = A_i^H, op( B_i ) = B_i^H, and A_i and B_i are k by n if trans ==
↪HIPBLAS_OP_C
```

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C\_i is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C\_i is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_C: op(A) = A^H HIPBLAS\_OP\_N: op(A) = A
- **n** – [in] [int] n specifies the number of rows and columns of C\_i. n >= 0.
- **k** – [in] [int] k specifies the number of columns of op(A). k >= 0.
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] device array of device pointers storing each matrix\_i A of dimension (lda, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if trans = HIPBLAS\_OP\_N, lda >= max( 1, n ), otherwise lda >= max( 1, k ).
- **BP** – [in] device array of device pointers storing each matrix\_i B of dimension (ldb, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (ldb, n)
- **ldb** – [in] [int] ldb specifies the first dimension of B\_i. if trans = HIPBLAS\_OP\_N, ldb >= max( 1, n ), otherwise ldb >= max( 1, k ).
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.

- **CP** – [in] device array of device pointers storing each matrix  $C_i$  on the GPU. The imaginary component of the diagonal elements are not used but are set to zero unless quick return.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $ldc \geq \max(1, n)$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasCherkxStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*BP, int ldb, *hipblasStride* strideB, const float \*beta, *hipblasComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasZherkxStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasDoubleComplex* \*BP, int ldb, *hipblasStride* strideB, const double \*beta, *hipblasDoubleComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

BLAS Level 3 API.

herkxStridedBatched performs a batch of the matrix-matrix operations for a Hermitian rank-k update

$$C_i := \alpha * \text{op}(A_i) * \text{op}(B_i)^H + \beta * C_i$$

where alpha and beta are scalars,  $\text{op}(A_i)$  and  $\text{op}(B_i)$  are  $n$  by  $k$  matrices, and  $C_i$  is a  $n \times n$  Hermitian matrix stored as either upper or lower. This routine should only be used when the caller can guarantee that the result of  $\text{op}(A) * \text{op}(B)^T$  will be Hermitian.

$\text{op}(A_i) = A_i$ ,  $\text{op}(B_i) = B_i$ , and  $A_i$  and  $B_i$  are  $n$  by  $k$  if  $\text{trans} == \text{HIPBLAS\_OP\_N}$   
 $\text{op}(A_i) = A_i^H$ ,  $\text{op}(B_i) = B_i^H$ , and  $A_i$  and  $B_i$  are  $k$  by  $n$  if  $\text{trans} == \text{HIPBLAS\_OP\_C}$

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER:  $C_i$  is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER:  $C_i$  is a lower triangular matrix
- **transA** – [in] [*hipblasOperation\_t*] HIPBLAS\_OP\_C:  $\text{op}(A_i) = A_i^H$ ,  $\text{op}(B_i) = B_i^H$  HIPBLAS\_OP\_N:  $\text{op}(A_i) = A_i$ ,  $\text{op}(B_i) = B_i$
- **n** – [in] [int]  $n$  specifies the number of rows and columns of  $C_i$ .  $n \geq 0$ .
- **k** – [in] [int]  $k$  specifies the number of columns of  $\text{op}(A)$ .  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] Device pointer to the first matrix  $A_1$  on the GPU of dimension (lda, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)

- **lda** – [in] [int] lda specifies the first dimension of A<sub>i</sub>. if trans = HIPBLAS\_OP\_N, lda ≥ max( 1, n ), otherwise lda ≥ max( 1, k ).
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A<sub>i</sub>) and the next one (A<sub>i+1</sub>)
- **BP** – [in] Device pointer to the first matrix B<sub>1</sub> on the GPU of dimension (ldb, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (ldb, n)
- **ldb** – [in] [int] ldb specifies the first dimension of B<sub>i</sub>. if trans = HIPBLAS\_OP\_N, ldb ≥ max( 1, n ), otherwise ldb ≥ max( 1, k ).
- **strideB** – [in] [hipblasStride] stride from the start of one matrix (B<sub>i</sub>) and the next one (B<sub>i+1</sub>)
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] Device pointer to the first matrix C<sub>1</sub> on the GPU. The imaginary component of the diagonal elements are not used but are set to zero unless quick return.
- **ldc** – [in] [int] ldc specifies the first dimension of C. ldc ≥ max( 1, n ).
- **strideC** – [inout] [hipblasStride] stride from the start of one matrix (C<sub>i</sub>) and the next one (C<sub>i+1</sub>)
- **batchCount** – [in] [int] number of instances in the batch.

#### 1.8.3.4 hipblasXher2k + Batched, StridedBatched

*hipblasStatus\_t* **hipblasCher2k**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*BP, int ldb, const float \*beta, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZher2k**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*BP, int ldb, const double \*beta, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

her2k performs one of the matrix-matrix operations for a Hermitian rank-2k update

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

where alpha and beta are scalars, op(A) and op(B) are n by k matrices, and C is a n x n Hermitian matrix stored as either upper or lower.

op( A ) = A, op( B ) = B, **and** A **and** B are n by k **if** trans == HIPBLAS\_OP\_N  
 op( A ) = A<sup>H</sup>, op( B ) = B<sup>H</sup>, **and** A **and** B are k by n **if** trans == HIPBLAS\_OP\_C

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.

- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_C:  $\text{op}(A) = A^H$ ,  $\text{op}(B) = B^H$  HIPBLAS\_OP\_N:  $\text{op}(A) = A$ ,  $\text{op}(B) = B$
- **n** – [in] [int] n specifies the number of rows and columns of C.  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of  $\text{op}(A)$ .  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] pointer storing matrix A on the GPU. Martrix dimension is (lda, k) when if trans = HIPBLAS\_OP\_N, otherwise (lda, n) only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if trans = HIPBLAS\_OP\_N,  $\text{lda} \geq \max(1, n)$ , otherwise  $\text{lda} \geq \max(1, k)$ .
- **BP** – [in] pointer storing matrix B on the GPU. Martrix dimension is (ldb, k) when if trans = HIPBLAS\_OP\_N, otherwise (ldb, n) only the upper/lower triangular part is accessed.
- **ldb** – [in] [int] ldb specifies the first dimension of B. if trans = HIPBLAS\_OP\_N,  $\text{ldb} \geq \max(1, n)$ , otherwise  $\text{ldb} \geq \max(1, k)$ .
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] pointer storing matrix C on the GPU. The imaginary component of the diagonal elements are not used but are set to zero unless quick return.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $\text{ldc} \geq \max(1, n)$ .

*hipblasStatus\_t* **hipblasCher2kBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const BP[], int ldb, const float \*beta, *hipblasComplex* \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasZher2kBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const BP[], int ldb, const double \*beta, *hipblasDoubleComplex* \*const CP[], int ldc, int batchSize)

BLAS Level 3 API.

her2kBatched performs a batch of the matrix-matrix operations for a Hermitian rank-2k update

$$C_i := \alpha * \text{op}(A_i) * \text{op}(B_i)^H + \text{conj}(\alpha) * \text{op}(B_i) * \text{op}(A_i)^H + \beta * C_i$$

where alpha and beta are scalars,  $\text{op}(A_i)$  and  $\text{op}(B_i)$  are n by k matrices, and  $C_i$  is a n x n Hermitian matrix stored as either upper or lower.

$\text{op}(A_i) = A_i$ ,  $\text{op}(B_i) = B_i$ , and  $A_i$  and  $B_i$  are n by k if trans == HIPBLAS\_OP\_N  
 $\text{op}(A_i) = A_i^H$ ,  $\text{op}(B_i) = B_i^H$ , and  $A_i$  and  $B_i$  are k by n if trans == HIPBLAS\_OP\_C

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support



## Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C<sub>i</sub> is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C<sub>i</sub> is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_C:  $\text{op}(A) = A^H$  HIPBLAS\_OP\_N:  $\text{op}(A) = A$
- **n** – [in] [int] n specifies the number of rows and columns of C<sub>i</sub>.  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of  $\text{op}(A)$ .  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] device array of device pointers storing each matrix<sub>i</sub> A of dimension (lda, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A<sub>i</sub>. if trans = HIPBLAS\_OP\_N,  $\text{lda} \geq \max(1, n)$ , otherwise  $\text{lda} \geq \max(1, k)$ .
- **BP** – [in] device array of device pointers storing each matrix<sub>i</sub> B of dimension (ldb, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (ldb, n)
- **ldb** – [in] [int] ldb specifies the first dimension of B<sub>i</sub>. if trans = HIPBLAS\_OP\_N,  $\text{ldb} \geq \max(1, n)$ , otherwise  $\text{ldb} \geq \max(1, k)$ .
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] device array of device pointers storing each matrix C<sub>i</sub> on the GPU. The imaginary component of the diagonal elements are not used but are set to zero unless quick return.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $\text{ldc} \geq \max(1, n)$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasCher2kStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*BP, int ldb, *hipblasStride* strideB, const float \*beta, *hipblasComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasZher2kStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasDoubleComplex* \*BP, int ldb, *hipblasStride* strideB, const double \*beta, *hipblasDoubleComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

BLAS Level 3 API.

her2kStridedBatched performs a batch of the matrix-matrix operations for a Hermitian rank-2k update

$$C_i := \alpha * \text{op}(A_i) * \text{op}(B_i)^H + \text{conj}(\alpha) * \text{op}(B_i) * \text{op}(A_i)^H + \beta * C_i$$

where alpha and beta are scalars,  $\text{op}(A_i)$  and  $\text{op}(B_i)$  are n by k matrices, and C<sub>i</sub> is a n x n Hermitian matrix stored as either upper or lower.



```

op( A_i ) = A_i, op( B_i ) = B_i, and A_i and B_i are n by k if trans == HIPBLAS_OP_
↪N
op( A_i ) = A_i^H, op( B_i ) = B_i^H, and A_i and B_i are k by n if trans == ↪
↪HIPBLAS_OP_C

```

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C\_i is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C\_i is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_C:  $\text{op}(A_i) = A_i^H$ ,  $\text{op}(B_i) = B_i^H$  HIPBLAS\_OP\_N:  $\text{op}(A_i) = A_i$ ,  $\text{op}(B_i) = B_i$
- **n** – [in] [int] n specifies the number of rows and columns of C\_i.  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of op(A).  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] Device pointer to the first matrix A\_1 on the GPU of dimension (lda, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if trans = HIPBLAS\_OP\_N,  $\text{lda} \geq \max(1, n)$ , otherwise  $\text{lda} \geq \max(1, k)$ .
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A\_i) and the next one (A\_i+1)
- **BP** – [in] Device pointer to the first matrix B\_1 on the GPU of dimension (ldb, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (ldb, n)
- **ldb** – [in] [int] ldb specifies the first dimension of B\_i. if trans = HIPBLAS\_OP\_N,  $\text{ldb} \geq \max(1, n)$ , otherwise  $\text{ldb} \geq \max(1, k)$ .
- **strideB** – [in] [hipblasStride] stride from the start of one matrix (B\_i) and the next one (B\_i+1)
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] Device pointer to the first matrix C\_1 on the GPU. The imaginary component of the diagonal elements are not used but are set to zero unless quick return.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $\text{ldc} \geq \max(1, n)$ .
- **strideC** – [inout] [hipblasStride] stride from the start of one matrix (C\_i) and the next one (C\_i+1)
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.3.5 hipblasXsymm + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSsymm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const float \*alpha, const float \*AP, int lda, const float \*BP, int ldb, const float \*beta, float \*CP, int ldc)

*hipblasStatus\_t* **hipblasDsymm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const double \*alpha, const double \*AP, int lda, const double \*BP, int ldb, const double \*beta, double \*CP, int ldc)

*hipblasStatus\_t* **hipblasCsymm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*BP, int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZsymm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*BP, int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

symm performs one of the matrix-matrix operations:

$C := \alpha * A * B + \beta * C$  if side == HIPBLAS\_SIDE\_LEFT,  $C := \alpha * B * A + \beta * C$  if side == HIPBLAS\_SIDE\_RIGHT,

where alpha and beta are scalars, B and C are m by n matrices, and A is a symmetric matrix stored as either upper or lower.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] HIPBLAS\_SIDE\_LEFT:  $C := \alpha * A * B + \beta * C$  HIPBLAS\_SIDE\_RIGHT:  $C := \alpha * B * A + \beta * C$
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: A is a lower triangular matrix
- **m** – [in] [int] m specifies the number of rows of B and C.  $m \geq 0$ .
- **n** – [in] [int] n specifies the number of columns of B and C.  $n \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A and B are not referenced.
- **AP** – [in] pointer storing matrix A on the GPU. A is m by m if side == HIPBLAS\_SIDE\_LEFT A is n by n if side == HIPBLAS\_SIDE\_RIGHT only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if side = HIPBLAS\_SIDE\_LEFT,  $lda \geq \max(1, m)$ , otherwise  $lda \geq \max(1, n)$ .
- **BP** – [in] pointer storing matrix B on the GPU. Matrix dimension is m by n

- **ldb** – [in] [int] ldb specifies the first dimension of B.  $ldb \geq \max(1, m)$
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] pointer storing matrix C on the GPU. Matrix dimension is m by n
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $ldc \geq \max(1, m)$

*hipblasStatus\_t* **hipblasSsymmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const float \*alpha, const float \*const AP[], int lda, const float \*const BP[], int ldb, const float \*beta, float \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasDsymmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const double \*alpha, const double \*const AP[], int lda, const double \*const BP[], int ldb, const double \*beta, double \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasCsymmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const BP[], int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasZsymmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const BP[], int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const CP[], int ldc, int batchCount)

BLAS Level 3 API.

symmBatched performs a batch of the matrix-matrix operations:

$C_i := \alpha * A_i * B_i + \beta * C_i$  if side == HIPBLAS\_SIDE\_LEFT,  $C_i := \alpha * B_i * A_i + \beta * C_i$  if side == HIPBLAS\_SIDE\_RIGHT,

where alpha and beta are scalars,  $B_i$  and  $C_i$  are m by n matrices, and  $A_i$  is a symmetric matrix stored as either upper or lower.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] HIPBLAS\_SIDE\_LEFT:  $C_i := \alpha * A_i * B_i + \beta * C_i$  HIPBLAS\_SIDE\_RIGHT:  $C_i := \alpha * B_i * A_i + \beta * C_i$
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER:  $A_i$  is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER:  $A_i$  is a lower triangular matrix
- **m** – [in] [int] m specifies the number of rows of  $B_i$  and  $C_i$ .  $m \geq 0$ .
- **n** – [in] [int] n specifies the number of columns of  $B_i$  and  $C_i$ .  $n \geq 0$ .

- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A\_i and B\_i are not referenced.
- **AP** – [in] device array of device pointers storing each matrix A\_i on the GPU. A\_i is m by m if side == HIPBLAS\_SIDE\_LEFT A\_i is n by n if side == HIPBLAS\_SIDE\_RIGHT only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if side = HIPBLAS\_SIDE\_LEFT, lda >= max( 1, m ), otherwise lda >= max( 1, n ).
- **BP** – [in] device array of device pointers storing each matrix B\_i on the GPU. Matrix dimension is m by n
- **ldb** – [in] [int] ldb specifies the first dimension of B\_i. ldb >= max( 1, m )
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C\_i need not be set before entry.
- **CP** – [in] device array of device pointers storing each matrix C\_i on the GPU. Matrix dimension is m by n
- **ldc** – [in] [int] ldc specifies the first dimension of C\_i. ldc >= max( 1, m )
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasSsymmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const float \*alpha, const float \*AP, int lda, *hipblasStride* strideA, const float \*BP, int ldb, *hipblasStride* strideB, const float \*beta, float \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasDsymmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const double \*alpha, const double \*AP, int lda, *hipblasStride* strideA, const double \*BP, int ldb, *hipblasStride* strideB, const double \*beta, double \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasCsymmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*BP, int ldb, *hipblasStride* strideB, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasZsymmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasDoubleComplex* \*BP, int ldb, *hipblasStride* strideB, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

BLAS Level 3 API.

symmStridedBatched performs a batch of the matrix-matrix operations:

$C_i := \alpha A_i B_i + \beta C_i$  if side == HIPBLAS\_SIDE\_LEFT,  $C_i := \alpha B_i A_i + \beta C_i$  if side == HIPBLAS\_SIDE\_RIGHT,

where alpha and beta are scalars, B\_i and C\_i are m by n matrices, and A\_i is a symmetric matrix stored as either upper or lower.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] HIPBLAS\_SIDE\_LEFT:  $C_i := \alpha A_i B_i + \beta C_i$  HIPBLAS\_SIDE\_RIGHT:  $C_i := \alpha B_i A_i + \beta C_i$
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A\_i is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: A\_i is a lower triangular matrix
- **m** – [in] [int] m specifies the number of rows of B\_i and C\_i.  $m \geq 0$ .
- **n** – [in] [int] n specifies the number of columns of B\_i and C\_i.  $n \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A\_i and B\_i are not referenced.
- **AP** – [in] device pointer to first matrix A\_1 A\_i is m by m if side == HIPBLAS\_SIDE\_LEFT A\_i is n by n if side == HIPBLAS\_SIDE\_RIGHT only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if side = HIPBLAS\_SIDE\_LEFT,  $lda \geq \max(1, m)$ , otherwise  $lda \geq \max(1, n)$ .
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A\_i) and the next one (A\_i+1)
- **BP** – [in] device pointer to first matrix B\_1 of dimension (ldb, n) on the GPU.
- **ldb** – [in] [int] ldb specifies the first dimension of B\_i.  $ldb \geq \max(1, m)$
- **strideB** – [in] [hipblasStride] stride from the start of one matrix (B\_i) and the next one (B\_i+1)
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] device pointer to first matrix C\_1 of dimension (ldc, n) on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $ldc \geq \max(1, m)$ .
- **strideC** – [inout] [hipblasStride] stride from the start of one matrix (C\_i) and the next one (C\_i+1)
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.3.6 hipblasXsyrrk + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSsyrrk**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const float \*AP, int lda, const float \*beta, float \*CP, int ldc)

*hipblasStatus\_t* **hipblasDsyrrk**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const double \*AP, int lda, const double \*beta, double \*CP, int ldc)

*hipblasStatus\_t* **hipblasCsyrrk**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZsyrrk**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

syrrk performs one of the matrix-matrix operations for a symmetric rank-k update

$$C := \alpha * \text{op}(A) * \text{op}(A)^T + \beta * C$$

where alpha and beta are scalars, op(A) is an n by k matrix, and C is a symmetric n x n matrix stored as either upper or lower.

op( A ) = A, and A is n by k if transA == HIPBLAS\_OP\_N  
op( A ) = A^T and A is k by n if transA == HIPBLAS\_OP\_T

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

HIPBLAS\_OP\_C is not supported for complex types, see cherk and zherk.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_T: op(A) = A^T HIPBLAS\_OP\_N: op(A) = A HIPBLAS\_OP\_C: op(A) = A^T
- **n** – [in] [int] n specifies the number of rows and columns of C. n >= 0.
- **k** – [in] [int] k specifies the number of columns of op(A). k >= 0.
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] pointer storing matrix A on the GPU. Martrix dimension is (lda, k) when if transA = HIPBLAS\_OP\_N, otherwise (lda, n) only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if transA = HIPBLAS\_OP\_N, lda >= max( 1, n ), otherwise lda >= max( 1, k ).

- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] pointer storing matrix C on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $\text{ldc} \geq \max(1, n)$ .

*hipblasStatus\_t* **hipblasSsyrrkBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const float \*const AP[], int lda, const float \*beta, float \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasDsyrrkBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const double \*const AP[], int lda, const double \*beta, double \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasCsyrrkBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*beta, *hipblasComplex* \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasZsyrrkBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const CP[], int ldc, int batchSize)

BLAS Level 3 API.

syrrkBatched performs a batch of the matrix-matrix operations for a symmetric rank-k update

$C_i := \alpha * \text{op}(A_i) * \text{op}(A_i)^T + \beta * C_i$

where alpha and beta are scalars,  $\text{op}(A_i)$  is an n by k matrix, and  $C_i$  is a symmetric n x n matrix stored as either upper or lower.

$\text{op}(A_i) = A_i$ , and  $A_i$  is n by k if  $\text{transA} == \text{HIPBLAS\_OP\_N}$   
 $\text{op}(A_i) = A_i^T$  and  $A_i$  is k by n if  $\text{transA} == \text{HIPBLAS\_OP\_T}$

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

HIPBLAS\_OP\_C is not supported for complex types, see cherk and zherk.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER:  $C_i$  is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER:  $C_i$  is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_T:  $\text{op}(A) = A^T$  HIPBLAS\_OP\_N:  $\text{op}(A) = A$  HIPBLAS\_OP\_C:  $\text{op}(A) = A^T$
- **n** – [in] [int] n specifies the number of rows and columns of  $C_i$ .  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of  $\text{op}(A)$ .  $k \geq 0$ .

- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] device array of device pointers storing each matrix  $A_i$  of dimension (lda, k) when transA is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of  $A_i$ . if transA = HIPBLAS\_OP\_N, lda  $\geq \max(1, n)$ , otherwise lda  $\geq \max(1, k)$ .
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] device array of device pointers storing each matrix  $C_i$  on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C. ldc  $\geq \max(1, n)$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasSsyrkStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const float \*AP, int lda, *hipblasStride* strideA, const float \*beta, float \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasDsyrkStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const double \*AP, int lda, *hipblasStride* strideA, const double \*beta, double \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasCsyrkStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasZsyrkStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

BLAS Level 3 API.

syrkStridedBatched performs a batch of the matrix-matrix operations for a symmetric rank-k update

$$C_i := \alpha * \text{op}(A_i) * \text{op}(A_i)^T + \beta * C_i$$

where alpha and beta are scalars, op( $A_i$ ) is an n by k matrix, and  $C_i$  is a symmetric n x n matrix stored as either upper or lower.

op(  $A_i$  ) =  $A_i$ , and  $A_i$  is n by k if transA == HIPBLAS\_OP\_N  
 op(  $A_i$  ) =  $A_i^T$  and  $A_i$  is k by n if transA == HIPBLAS\_OP\_T

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support



HIPBLAS\_OP\_C is not supported for complex types, see `cherk` and `zherk`.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C\_i is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C\_i is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_T:  $op(A) = A^T$  HIPBLAS\_OP\_N:  $op(A) = A$  HIPBLAS\_OP\_C:  $op(A) = A^T$
- **n** – [in] [int] n specifies the number of rows and columns of C\_i.  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of op(A).  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] Device pointer to the first matrix A\_1 on the GPU of dimension (lda, k) when transA is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if transA = HIPBLAS\_OP\_N,  $lda \geq \max(1, n)$ , otherwise  $lda \geq \max(1, k)$ .
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A\_i) and the next one (A\_i+1)
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] Device pointer to the first matrix C\_1 on the GPU. on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $ldc \geq \max(1, n)$ .
- **strideC** – [inout] [hipblasStride] stride from the start of one matrix (C\_i) and the next one (C\_i+1)
- **batchCount** – [in] [int] number of instances in the batch.

#### 1.8.3.7 hipblasXsyr2k + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSsyr2k**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const float \*AP, int lda, const float \*BP, int ldb, const float \*beta, float \*CP, int ldc)

*hipblasStatus\_t* **hipblasDsyr2k**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const double \*AP, int lda, const double \*BP, int ldb, const double \*beta, double \*CP, int ldc)

*hipblasStatus\_t* **hipblasCsyr2k**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*BP, int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZsyr2k**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*BP, int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

syr2k performs one of the matrix-matrix operations for a symmetric rank-2k update

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

where alpha and beta are scalars, op(A) and op(B) are n by k matrix, and C is a symmetric n x n matrix stored as either upper or lower.

op( A ) = A, op( B ) = B, **and** A **and** B are n by k **if** trans == HIPBLAS\_OP\_N  
 op( A ) = A<sup>T</sup>, op( B ) = B<sup>T</sup>, **and** A **and** B are k by n **if** trans == HIPBLAS\_OP\_T

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER: C is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C is a lower triangular matrix
- **transA** – [in] [*hipblasOperation\_t*] HIPBLAS\_OP\_T: op( A ) = A<sup>T</sup>, op( B ) = B<sup>T</sup> HIPBLAS\_OP\_N: op( A ) = A, op( B ) = B
- **n** – [in] [int] n specifies the number of rows and columns of C. n >= 0.
- **k** – [in] [int] k specifies the number of columns of op(A) and op(B). k >= 0.
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] pointer storing matrix A on the GPU. Martrix dimension is ( lda, k ) when if trans = HIPBLAS\_OP\_N, otherwise (lda, n) only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if trans = HIPBLAS\_OP\_N, lda >= max( 1, n ), otherwise lda >= max( 1, k ).
- **BP** – [in] pointer storing matrix B on the GPU. Martrix dimension is ( ldb, k ) when if trans = HIPBLAS\_OP\_N, otherwise (ldb, n) only the upper/lower triangular part is accessed.
- **ldb** – [in] [int] ldb specifies the first dimension of B. if trans = HIPBLAS\_OP\_N, ldb >= max( 1, n ), otherwise ldb >= max( 1, k ).
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] pointer storing matrix C on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C. ldc >= max( 1, n ).

*hipblasStatus\_t* **hipblasSsyr2kBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const float \*const AP[], int lda, const float \*const BP[], int ldb, const float \*beta, float \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasDsyrr2kBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const double \*const AP[], int lda, const double \*const BP[], int ldb, const double \*beta, double \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasCsyrr2kBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const BP[], int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasZsyrr2kBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const BP[], int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const CP[], int ldc, int batchCount)

BLAS Level 3 API.

syrr2kBatched performs a batch of the matrix-matrix operations for a symmetric rank-2k update

$$C_i := \alpha * (\text{op}(A_i) * \text{op}(B_i)^T + \text{op}(B_i) * \text{op}(A_i)^T) + \beta * C_i$$

where alpha and beta are scalars, op(A\_i) and op(B\_i) are n by k matrix, and C\_i is a symmetric n x n matrix stored as either upper or lower.

op( A\_i ) = A\_i, op( B\_i ) = B\_i, **and** A\_i **and** B\_i are n by k **if** trans == HIPBLAS\_OP\_↪N  
 op( A\_i ) = A\_i^T, op( B\_i ) = B\_i^T, **and** A\_i **and** B\_i are k by n **if** trans == ↪HIPBLAS\_OP\_T

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C\_i is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C\_i is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_T: op( A\_i ) = A\_i^T, op( B\_i ) = B\_i^T HIPBLAS\_OP\_N: op( A\_i ) = A\_i, op( B\_i ) = B\_i
- **n** – [in] [int] n specifies the number of rows and columns of C\_i. n >= 0.
- **k** – [in] [int] k specifies the number of columns of op(A). k >= 0.
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] device array of device pointers storing each matrix\_i A of dimension (lda, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if trans = HIPBLAS\_OP\_N, lda >= max( 1, n ), otherwise lda >= max( 1, k ).

- **BP** – [in] device array of device pointers storing each matrix  $B_i$  of dimension (ldb, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (ldb, n)
- **ldb** – [in] [int] ldb specifies the first dimension of B. if trans = HIPBLAS\_OP\_N,  $ldb \geq \max(1, n)$ , otherwise  $ldb \geq \max(1, k)$ .
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] device array of device pointers storing each matrix  $C_i$  on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $ldc \geq \max(1, n)$ .
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasSsyr2kStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const float \*AP, int lda, *hipblasStride* strideA, const float \*BP, int ldb, *hipblasStride* strideB, const float \*beta, float \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasDsyr2kStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const double \*AP, int lda, *hipblasStride* strideA, const double \*BP, int ldb, *hipblasStride* strideB, const double \*beta, double \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasCsyr2kStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*BP, int ldb, *hipblasStride* strideB, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasZsyr2kStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasDoubleComplex* \*BP, int ldb, *hipblasStride* strideB, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

BLAS Level 3 API.

syr2kStridedBatched performs a batch of the matrix-matrix operations for a symmetric rank-2k update

$$C_i := \alpha * (\text{op}(A_i) * \text{op}(B_i)^T + \text{op}(B_i) * \text{op}(A_i)^T) + \beta * C_i$$

where alpha and beta are scalars, op( $A_i$ ) and op( $B_i$ ) are n by k matrix, and  $C_i$  is a symmetric n x n matrix stored as either upper or lower.

op(  $A_i$  ) =  $A_i$ , op(  $B_i$  ) =  $B_i$ , and  $A_i$  and  $B_i$  are n by k if trans == HIPBLAS\_OP\_N  
 ↪ N  
 op(  $A_i$  ) =  $A_i^T$ , op(  $B_i$  ) =  $B_i^T$ , and  $A_i$  and  $B_i$  are k by n if trans ==  
 ↪ HIPBLAS\_OP\_T

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C<sub>i</sub> is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C<sub>i</sub> is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_T: op( A<sub>i</sub> ) = A<sub>i</sub><sup>T</sup>, op( B<sub>i</sub> ) = B<sub>i</sub><sup>T</sup> HIPBLAS\_OP\_N: op( A<sub>i</sub> ) = A<sub>i</sub>, op( B<sub>i</sub> ) = B<sub>i</sub>
- **n** – [in] [int] n specifies the number of rows and columns of C<sub>i</sub>. n >= 0.
- **k** – [in] [int] k specifies the number of columns of op(A). k >= 0.
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] Device pointer to the first matrix A<sub>1</sub> on the GPU of dimension (lda, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A<sub>i</sub>. if trans = HIPBLAS\_OP\_N, lda >= max( 1, n ), otherwise lda >= max( 1, k ).
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A<sub>i</sub>) and the next one (A<sub>i</sub>+1)
- **BP** – [in] Device pointer to the first matrix B<sub>1</sub> on the GPU of dimension (ldb, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (ldb, n)
- **ldb** – [in] [int] ldb specifies the first dimension of B<sub>i</sub>. if trans = HIPBLAS\_OP\_N, ldb >= max( 1, n ), otherwise ldb >= max( 1, k ).
- **strideB** – [in] [hipblasStride] stride from the start of one matrix (B<sub>i</sub>) and the next one (B<sub>i</sub>+1)
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] Device pointer to the first matrix C<sub>1</sub> on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C. ldc >= max( 1, n ).
- **strideC** – [inout] [hipblasStride] stride from the start of one matrix (C<sub>i</sub>) and the next one (C<sub>i</sub>+1)
- **batchCount** – [in] [int] number of instances in the batch.

#### 1.8.3.8 hipblasXsyrkx + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSsyrkx**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const float \*AP, int lda, const float \*BP, int ldb, const float \*beta, float \*CP, int ldc)

*hipblasStatus\_t* **hipblasDsyrkx**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const double \*AP, int lda, const double \*BP, int ldb, const double \*beta, double \*CP, int ldc)

*hipblasStatus\_t* **hipblasCsyrkx**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*BP, int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZsyrkx**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*BP, int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

syrkx performs one of the matrix-matrix operations for a symmetric rank-k update

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

where alpha and beta are scalars, op(A) and op(B) are n by k matrix, and C is a symmetric n x n matrix stored as either upper or lower. This routine should only be used when the caller can guarantee that the result of op(A) \* op(B)^T will be symmetric.

op( A ) = A, op( B ) = B, and A and B are n by k if trans == HIPBLAS\_OP\_N  
 op( A ) = A^T, op( B ) = B^T, and A and B are k by n if trans == HIPBLAS\_OP\_T

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_T: op( A ) = A^T, op( B ) = B^T HIPBLAS\_OP\_N: op( A ) = A, op( B ) = B
- **n** – [in] [int] n specifies the number of rows and columns of C. n >= 0.
- **k** – [in] [int] k specifies the number of columns of op(A) and op(B). k >= 0.
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] pointer storing matrix A on the GPU. Martrix dimension is ( lda, k ) when if trans = HIPBLAS\_OP\_N, otherwise (lda, n) only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if trans = HIPBLAS\_OP\_N, lda >= max( 1, n ), otherwise lda >= max( 1, k ).
- **BP** – [in] pointer storing matrix B on the GPU. Martrix dimension is ( ldb, k ) when if trans = HIPBLAS\_OP\_N, otherwise (ldb, n) only the upper/lower triangular part is accessed.
- **ldb** – [in] [int] ldb specifies the first dimension of B. if trans = HIPBLAS\_OP\_N, ldb >= max( 1, n ), otherwise ldb >= max( 1, k ).
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] pointer storing matrix C on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C. ldc >= max( 1, n ).

*hipblasStatus\_t* **hipblasSsyrrkxBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const float \*const AP[], int lda, const float \*const BP[], int ldb, const float \*beta, float \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasDsyrrkxBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const double \*const AP[], int lda, const double \*const BP[], int ldb, const double \*beta, double \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasCsyrrkxBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const BP[], int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasZsyrrkxBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const BP[], int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const CP[], int ldc, int batchCount)

BLAS Level 3 API.

syrrkxBatched performs a batch of the matrix-matrix operations for a symmetric rank-k update

$$C\_i := \alpha * \text{op}(A\_i) * \text{op}(B\_i)^T + \beta * C\_i$$

where alpha and beta are scalars, op(A\_i) and op(B\_i) are n by k matrix, and C\_i is a symmetric n x n matrix stored as either upper or lower. This routine should only be used when the caller can guarantee that the result of op(A\_i) \* op(B\_i)^T will be symmetric.

op( A\_i ) = A\_i, op( B\_i ) = B\_i, **and** A\_i **and** B\_i are n by k **if** trans == HIPBLAS\_OP\_↪N  
 op( A\_i ) = A\_i^T, op( B\_i ) = B\_i^T, **and** A\_i **and** B\_i are k by n **if** trans == ↪HIPBLAS\_OP\_T

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C\_i is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C\_i is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_T: op( A\_i ) = A\_i^T, op( B\_i ) = B\_i^T HIPBLAS\_OP\_N: op( A\_i ) = A\_i, op( B\_i ) = B\_i
- **n** – [in] [int] n specifies the number of rows and columns of C\_i. n >= 0.
- **k** – [in] [int] k specifies the number of columns of op(A). k >= 0.



- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] device array of device pointers storing each matrix\_i A of dimension (lda, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if trans = HIPBLAS\_OP\_N, lda >= max( 1, n ), otherwise lda >= max( 1, k ).
- **BP** – [in] device array of device pointers storing each matrix\_i B of dimension (ldb, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (ldb, n)
- **ldb** – [in] [int] ldb specifies the first dimension of B. if trans = HIPBLAS\_OP\_N, ldb >= max( 1, n ), otherwise ldb >= max( 1, k ).
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] device array of device pointers storing each matrix C\_i on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C. ldc >= max( 1, n ).
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasSyrkxStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const float \*alpha, const float \*AP, int lda, *hipblasStride* strideA, const float \*BP, int ldb, *hipblasStride* strideB, const float \*beta, float \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasDsykxStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const double \*alpha, const double \*AP, int lda, *hipblasStride* strideA, const double \*BP, int ldb, *hipblasStride* strideB, const double \*beta, double \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasCsykxStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*BP, int ldb, *hipblasStride* strideB, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasZsykxStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasDoubleComplex* \*BP, int ldb, *hipblasStride* strideB, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

BLAS Level 3 API.

syrkxStridedBatched performs a batch of the matrix-matrix operations for a symmetric rank-k update

$$C_i := \alpha * \text{op}(A_i) * \text{op}(B_i)^T + \beta * C_i$$

where alpha and beta are scalars, op(A\_i) and op(B\_i) are n by k matrix, and C\_i is a symmetric n x n matrix stored as either upper or lower. This routine should only be used when the caller can guarantee that the result of



$\text{op}(A_i) * \text{op}(B_i)^T$  will be symmetric.

```

op( A_i ) = A_i, op( B_i ) = B_i, and A_i and B_i are n by k if trans == HIPBLAS_OP_
↪N
op( A_i ) = A_i^T, op( B_i ) = B_i^T, and A_i and B_i are k by n if trans ==_
↪HIPBLAS_OP_T

```

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: C\_i is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: C\_i is a lower triangular matrix
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_T:  $\text{op}(A_i) = A_i^T$ ,  $\text{op}(B_i) = B_i^T$  HIPBLAS\_OP\_N:  $\text{op}(A_i) = A_i$ ,  $\text{op}(B_i) = B_i$
- **n** – [in] [int] n specifies the number of rows and columns of C\_i.  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of op(A).  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A is not referenced and A need not be set before entry.
- **AP** – [in] Device pointer to the first matrix A\_1 on the GPU of dimension (lda, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (lda, n)
- **lda** – [in] [int] lda specifies the first dimension of A\_i. if trans = HIPBLAS\_OP\_N,  $\text{lda} \geq \max(1, n)$ , otherwise  $\text{lda} \geq \max(1, k)$ .
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A\_i) and the next one (A\_{i+1})
- **BP** – [in] Device pointer to the first matrix B\_1 on the GPU of dimension (ldb, k) when trans is HIPBLAS\_OP\_N, otherwise of dimension (ldb, n)
- **ldb** – [in] [int] ldb specifies the first dimension of B\_i. if trans = HIPBLAS\_OP\_N,  $\text{ldb} \geq \max(1, n)$ , otherwise  $\text{ldb} \geq \max(1, k)$ .
- **strideB** – [in] [hipblasStride] stride from the start of one matrix (B\_i) and the next one (B\_{i+1})
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] Device pointer to the first matrix C\_1 on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $\text{ldc} \geq \max(1, n)$ .
- **strideC** – [inout] [hipblasStride] stride from the start of one matrix (C\_i) and the next one (C\_{i+1})
- **batchCount** – [in] [int] number of instances in the batch.

### 1.8.3.9 hipblasXgeam + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSgeam**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const float \*alpha, const float \*AP, int lda, const float \*beta, const float \*BP, int ldb, float \*CP, int ldc)

*hipblasStatus\_t* **hipblasDgeam**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const double \*alpha, const double \*AP, int lda, const double \*beta, const double \*BP, int ldb, double \*CP, int ldc)

*hipblasStatus\_t* **hipblasCgeam**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*beta, const *hipblasComplex* \*BP, int ldb, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZgeam**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*beta, const *hipblasDoubleComplex* \*BP, int ldb, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

geam performs one of the matrix-matrix operations

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

where  $\text{op}(X)$  is one of

$\text{op}(X) = X$             **or**  
 $\text{op}(X) = X^{**T}$        **or**  
 $\text{op}(X) = X^{**H},$

alpha and beta are scalars, and A, B and C are matrices, with  $\text{op}(A)$  an m by n matrix,  $\text{op}(B)$  an m by n matrix, and C an m by n matrix.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **transA** – [in] [*hipblasOperation\_t*] specifies the form of  $\text{op}(A)$
- **transB** – [in] [*hipblasOperation\_t*] specifies the form of  $\text{op}(B)$
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **alpha** – [in] device pointer or host pointer specifying the scalar alpha.
- **AP** – [in] device pointer storing matrix A.
- **lda** – [in] [int] specifies the leading dimension of A.
- **beta** – [in] device pointer or host pointer specifying the scalar beta.

- **BP** – [in] device pointer storing matrix B.
- **ldb** – [in] [int] specifies the leading dimension of B.
- **CP** – [inout] device pointer storing matrix C.
- **ldc** – [in] [int] specifies the leading dimension of C.

*hipblasStatus\_t* **hipblasSgeamBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const float \*alpha, const float \*const AP[], int lda, const float \*beta, const float \*const BP[], int ldb, float \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasDgeamBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const double \*alpha, const double \*const AP[], int lda, const double \*beta, const double \*const BP[], int ldb, double \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasCgeamBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*beta, const *hipblasComplex* \*const BP[], int ldb, *hipblasComplex* \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasZgeamBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*beta, const *hipblasDoubleComplex* \*const BP[], int ldb, *hipblasDoubleComplex* \*const CP[], int ldc, int batchSize)

BLAS Level 3 API.

geamBatched performs one of the batched matrix-matrix operations

$$C_i = \alpha * op(A_i) + \beta * op(B_i) \quad \text{for } i = 0, 1, \dots, \text{batchCount} - 1$$

where alpha and beta are scalars, and op(A<sub>i</sub>), op(B<sub>i</sub>) and C<sub>i</sub> are m by n matrices and op(X) is one of

$$\begin{aligned} op(X) &= X && \text{or} \\ op(X) &= X^{**T} \end{aligned}$$

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **transA** – [in] [hipblasOperation\_t] specifies the form of op(A)
- **transB** – [in] [hipblasOperation\_t] specifies the form of op(B)
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **alpha** – [in] device pointer or host pointer specifying the scalar alpha.

- **AP** – [in] device array of device pointers storing each matrix  $A_i$  on the GPU. Each  $A_i$  is of dimension ( lda, k ), where k is m when transA == HIPBLAS\_OP\_N and is n when transA == HIPBLAS\_OP\_T.
- **lda** – [in] [int] specifies the leading dimension of A.
- **beta** – [in] device pointer or host pointer specifying the scalar beta.
- **BP** – [in] device array of device pointers storing each matrix  $B_i$  on the GPU. Each  $B_i$  is of dimension ( ldb, k ), where k is m when transB == HIPBLAS\_OP\_N and is n when transB == HIPBLAS\_OP\_T.
- **ldb** – [in] [int] specifies the leading dimension of B.
- **CP** – [inout] device array of device pointers storing each matrix  $C_i$  on the GPU. Each  $C_i$  is of dimension ( ldc, n ).
- **ldc** – [in] [int] specifies the leading dimension of C.
- **batchCount** – [in] [int] number of instances i in the batch.

*hipblasStatus\_t* **hipblasSgeamStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const float \*alpha, const float \*AP, int lda, *hipblasStride* strideA, const float \*beta, const float \*BP, int ldb, *hipblasStride* strideB, float \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasDgeamStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const double \*alpha, const double \*AP, int lda, *hipblasStride* strideA, const double \*beta, const double \*BP, int ldb, *hipblasStride* strideB, double \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasCgeamStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*beta, const *hipblasComplex* \*BP, int ldb, *hipblasStride* strideB, *hipblasComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasZgeamStridedBatched**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasDoubleComplex* \*beta, const *hipblasDoubleComplex* \*BP, int ldb, *hipblasStride* strideB, *hipblasDoubleComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

BLAS Level 3 API.

geamStridedBatched performs one of the batched matrix-matrix operations

$$C_i = \alpha * \text{op}(A_i) + \beta * \text{op}(B_i) \quad \text{for } i = 0, 1, \dots, \text{batchCount} - 1$$

where alpha and beta are scalars, and op( $A_i$ ), op( $B_i$ ) and  $C_i$  are m by n matrices and op( X ) is one of

$$\text{op}(X) = X \quad \text{or}$$

$$\text{op}(X) = X^{**T}$$

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **transA** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(A)$
- **transB** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(B)$
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **alpha** – [in] device pointer or host pointer specifying the scalar alpha.
- **AP** – [in] device pointer to the first matrix  $A_0$  on the GPU. Each  $A_i$  is of dimension (lda, k), where k is m when  $\text{transA} == \text{HIPBLAS\_OP\_N}$  and is n when  $\text{transA} == \text{HIPBLAS\_OP\_T}$ .
- **lda** – [in] [int] specifies the leading dimension of A.
- **strideA** – [in] [hipblasStride] stride from the start of one matrix ( $A_i$ ) and the next one ( $A_{i+1}$ )
- **beta** – [in] device pointer or host pointer specifying the scalar beta.
- **BP** – [in] pointer to the first matrix  $B_0$  on the GPU. Each  $B_i$  is of dimension (ldb, k), where k is m when  $\text{transB} == \text{HIPBLAS\_OP\_N}$  and is n when  $\text{transB} == \text{HIPBLAS\_OP\_T}$ .
- **ldb** – [in] [int] specifies the leading dimension of B.
- **strideB** – [in] [hipblasStride] stride from the start of one matrix ( $B_i$ ) and the next one ( $B_{i+1}$ )
- **CP** – [inout] pointer to the first matrix  $C_0$  on the GPU. Each  $C_i$  is of dimension (ldc, n).
- **ldc** – [in] [int] specifies the leading dimension of C.
- **strideC** – [in] [hipblasStride] stride from the start of one matrix ( $C_i$ ) and the next one ( $C_{i+1}$ )
- **batchCount** – [in] [int] number of instances i in the batch.

#### 1.8.3.10 hipblasXhemm + Batched, StridedBatched

*hipblasStatus\_t* **hipblasChemmm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*BP, int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZhemmm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*BP, int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

hemm performs one of the matrix-matrix operations:

$C := \alpha * A * B + \beta * C$  if side == HIPBLAS\_SIDE\_LEFT,  $C := \alpha * B * A + \beta * C$  if side == HIPBLAS\_SIDE\_RIGHT,

where alpha and beta are scalars, B and C are m by n matrices, and A is a Hermitian matrix stored as either upper or lower.

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] HIPBLAS\_SIDE\_LEFT:  $C := \alpha * A * B + \beta * C$  HIPBLAS\_SIDE\_RIGHT:  $C := \alpha * B * A + \beta * C$
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER: A is a lower triangular matrix
- **n** – [in] [int] n specifies the number of rows of B and C.  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of B and C.  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then A and B are not referenced.
- **AP** – [in] pointer storing matrix A on the GPU. A is m by m if side == HIPBLAS\_SIDE\_LEFT A is n by n if side == HIPBLAS\_SIDE\_RIGHT Only the upper/lower triangular part is accessed. The imaginary component of the diagonal elements is not used.
- **lda** – [in] [int] lda specifies the first dimension of A. if side = HIPBLAS\_SIDE\_LEFT,  $lda \geq \max(1, m)$ , otherwise  $lda \geq \max(1, n)$ .
- **BP** – [in] pointer storing matrix B on the GPU. Matrix dimension is m by n
- **ldb** – [in] [int] ldb specifies the first dimension of B.  $ldb \geq \max(1, m)$
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] pointer storing matrix C on the GPU. Matrix dimension is m by n
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $ldc \geq \max(1, m)$

*hipblasStatus\_t* **hipblasChemmmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const BP[], int ldb, const *hipblasComplex* \*beta, *hipblasComplex* \*const CP[], int ldc, int batchSize)

*hipblasStatus\_t* **hipblasZhemmmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const BP[], int ldb, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*const CP[], int ldc, int batchSize)

BLAS Level 3 API.

hemmBatched performs a batch of the matrix-matrix operations:

$C_i := \alpha * A_i * B_i + \beta * C_i$  if `side == HIPBLAS_SIDE_LEFT`,  $C_i := \alpha * B_i * A_i + \beta * C_i$  if `side == HIPBLAS_SIDE_RIGHT`,

where  $\alpha$  and  $\beta$  are scalars,  $B_i$  and  $C_i$  are  $m$  by  $n$  matrices, and  $A_i$  is a Hermitian matrix stored as either upper or lower.

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] HIPBLAS\_SIDE\_LEFT:  $C_i := \alpha * A_i * B_i + \beta * C_i$  HIPBLAS\_SIDE\_RIGHT:  $C_i := \alpha * B_i * A_i + \beta * C_i$
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER:  $A_i$  is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER:  $A_i$  is a lower triangular matrix
- **n** – [in] [int]  $n$  specifies the number of rows of  $B_i$  and  $C_i$ .  $n \geq 0$ .
- **k** – [in] [int]  $k$  specifies the number of columns of  $B_i$  and  $C_i$ .  $k \geq 0$ .
- **alpha** – [in]  $\alpha$  specifies the scalar  $\alpha$ . When  $\alpha$  is zero then  $A_i$  and  $B_i$  are not referenced.
- **AP** – [in] device array of device pointers storing each matrix  $A_i$  on the GPU.  $A_i$  is  $m$  by  $m$  if `side == HIPBLAS_SIDE_LEFT`  $A_i$  is  $n$  by  $n$  if `side == HIPBLAS_SIDE_RIGHT` Only the upper/lower triangular part is accessed. The imaginary component of the diagonal elements is not used.
- **lda** – [in] [int]  $lda$  specifies the first dimension of  $A_i$ . if `side = HIPBLAS_SIDE_LEFT`,  $lda \geq \max(1, m)$ , otherwise  $lda \geq \max(1, n)$ .
- **BP** – [in] device array of device pointers storing each matrix  $B_i$  on the GPU. Matrix dimension is  $m$  by  $n$
- **ldb** – [in] [int]  $ldb$  specifies the first dimension of  $B_i$ .  $ldb \geq \max(1, m)$
- **beta** – [in]  $\beta$  specifies the scalar  $\beta$ . When  $\beta$  is zero then  $C_i$  need not be set before entry.
- **CP** – [in] device array of device pointers storing each matrix  $C_i$  on the GPU. Matrix dimension is  $m$  by  $n$
- **ldc** – [in] [int]  $ldc$  specifies the first dimension of  $C_i$ .  $ldc \geq \max(1, m)$
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasChemmmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasComplex* \*BP, int ldb, *hipblasStride* strideB, const *hipblasComplex* \*beta, *hipblasComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

*hipblasStatus\_t* **hipblasZhemmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, int n, int k, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, const *hipblasDoubleComplex* \*BP, int ldb, *hipblasStride* strideB, const *hipblasDoubleComplex* \*beta, *hipblasDoubleComplex* \*CP, int ldc, *hipblasStride* strideC, int batchCount)

BLAS Level 3 API.

hemmStridedBatched performs a batch of the matrix-matrix operations:

$C_i := \alpha * A_i * B_i + \beta * C_i$  if side == HIPBLAS\_SIDE\_LEFT,  $C_i := \alpha * B_i * A_i + \beta * C_i$  if side == HIPBLAS\_SIDE\_RIGHT,

where alpha and beta are scalars,  $B_i$  and  $C_i$  are m by n matrices, and  $A_i$  is a Hermitian matrix stored as either upper or lower.

- Supported precisions in rocBLAS : c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **side** – [in] [*hipblasSideMode\_t*] HIPBLAS\_SIDE\_LEFT:  $C_i := \alpha * A_i * B_i + \beta * C_i$  HIPBLAS\_SIDE\_RIGHT:  $C_i := \alpha * B_i * A_i + \beta * C_i$
- **uplo** – [in] [*hipblasFillMode\_t*] HIPBLAS\_FILL\_MODE\_UPPER:  $A_i$  is an upper triangular matrix HIPBLAS\_FILL\_MODE\_LOWER:  $A_i$  is a lower triangular matrix
- **n** – [in] [int] n specifies the number of rows of  $B_i$  and  $C_i$ .  $n \geq 0$ .
- **k** – [in] [int] k specifies the number of columns of  $B_i$  and  $C_i$ .  $k \geq 0$ .
- **alpha** – [in] alpha specifies the scalar alpha. When alpha is zero then  $A_i$  and  $B_i$  are not referenced.
- **AP** – [in] device pointer to first matrix  $A_1$   $A_i$  is m by m if side == HIPBLAS\_SIDE\_LEFT  $A_i$  is n by n if side == HIPBLAS\_SIDE\_RIGHT Only the upper/lower triangular part is accessed. The imaginary component of the diagonal elements is not used.
- **lda** – [in] [int] lda specifies the first dimension of  $A_i$ . if side = HIPBLAS\_SIDE\_LEFT,  $lda \geq \max(1, m)$ , otherwise  $lda \geq \max(1, n)$ .
- **strideA** – [in] [*hipblasStride*] stride from the start of one matrix ( $A_i$ ) and the next one ( $A_{i+1}$ )
- **BP** – [in] device pointer to first matrix  $B_1$  of dimension (ldb, n) on the GPU
- **ldb** – [in] [int] ldb specifies the first dimension of  $B_i$ . if side = HIPBLAS\_OP\_N,  $ldb \geq \max(1, m)$ , otherwise  $ldb \geq \max(1, n)$ .
- **strideB** – [in] [*hipblasStride*] stride from the start of one matrix ( $B_i$ ) and the next one ( $B_{i+1}$ )
- **beta** – [in] beta specifies the scalar beta. When beta is zero then C need not be set before entry.
- **CP** – [in] device pointer to first matrix  $C_1$  of dimension (ldc, n) on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C.  $ldc \geq \max(1, m)$



- **strideC** – [inout] [hipblasStride] stride from the start of one matrix (C<sub>i</sub>) and the next one (C<sub>i+1</sub>)
- **batchCount** – [in] [int] number of instances in the batch

### 1.8.3.11 hipblasXtrmm + Batched, StridedBatched

*hipblasStatus\_t* **hipblasStrmm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const float \*alpha, const float \*AP, int lda, float \*BP, int ldb)

*hipblasStatus\_t* **hipblasDtrmm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const double \*alpha, const double \*AP, int lda, double \*BP, int ldb)

*hipblasStatus\_t* **hipblasCtrmm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasComplex* \*BP, int ldb)

*hipblasStatus\_t* **hipblasZtrmm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasDoubleComplex* \*BP, int ldb)

BLAS Level 3 API.

trmm performs one of the matrix-matrix operations

$B := \alpha * \text{op}(A) * B$ , or  $B := \alpha * B * \text{op}(A)$

where alpha is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

$\text{op}(A) = A$     **or**     $\text{op}(A) = A^T$     **or**     $\text{op}(A) = A^H$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

When uplo == HIPBLAS\_FILL\_MODE\_UPPER the leading k by 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.

When uplo == HIPBLAS\_FILL\_MODE\_LOWER the leading k by 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 == HIPBLAS\_DIAG\_UNIT the diagonal elements of A are not referenced either, but are assumed to be unity.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.

- **side** – [in] [hipblasSideMode\_t] Specifies whether  $\text{op}(A)$  multiplies  $B$  from the left or right as follows: HIPBLAS\_SIDE\_LEFT:  $B := \alpha * \text{op}(A) * B$ . HIPBLAS\_SIDE\_RIGHT:  $B := \alpha * B * \text{op}(A)$ .
- **uplo** – [in] [hipblasFillMode\_t] Specifies whether the matrix  $A$  is an upper or lower triangular matrix as follows: HIPBLAS\_FILL\_MODE\_UPPER:  $A$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER:  $A$  is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t] Specifies the form of  $\text{op}(A)$  to be used in the matrix multiplication as follows: HIPBLAS\_OP\_N:  $\text{op}(A) = A$ . HIPBLAS\_OP\_T:  $\text{op}(A) = A^T$ . HIPBLAS\_OP\_C:  $\text{op}(A) = A^H$ .
- **diag** – [in] [hipblasDiagType\_t] Specifies whether or not  $A$  is unit triangular as follows: HIPBLAS\_DIAG\_UNIT:  $A$  is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT:  $A$  is not assumed to be unit triangular.
- **m** – [in] [int]  $m$  specifies the number of rows of  $B$ .  $m \geq 0$ .
- **n** – [in] [int]  $n$  specifies the number of columns of  $B$ .  $n \geq 0$ .
- **alpha** – [in]  $\alpha$  specifies the scalar  $\alpha$ . When  $\alpha$  is zero then  $A$  is not referenced and  $B$  need not be set before entry.
- **AP** – [in] Device pointer to matrix  $A$  on the GPU.  $A$  has dimension  $(lda, k)$ , where  $k$  is  $m$  when  $\text{side} == \text{HIPBLAS\_SIDE\_LEFT}$  and is  $n$  when  $\text{side} == \text{HIPBLAS\_SIDE\_RIGHT}$ .
- **lda** – [in] [int]  $lda$  specifies the first dimension of  $A$ . if  $\text{side} == \text{HIPBLAS\_SIDE\_LEFT}$ ,  $lda \geq \max(1, m)$ , if  $\text{side} == \text{HIPBLAS\_SIDE\_RIGHT}$ ,  $lda \geq \max(1, n)$ .
- **BP** – [inout] Device pointer to the first matrix  $B_0$  on the GPU. On entry, the leading  $m$  by  $n$  part of the array  $B$  must contain the matrix  $B$ , and on exit is overwritten by the transformed matrix.
- **ldb** – [in] [int]  $ldb$  specifies the first dimension of  $B$ .  $ldb \geq \max(1, m)$ .

*hipblasStatus\_t* **hipblasStrmmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const float \*alpha, const float \*const AP[], int lda, float \*const BP[], int ldb, int batchCount)

*hipblasStatus\_t* **hipblasDtrmmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const double \*alpha, const double \*const AP[], int lda, double \*const BP[], int ldb, int batchCount)

*hipblasStatus\_t* **hipblasCtrmmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*const AP[], int lda, *hipblasComplex* \*const BP[], int ldb, int batchCount)

*hipblasStatus\_t* **hipblasZtrmmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*const AP[], int lda, *hipblasDoubleComplex* \*const BP[], int ldb, int batchCount)

BLAS Level 3 API.

trmmBatched performs one of the batched matrix-matrix operations

$B_i := \alpha * \text{op}(A_i) * B_i$ , or  $B_i := \alpha * B_i * \text{op}(A_i)$  for  $i = 0, 1, \dots, \text{batchCount} - 1$

where  $\alpha$  is a scalar,  $B_i$  is an  $m$  by  $n$  matrix,  $A_i$  is a unit, or non-unit, upper or lower triangular matrix and  $op(A_i)$  is one of

$$op(A_i) = A_i \quad \text{or} \quad op(A_i) = A_i^T \quad \text{or} \quad op(A_i) = A_i^H.$$

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

When `uplo == HIPBLAS_FILL_MODE_UPPER` the leading  $k$  by  $k$  upper triangular part of the array  $A$  must contain the upper triangular matrix and the strictly lower triangular part of  $A$  is not referenced.

When `uplo == HIPBLAS_FILL_MODE_LOWER` the leading  $k$  by  $k$  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 == HIPBLAS_DIAG_UNIT` the diagonal elements of  $A_i$  are not referenced either, but are assumed to be unity.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] Specifies whether  $op(A_i)$  multiplies  $B_i$  from the left or right as follows: `HIPBLAS_SIDE_LEFT`:  $B_i := \alpha * op(A_i) * B_i$ . `HIPBLAS_SIDE_RIGHT`:  $B_i := \alpha * B_i * op(A_i)$ .
- **uplo** – [in] [hipblasFillMode\_t] Specifies whether the matrix  $A$  is an upper or lower triangular matrix as follows: `HIPBLAS_FILL_MODE_UPPER`:  $A$  is an upper triangular matrix. `HIPBLAS_FILL_MODE_LOWER`:  $A$  is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t] Specifies the form of  $op(A_i)$  to be used in the matrix multiplication as follows: `HIPBLAS_OP_N`:  $op(A_i) = A_i$ . `HIPBLAS_OP_T`:  $op(A_i) = A_i^T$ . `HIPBLAS_OP_C`:  $op(A_i) = A_i^H$ .
- **diag** – [in] [hipblasDiagType\_t] Specifies whether or not  $A_i$  is unit triangular as follows: `HIPBLAS_DIAG_UNIT`:  $A_i$  is assumed to be unit triangular. `HIPBLAS_DIAG_NON_UNIT`:  $A_i$  is not assumed to be unit triangular.
- **m** – [in] [int]  $m$  specifies the number of rows of  $B_i$ .  $m \geq 0$ .
- **n** – [in] [int]  $n$  specifies the number of columns of  $B_i$ .  $n \geq 0$ .
- **alpha** – [in]  $\alpha$  specifies the scalar  $\alpha$ . When  $\alpha$  is zero then  $A_i$  is not referenced and  $B_i$  need not be set before entry.
- **AP** – [in] Device array of device pointers storing each matrix  $A_i$  on the GPU. Each  $A_i$  is of dimension  $(lda, k)$ , where  $k$  is  $m$  when `side == HIPBLAS_SIDE_LEFT` and is  $n$  when `side == HIPBLAS_SIDE_RIGHT`.
- **lda** – [in] [int]  $lda$  specifies the first dimension of  $A$ . if `side == HIPBLAS_SIDE_LEFT`,  $lda \geq \max(1, m)$ , if `side == HIPBLAS_SIDE_RIGHT`,  $lda \geq \max(1, n)$ .
- **BP** – [inout] device array of device pointers storing each matrix  $B_i$  on the GPU. On entry, the leading  $m$  by  $n$  part of the array  $B_i$  must contain the matrix  $B_i$ , and on exit is overwritten by the transformed matrix.
- **ldb** – [in] [int]  $ldb$  specifies the first dimension of  $B_i$ .  $ldb \geq \max(1, m)$ .
- **batchCount** – [in] [int] number of instances  $i$  in the batch.

*hipblasStatus\_t* **hipblasStrmmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const float \*alpha, const float \*AP, int lda, *hipblasStride* strideA, float \*BP, int ldb, *hipblasStride* strideB, int batchCount)

*hipblasStatus\_t* **hipblasDtrmmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const double \*alpha, const double \*AP, int lda, *hipblasStride* strideA, double \*BP, int ldb, *hipblasStride* strideB, int batchCount)

*hipblasStatus\_t* **hipblasCtrmmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasComplex* \*alpha, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasComplex* \*BP, int ldb, *hipblasStride* strideB, int batchCount)

*hipblasStatus\_t* **hipblasZtrmmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasDoubleComplex* \*alpha, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasDoubleComplex* \*BP, int ldb, *hipblasStride* strideB, int batchCount)

BLAS Level 3 API.

trmmStridedBatched performs one of the strided\_batched matrix-matrix operations

$B_i := \alpha * \text{op}(A_i) * B_i$ , or  $B_i := \alpha * B_i * \text{op}(A_i)$  for  $i = 0, 1, \dots, \text{batchCount} - 1$

where alpha is a scalar,  $B_i$  is an m by n matrix,  $A_i$  is a unit, or non-unit, upper or lower triangular matrix and  $\text{op}(A_i)$  is one of

$\text{op}(A_i) = A_i$ <b>or</b> $\text{op}(A_i) = A_i^T$ <b>or</b> $\text{op}(A_i) = A_i^H$ .
--

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

When uplo == HIPBLAS\_FILL\_MODE\_UPPER the leading k by 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.

When uplo == HIPBLAS\_FILL\_MODE\_LOWER the leading k by 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 == HIPBLAS\_DIAG\_UNIT the diagonal elements of  $A_i$  are not referenced either, but are assumed to be unity.

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.

- **side** – [in] [hipblasSideMode\_t] Specifies whether  $\text{op}(A_i)$  multiplies  $B_i$  from the left or right as follows: HIPBLAS\_SIDE\_LEFT:  $B_i := \alpha * \text{op}(A_i) * B_i$ . HIPBLAS\_SIDE\_RIGHT:  $B_i := \alpha * B_i * \text{op}(A_i)$ .
- **uplo** – [in] [hipblasFillMode\_t] Specifies whether the matrix  $A$  is an upper or lower triangular matrix as follows: HIPBLAS\_FILL\_MODE\_UPPER:  $A$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER:  $A$  is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t] Specifies the form of  $\text{op}(A_i)$  to be used in the matrix multiplication as follows: HIPBLAS\_OP\_N:  $\text{op}(A_i) = A_i$ . HIPBLAS\_OP\_T:  $\text{op}(A_i) = A_i^T$ . HIPBLAS\_OP\_C:  $\text{op}(A_i) = A_i^H$ .
- **diag** – [in] [hipblasDiagType\_t] Specifies whether or not  $A_i$  is unit triangular as follows: HIPBLAS\_DIAG\_UNIT:  $A_i$  is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT:  $A_i$  is not assumed to be unit triangular.
- **m** – [in] [int]  $m$  specifies the number of rows of  $B_i$ .  $m \geq 0$ .
- **n** – [in] [int]  $n$  specifies the number of columns of  $B_i$ .  $n \geq 0$ .
- **alpha** – [in]  $\alpha$  specifies the scalar  $\alpha$ . When  $\alpha$  is zero then  $A_i$  is not referenced and  $B_i$  need not be set before entry.
- **AP** – [in] Device pointer to the first matrix  $A_0$  on the GPU. Each  $A_i$  is of dimension  $(lda, k)$ , where  $k$  is  $m$  when  $\text{side} == \text{HIPBLAS\_SIDE\_LEFT}$  and is  $n$  when  $\text{side} == \text{HIPBLAS\_SIDE\_RIGHT}$ .
- **lda** – [in] [int]  $lda$  specifies the first dimension of  $A$ . if  $\text{side} == \text{HIPBLAS\_SIDE\_LEFT}$ ,  $lda \geq \max(1, m)$ , if  $\text{side} == \text{HIPBLAS\_SIDE\_RIGHT}$ ,  $lda \geq \max(1, n)$ .
- **strideA** – [in] [hipblasStride] stride from the start of one matrix ( $A_i$ ) and the next one ( $A_{i+1}$ )
- **BP** – [inout] Device pointer to the first matrix  $B_0$  on the GPU. On entry, the leading  $m$  by  $n$  part of the array  $B_i$  must contain the matrix  $B_i$ , and on exit is overwritten by the transformed matrix.
- **ldb** – [in] [int]  $ldb$  specifies the first dimension of  $B_i$ .  $ldb \geq \max(1, m)$ .
- **strideB** – [in] [hipblasStride] stride from the start of one matrix ( $B_i$ ) and the next one ( $B_{i+1}$ )
- **batchCount** – [in] [int] number of instances  $i$  in the batch.

### 1.8.3.12 hipblasXtrsm + Batched, StridedBatched

*hipblasStatus\_t* **hipblasStrsm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const float \*alpha, float \*AP, int lda, float \*BP, int ldb)

*hipblasStatus\_t* **hipblasDtrsm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const double \*alpha, double \*AP, int lda, double \*BP, int ldb)

*hipblasStatus\_t* **hipblasCtrsm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasComplex* \*alpha, *hipblasComplex* \*AP, int lda, *hipblasComplex* \*BP, int ldb)

*hipblasStatus\_t* **hipblasZtrsm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasDoubleComplex* \*alpha, *hipblasDoubleComplex* \*AP, int lda, *hipblasDoubleComplex* \*BP, int ldb)

BLAS Level 3 API.

trsm solves

$$\text{op}(A) * X = \alpha * B \quad \text{or} \quad X * \text{op}(A) = \alpha * B,$$

where alpha is a scalar, X and B are m by n matrices, A is triangular matrix and op(A) is one of

$$\text{op}(A) = A \quad \text{or} \quad \text{op}(A) = A^T \quad \text{or} \quad \text{op}(A) = A^H.$$

The matrix X is overwritten on B.

Note about memory allocation: When trsm is launched with a k evenly divisible by the internal block size of 128, and is no larger than 10 of these blocks, the API takes advantage of utilizing pre-allocated memory found in the handle to increase overall performance. This memory can be managed by using the environment variable WORKBUF\_TRSM\_B\_CHNK. When this variable is not set the device memory used for temporary storage will default to 1 MB and may result in chunking, which in turn may reduce performance. Under these circumstances it is recommended that WORKBUF\_TRSM\_B\_CHNK be set to the desired chunk of right hand sides to be used at a time.

(where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT)

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] HIPBLAS\_SIDE\_LEFT:  $\text{op}(A) * X = \alpha * B$ . HIPBLAS\_SIDE\_RIGHT:  $X * \text{op}(A) = \alpha * B$ .
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: A is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_N:  $\text{op}(A) = A$ . HIPBLAS\_OP\_T:  $\text{op}(A) = A^T$ . HIPBLAS\_OP\_C:  $\text{op}(A) = A^H$ .
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: A is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: A is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of B.  $m \geq 0$ .
- **n** – [in] [int] n specifies the number of columns of B.  $n \geq 0$ .
- **alpha** – [in] device pointer or host pointer specifying the scalar alpha. When alpha is &zero then A is not referenced and B need not be set before entry.
- **AP** – [in] device pointer storing matrix A. of dimension ( lda, k ), where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if side = HIPBLAS\_SIDE\_LEFT,  $lda \geq \max(1, m)$ , if side = HIPBLAS\_SIDE\_RIGHT,  $lda \geq \max(1, n)$ .
- **BP** – [inout] device pointer storing matrix B.

- **ldb** – [in] [int] ldb specifies the first dimension of B.  $ldb \geq \max(1, m)$ .

*hipblasStatus\_t* **hipblasStrsmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const float \*alpha, float \*const AP[], int lda, float \*BP[], int ldb, int batchCount)

*hipblasStatus\_t* **hipblasDtrsmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const double \*alpha, double \*const AP[], int lda, double \*BP[], int ldb, int batchCount)

*hipblasStatus\_t* **hipblasCtrsmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasComplex* \*alpha, *hipblasComplex* \*const AP[], int lda, *hipblasComplex* \*BP[], int ldb, int batchCount)

*hipblasStatus\_t* **hipblasZtrsmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasDoubleComplex* \*alpha, *hipblasDoubleComplex* \*const AP[], int lda, *hipblasDoubleComplex* \*BP[], int ldb, int batchCount)

BLAS Level 3 API.

trsmBatched performs the following batched operation:

$$\text{op}(A_i) * X_i = \alpha * B_i \text{ or } X_i * \text{op}(A_i) = \alpha * B_i, \text{ for } i = 1, \dots, \text{batchCount}.$$

where alpha is a scalar, X and B are batched m by n matrices, A is triangular batched matrix and op(A) is one of

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

Each matrix  $X_i$  is overwritten on  $B_i$  for  $i = 1, \dots, \text{batchCount}$ .

Note about memory allocation: When trsm is launched with a k evenly divisible by the internal block size of 128, and is no larger than 10 of these blocks, the API takes advantage of utilizing pre-allocated memory found in the handle to increase overall performance. This memory can be managed by using the environment variable WORKBUF\_TRSM\_B\_CHUNK. When this variable is not set the device memory used for temporary storage will default to 1 MB and may result in chunking, which in turn may reduce performance. Under these circumstances it is recommended that WORKBUF\_TRSM\_B\_CHUNK be set to the desired chunk of right hand sides to be used at a time. (where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT)

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] HIPBLAS\_SIDE\_LEFT:  $\text{op}(A) * X = \alpha * B$ . HIPBLAS\_SIDE\_RIGHT:  $X * \text{op}(A) = \alpha * B$ .
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: each  $A_i$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: each  $A_i$  is a lower triangular matrix.



- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_N:  $\text{op}(A) = A$ . HIPBLAS\_OP\_T:  $\text{op}(A) = A^T$ . HIPBLAS\_OP\_C:  $\text{op}(A) = A^H$ .
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: each  $A_i$  is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: each  $A_i$  is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of each  $B_i$ .  $m \geq 0$ .
- **n** – [in] [int] n specifies the number of columns of each  $B_i$ .  $n \geq 0$ .
- **alpha** – [in] device pointer or host pointer specifying the scalar alpha. When alpha is &zero then A is not referenced and B need not be set before entry.
- **AP** – [in] device array of device pointers storing each matrix  $A_i$  on the GPU. Matrices are of dimension ( lda, k ), where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of each  $A_i$ . if side = HIPBLAS\_SIDE\_LEFT,  $\text{lda} \geq \max(1, m)$ , if side = HIPBLAS\_SIDE\_RIGHT,  $\text{lda} \geq \max(1, n)$ .
- **BP** – [inout] device array of device pointers storing each matrix  $B_i$  on the GPU.
- **ldb** – [in] [int] ldb specifies the first dimension of each  $B_i$ .  $\text{ldb} \geq \max(1, m)$ .
- **batchCount** – [in] [int] number of trsm operations in the batch.

*hipblasStatus\_t* **hipblasStrsmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const float \*alpha, float \*AP, int lda, *hipblasStride* strideA, float \*BP, int ldb, *hipblasStride* strideB, int batchCount)

*hipblasStatus\_t* **hipblasDtrsmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const double \*alpha, double \*AP, int lda, *hipblasStride* strideA, double \*BP, int ldb, *hipblasStride* strideB, int batchCount)

*hipblasStatus\_t* **hipblasCtrsmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasComplex* \*alpha, *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasComplex* \*BP, int ldb, *hipblasStride* strideB, int batchCount)

*hipblasStatus\_t* **hipblasZtrsmStridedBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const *hipblasDoubleComplex* \*alpha, *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasDoubleComplex* \*BP, int ldb, *hipblasStride* strideB, int batchCount)

BLAS Level 3 API.

trsmStridedBatched performs the following strided batched operation:



$$\text{op}(A_i) * X_i = \alpha * B_i \text{ or } X_i * \text{op}(A_i) = \alpha * B_i, \text{ for } i = 1, \dots, \text{batchCount}.$$

where  $\alpha$  is a scalar,  $X$  and  $B$  are strided batched  $m$  by  $n$  matrices,  $A$  is triangular strided batched matrix and  $\text{op}(A)$  is one of

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

Each matrix  $X_i$  is overwritten on  $B_i$  for  $i = 1, \dots, \text{batchCount}$ .

Note about memory allocation: When `trsm` is launched with a  $k$  evenly divisible by the internal block size of 128, and is no larger than 10 of these blocks, the API takes advantage of utilizing pre-allocated memory found in the handle to increase overall performance. This memory can be managed by using the environment variable `WORKBUF_TRSM_B_CHNK`. When this variable is not set the device memory used for temporary storage will default to 1 MB and may result in chunking, which in turn may reduce performance. Under these circumstances it is recommended that `WORKBUF_TRSM_B_CHNK` be set to the desired chunk of right hand sides to be used at a time. (where  $k$  is  $m$  when `HIPBLAS_SIDE_LEFT` and is  $n$  when `HIPBLAS_SIDE_RIGHT`)

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] `HIPBLAS_SIDE_LEFT`:  $\text{op}(A) * X = \alpha * B$ . `HIPBLAS_SIDE_RIGHT`:  $X * \text{op}(A) = \alpha * B$ .
- **uplo** – [in] [hipblasFillMode\_t] `HIPBLAS_FILL_MODE_UPPER`: each  $A_i$  is an upper triangular matrix. `HIPBLAS_FILL_MODE_LOWER`: each  $A_i$  is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t] `HIPBLAS_OP_N`:  $\text{op}(A) = A$ . `HIPBLAS_OP_T`:  $\text{op}(A) = A^T$ . `HIPBLAS_OP_C`:  $\text{op}(A) = A^H$ .
- **diag** – [in] [hipblasDiagType\_t] `HIPBLAS_DIAG_UNIT`: each  $A_i$  is assumed to be unit triangular. `HIPBLAS_DIAG_NON_UNIT`: each  $A_i$  is not assumed to be unit triangular.
- **m** – [in] [int]  $m$  specifies the number of rows of each  $B_i$ .  $m \geq 0$ .
- **n** – [in] [int]  $n$  specifies the number of columns of each  $B_i$ .  $n \geq 0$ .
- **alpha** – [in] device pointer or host pointer specifying the scalar  $\alpha$ . When  $\alpha$  is `&zero` then  $A$  is not referenced and  $B$  need not be set before entry.
- **AP** – [in] device pointer pointing to the first matrix  $A_1$ . of dimension  $(lda, k)$ , where  $k$  is  $m$  when `HIPBLAS_SIDE_LEFT` and is  $n$  when `HIPBLAS_SIDE_RIGHT` only the upper/lower triangular part is accessed.
- **lda** – [in] [int]  $lda$  specifies the first dimension of each  $A_i$ . if `side = HIPBLAS_SIDE_LEFT`,  $lda \geq \max(1, m)$ , if `side = HIPBLAS_SIDE_RIGHT`,  $lda \geq \max(1, n)$ .
- **strideA** – [in] [hipblasStride] stride from the start of one  $A_i$  matrix to the next  $A_{(i+1)}$ .
- **BP** – [inout] device pointer pointing to the first matrix  $B_1$ .
- **ldb** – [in] [int]  $ldb$  specifies the first dimension of each  $B_i$ .  $ldb \geq \max(1, m)$ .
- **strideB** – [in] [hipblasStride] stride from the start of one  $B_i$  matrix to the next  $B_{(i+1)}$ .
- **batchCount** – [in] [int] number of `trsm` operations in the batch.

### 1.8.3.13 hipblasXtrtri + Batched, StridedBatched

*hipblasStatus\_t* **hipblasStrtri**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const float \*AP, int lda, float \*invA, int ldinvA)

*hipblasStatus\_t* **hipblasDtrtri**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const double \*AP, int lda, double \*invA, int ldinvA)

*hipblasStatus\_t* **hipblasCtrtri**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const *hipblasComplex* \*AP, int lda, *hipblasComplex* \*invA, int ldinvA)

*hipblasStatus\_t* **hipblasZtrtri**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const *hipblasDoubleComplex* \*AP, int lda, *hipblasDoubleComplex* \*invA, int ldinvA)

BLAS Level 3 API.

trtri compute the inverse of a matrix A, namely, invA

**and** write the result into invA;

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’ if HIPBLAS\_FILL\_MODE\_UPPER, the lower part of A is not referenced if HIPBLAS\_FILL\_MODE\_LOWER, the upper part of A is not referenced
- **diag** – [in] [*hipblasDiagType\_t*] = ‘HIPBLAS\_DIAG\_NON\_UNIT’, A is non-unit triangular; = ‘HIPBLAS\_DIAG\_UNIT’, A is unit triangular;
- **n** – [in] [int] size of matrix A and invA
- **AP** – [in] device pointer storing matrix A.
- **lda** – [in] [int] specifies the leading dimension of A.
- **invA** – [out] device pointer storing matrix invA.
- **ldinvA** – [in] [int] specifies the leading dimension of invA.

*hipblasStatus\_t* **hipblasStrtriBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const float \*const AP[], int lda, float \*invA[], int ldinvA, int batchSize)

*hipblasStatus\_t* **hipblasDtrtriBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const double \*const AP[], int lda, double \*invA[], int ldinvA, int batchSize)

*hipblasStatus\_t* **hipblasCttrtriBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const *hipblasComplex* \*const AP[], int lda, *hipblasComplex* \*invA[], int ldinvA, int batchCount)

*hipblasStatus\_t* **hipblasZttrtriBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const *hipblasDoubleComplex* \*const AP[], int lda, *hipblasDoubleComplex* \*invA[], int ldinvA, int batchCount)

BLAS Level 3 API.

trtriBatched compute the inverse of  $A_i$  and write into  $invA_i$  where  $A_i$  and  $invA_i$  are the  $i$ -th matrices in the batch, for  $i = 1, \dots, \text{batchCount}$ .

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **uplo** – [in] [*hipblasFillMode\_t*] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’
- **diag** – [in] [*hipblasDiagType\_t*] = ‘HIPBLAS\_DIAG\_NON\_UNIT’, A is non-unit triangular; = ‘HIPBLAS\_DIAG\_UNIT’, A is unit triangular;
- **n** – [in] [int]
- **AP** – [in] device array of device pointers storing each matrix  $A_i$ .
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .
- **invA** – [out] device array of device pointers storing the inverse of each matrix  $A_i$ . Partial inplace operation is supported, see below. If UPLO = ‘U’, the leading N-by-N upper triangular part of the invA will store the inverse of the upper triangular matrix, and the strictly lower triangular part of invA is cleared. If UPLO = ‘L’, the leading N-by-N lower triangular part of the invA will store the inverse of the lower triangular matrix, and the strictly upper triangular part of invA is cleared.
- **ldinvA** – [in] [int] specifies the leading dimension of each  $invA_i$ .
- **batchCount** – [in] [int] numbers of matrices in the batch

*hipblasStatus\_t* **hipblasStrtriStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const float \*AP, int lda, *hipblasStride* strideA, float \*invA, int ldinvA, *hipblasStride* stride\_invA, int batchCount)

*hipblasStatus\_t* **hipblasDtrtriStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const double \*AP, int lda, *hipblasStride* strideA, double \*invA, int ldinvA, *hipblasStride* stride\_invA, int batchCount)

*hipblasStatus\_t* **hipblasCtrtriStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const *hipblasComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasComplex* \*invA, int ldinvA, *hipblasStride* stride\_invA, int batchCount)

*hipblasStatus\_t* **hipblasZtrtriStridedBatched**(*hipblasHandle\_t* handle, *hipblasFillMode\_t* uplo, *hipblasDiagType\_t* diag, int n, const *hipblasDoubleComplex* \*AP, int lda, *hipblasStride* strideA, *hipblasDoubleComplex* \*invA, int ldinvA, *hipblasStride* stride\_invA, int batchCount)

BLAS Level 3 API.

trtriStridedBatched compute the inverse of A<sub>i</sub> and write into invA<sub>i</sub> where A<sub>i</sub> and invA<sub>i</sub> are the i-th matrices in the batch, for i = 1, ..., batchCount

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **uplo** – [in] [hipblasFillMode\_t] specifies whether the upper ‘HIPBLAS\_FILL\_MODE\_UPPER’ or lower ‘HIPBLAS\_FILL\_MODE\_LOWER’
- **diag** – [in] [hipblasDiagType\_t] = ‘HIPBLAS\_DIAG\_NON\_UNIT’, A is non-unit triangular; = ‘HIPBLAS\_DIAG\_UNIT’, A is unit triangular;
- **n** – [in] [int]
- **AP** – [in] device pointer pointing to address of first matrix A<sub>1</sub>.
- **lda** – [in] [int] specifies the leading dimension of each A.
- **strideA** – [in] [hipblasStride] “batch stride a”: stride from the start of one A<sub>i</sub> matrix to the next A<sub>(i + 1)</sub>.
- **invA** – [out] device pointer storing the inverses of each matrix A<sub>i</sub>. Partial inplace operation is supported, see below. If UPLO = ‘U’, the leading N-by-N upper triangular part of the invA will store the inverse of the upper triangular matrix, and the strictly lower triangular part of invA is cleared. If UPLO = ‘L’, the leading N-by-N lower triangular part of the invA will store the inverse of the lower triangular matrix, and the strictly upper triangular part of invA is cleared.
- **ldinvA** – [in] [int] specifies the leading dimension of each invA<sub>i</sub>.
- **stride\_invA** – [in] [hipblasStride] “batch stride invA”: stride from the start of one invA<sub>i</sub> matrix to the next invA<sub>(i + 1)</sub>.
- **batchCount** – [in] [int] numbers of matrices in the batch

### 1.8.3.14 hipblasXdgmm + Batched, StridedBatched

*hipblasStatus\_t* **hipblasSdgmm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, int m, int n, const float \*AP, int lda, const float \*x, int incx, float \*CP, int ldc)

*hipblasStatus\_t* **hipblasDdgmm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, int m, int n, const double \*AP, int lda, const double \*x, int incx, double \*CP, int ldc)

*hipblasStatus\_t* **hipblasCdgm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, int m, int n, const *hipblasComplex* \*AP, int lda, const *hipblasComplex* \*x, int incx, *hipblasComplex* \*CP, int ldc)

*hipblasStatus\_t* **hipblasZdgm**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, int m, int n, const *hipblasDoubleComplex* \*AP, int lda, const *hipblasDoubleComplex* \*x, int incx, *hipblasDoubleComplex* \*CP, int ldc)

BLAS Level 3 API.

dgmm performs one of the matrix-matrix operations

```
C = A * diag(x) if side == HIPBLAS_SIDE_RIGHT
C = diag(x) * A if side == HIPBLAS_SIDE_LEFT
```

where C and A are m by n dimensional matrices. diag( x ) is a diagonal matrix and x is vector of dimension n if side == HIPBLAS\_SIDE\_RIGHT and dimension m if side == HIPBLAS\_SIDE\_LEFT.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **side** – [in] [*hipblasSideMode\_t*] specifies the side of diag(x)
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **AP** – [in] device pointer storing matrix A.
- **lda** – [in] [int] specifies the leading dimension of A.
- **x** – [in] device pointer storing vector x.
- **incx** – [in] [int] specifies the increment between values of x
- **CP** – [inout] device pointer storing matrix C.
- **ldc** – [in] [int] specifies the leading dimension of C.

*hipblasStatus\_t* **hipblasSdgmmBatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, int m, int n, const float \*const AP[], int lda, const float \*const x[], int incx, float \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasDdgmmbatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, int m, int n, const double \*const AP[], int lda, const double \*const x[], int incx, double \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasCdgmmbatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, int m, int n, const *hipblasComplex* \*const AP[], int lda, const *hipblasComplex* \*const x[], int incx, *hipblasComplex* \*const CP[], int ldc, int batchCount)

*hipblasStatus\_t* **hipblasZdgmmbatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, int m, int n, const *hipblasDoubleComplex* \*const AP[], int lda, const *hipblasDoubleComplex* \*const x[], int incx, *hipblasDoubleComplex* \*const CP[], int ldc, int batchCount)

BLAS Level 3 API.

dgmmBatched performs one of the batched matrix-matrix operations

```
C_i = A_i * diag(x_i) for i = 0, 1, ... batchCount-1 if side == HIPBLAS_SIDE_RIGHT
C_i = diag(x_i) * A_i for i = 0, 1, ... batchCount-1 if side == HIPBLAS_SIDE_LEFT
```

where C\_i and A\_i are m by n dimensional matrices. diag(x\_i) is a diagonal matrix and x\_i is vector of dimension n if side == HIPBLAS\_SIDE\_RIGHT and dimension m if side == HIPBLAS\_SIDE\_LEFT.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **side** – [in] [*hipblasSideMode\_t*] specifies the side of diag(x)
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **AP** – [in] device array of device pointers storing each matrix A\_i on the GPU. Each A\_i is of dimension ( lda, n )
- **lda** – [in] [int] specifies the leading dimension of A\_i.
- **x** – [in] device array of device pointers storing each vector x\_i on the GPU. Each x\_i is of dimension n if side == HIPBLAS\_SIDE\_RIGHT and dimension m if side == HIPBLAS\_SIDE\_LEFT
- **incx** – [in] [int] specifies the increment between values of x\_i
- **CP** – [inout] device array of device pointers storing each matrix C\_i on the GPU. Each C\_i is of dimension ( ldc, n ).
- **ldc** – [in] [int] specifies the leading dimension of C\_i.
- **batchCount** – [in] [int] number of instances in the batch.

*hipblasStatus\_t* **hipblasSdgmmbatched**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, int m, int n, const float \*AP, int lda, *hipblasStride* strideA, const float \*x, int incx, *hipblasStride* stridex, float \*CP, int ldc, *hipblasStride* strideC, int batchCount)

```
hipblasStatus_t hipblasDdgmmStridedBatched(hipblasHandle_t handle, hipblasSideMode_t side, int m, int n,
                                             const double *AP, int lda, hipblasStride strideA, const double *x,
                                             int incx, hipblasStride stridex, double *CP, int ldc, hipblasStride
                                             strideC, int batchCount)
```

```
hipblasStatus_t hipblasCdgmStridedBatched(hipblasHandle_t handle, hipblasSideMode_t side, int m, int n,
                                             const hipblasComplex *AP, int lda, hipblasStride strideA, const
                                             hipblasComplex *x, int incx, hipblasStride stridex,
                                             hipblasComplex *CP, int ldc, hipblasStride strideC, int
                                             batchCount)
```

```
hipblasStatus_t hipblasZdgmmStridedBatched(hipblasHandle_t handle, hipblasSideMode_t side, int m, int n,
                                             const hipblasDoubleComplex *AP, int lda, hipblasStride strideA,
                                             const hipblasDoubleComplex *x, int incx, hipblasStride stridex,
                                             hipblasDoubleComplex *CP, int ldc, hipblasStride strideC, int
                                             batchCount)
```

BLAS Level 3 API.

dgmmStridedBatched performs one of the batched matrix-matrix operations

```
C_i = A_i * diag(x_i)  if side == HIPBLAS_SIDE_RIGHT  for i = 0, 1, ...,
↪batchCount-1
C_i = diag(x_i) * A_i  if side == HIPBLAS_SIDE_LEFT   for i = 0, 1, ...,
↪batchCount-1
```

where C<sub>i</sub> and A<sub>i</sub> are m by n dimensional matrices. diag(x<sub>i</sub>) is a diagonal matrix and x<sub>i</sub> is vector of dimension n if side == HIPBLAS\_SIDE\_RIGHT and dimension m if side == HIPBLAS\_SIDE\_LEFT.

- Supported precisions in rocBLAS : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] specifies the side of diag(x)
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **AP** – [in] device pointer to the first matrix A<sub>0</sub> on the GPU. Each A<sub>i</sub> is of dimension ( lda, n )
- **lda** – [in] [int] specifies the leading dimension of A.
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A<sub>i</sub>) and the next one (A<sub>i+1</sub>)
- **x** – [in] pointer to the first vector x<sub>0</sub> on the GPU. Each x<sub>i</sub> is of dimension n if side == HIPBLAS\_SIDE\_RIGHT and dimension m if side == HIPBLAS\_SIDE\_LEFT
- **incx** – [in] [int] specifies the increment between values of x
- **stridex** – [in] [hipblasStride] stride from the start of one vector(x<sub>i</sub>) and the next one (x<sub>i+1</sub>)

- **CP** – [inout] device pointer to the first matrix C\_0 on the GPU. Each C\_i is of dimension ( ldc, n ).
- **ldc** – [in] [int] specifies the leading dimension of C.
- **strideC** – [in] [hipblasStride] stride from the start of one matrix (C\_i) and the next one (C\_{i+1}).
- **batchCount** – [in] [int] number of instances i in the batch.

## 1.8.4 SOLVER API

### List of SOLVER APIs

- *hipblasXgetrf + Batched, stridedBatched*
- *hipblasXgetrs + Batched, stridedBatched*
- *hipblasXgetri + Batched, stridedBatched*
- *hipblasXgeqrf + Batched, stridedBatched*

#### 1.8.4.1 hipblasXgetrf + Batched, stridedBatched

*hipblasStatus\_t* **hipblasSgetrf**(*hipblasHandle\_t* handle, const int n, float \*A, const int lda, int \*ipiv, int \*info)

*hipblasStatus\_t* **hipblasDgetrf**(*hipblasHandle\_t* handle, const int n, double \*A, const int lda, int \*ipiv, int \*info)

*hipblasStatus\_t* **hipblasCgetrf**(*hipblasHandle\_t* handle, const int n, *hipblasComplex* \*A, const int lda, int \*ipiv, int \*info)

*hipblasStatus\_t* **hipblasZgetrf**(*hipblasHandle\_t* handle, const int n, *hipblasDoubleComplex* \*A, const int lda, int \*ipiv, int \*info)

SOLVER API.

getrf computes the LU factorization of a general n-by-n matrix A using partial pivoting with row interchanges. The LU factorization can be done without pivoting if ipiv is passed as a nullptr.

In the case that ipiv is not null, the factorization has the form:

$$A = PLU$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements, and U is upper triangular.

In the case that ipiv is null, the factorization is done without pivoting:

$$A = LU$$



- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

### Parameters

- **handle** – [in] hipblasHandle\_t.
- **n** – [in] int.  $n \geq 0$ .  
The number of columns and rows of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $\text{lda} \times n$ .  
On entry, the n-by-n matrix A to be factored. On exit, the factors L and U from the factorization. The unit diagonal elements of L are not stored.
- **lda** – [in] int.  $\text{lda} \geq n$ .  
Specifies the leading dimension of A.
- **ipiv** – [out] pointer to int. Array on the GPU of dimension n.  
The vector of pivot indices. Elements of ipiv are 1-based indices. For  $1 \leq i \leq n$ , the row i of the matrix was interchanged with row ipiv[i]. Matrix P of the factorization can be derived from ipiv. The factorization here can be done without pivoting if ipiv is passed in as a nullptr.
- **info** – [out] pointer to a int on the GPU.  
If info = 0, successful exit. If info = j > 0, U is singular. U[j,j] is the first zero pivot.

*hipblasStatus\_t* **hipblasSgetrfBatched**(*hipblasHandle\_t* handle, const int n, float \*const A[], const int lda, int \*ipiv, int \*info, const int batchSize)

*hipblasStatus\_t* **hipblasDgetrfBatched**(*hipblasHandle\_t* handle, const int n, double \*const A[], const int lda, int \*ipiv, int \*info, const int batchSize)

*hipblasStatus\_t* **hipblasCgetrfBatched**(*hipblasHandle\_t* handle, const int n, *hipblasComplex* \*const A[], const int lda, int \*ipiv, int \*info, const int batchSize)

*hipblasStatus\_t* **hipblasZgetrfBatched**(*hipblasHandle\_t* handle, const int n, *hipblasDoubleComplex* \*const A[], const int lda, int \*ipiv, int \*info, const int batchSize)

### SOLVER API.

getrfBatched computes the LU factorization of a batch of general n-by-n matrices using partial pivoting with row interchanges. The LU factorization can be done without pivoting if ipiv is passed as a nullptr.

In the case that ipiv is not null, the factorization of matrix  $A_i$  in the batch has the form:

$$A_i = P_i L_i U_i$$

where  $P_i$  is a permutation matrix,  $L_i$  is lower triangular with unit diagonal elements, and  $U_i$  is upper triangular.

In the case that ipiv is null, the factorization is done without pivoting:

$$A_i = L_i U_i$$

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

### Parameters

- **handle** – [in] hipblasHandle\_t.
- **n** – [in] int.  $n \geq 0$ .  
The number of columns and rows of all matrices  $A_i$  in the batch.
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda \times n$ .  
On entry, the  $n$ -by- $n$  matrices  $A_i$  to be factored. On exit, the factors  $L_i$  and  $U_i$  from the factorizations. The unit diagonal elements of  $L_i$  are not stored.
- **lda** – [in] int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **ipiv** – [out] pointer to int. Array on the GPU.  
Contains the vectors of pivot indices  $ipiv_i$  (corresponding to  $A_i$ ). Dimension of  $ipiv_i$  is  $n$ . Elements of  $ipiv_i$  are 1-based indices. For each instance  $A_i$  in the batch and for  $1 \leq j \leq n$ , the row  $j$  of the matrix  $A_i$  was interchanged with row  $ipiv_i[j]$ . Matrix  $P_i$  of the factorization can be derived from  $ipiv_i$ . The factorization here can be done without pivoting if  $ipiv$  is passed in as a nullptr.
- **info** – [out] pointer to int. Array of batchCount integers on the GPU.  
If  $info[i] = 0$ , successful exit for factorization of  $A_i$ . If  $info[i] = j > 0$ ,  $U_i$  is singular.  $U_i[j,j]$  is the first zero pivot.
- **batchCount** – [in] int.  $batchCount \geq 0$ .  
Number of matrices in the batch.

*hipblasStatus\_t* **hipblasSgetrfStridedBatched**(*hipblasHandle\_t* handle, const int n, float \*A, const int lda, const *hipblasStride* strideA, int \*ipiv, const *hipblasStride* strideP, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasDgetrfStridedBatched**(*hipblasHandle\_t* handle, const int n, double \*A, const int lda, const *hipblasStride* strideA, int \*ipiv, const *hipblasStride* strideP, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasCgetrfStridedBatched**(*hipblasHandle\_t* handle, const int n, *hipblasComplex* \*A, const int lda, const *hipblasStride* strideA, int \*ipiv, const *hipblasStride* strideP, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasZgetrfStridedBatched**(*hipblasHandle\_t* handle, const int n, *hipblasDoubleComplex* \*A, const int lda, const *hipblasStride* strideA, int \*ipiv, const *hipblasStride* strideP, int \*info, const int batchCount)

### SOLVER API.

getrfStridedBatched computes the LU factorization of a batch of general  $n$ -by- $n$  matrices using partial pivoting with row interchanges. The LU factorization can be done without pivoting if  $ipiv$  is passed as a nullptr.

In the case that `ipiv` is not null, the factorization of matrix  $A_i$  in the batch has the form:

$$A_i = P_i L_i U_i$$

where  $P_i$  is a permutation matrix,  $L_i$  is lower triangular with unit diagonal elements, and  $U_i$  is upper triangular.

In the case that `ipiv` is null, the factorization is done without pivoting:

$$A_i = L_i U_i$$

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] `hipblasHandle_t`.
- **n** – [in] `int`.  $n \geq 0$ .  
The number of columns and rows of all matrices  $A_i$  in the batch.
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of `strideA`).  
On entry, the  $n$ -by- $n$  matrices  $A_i$  to be factored. On exit, the factors  $L_i$  and  $U_i$  from the factorization. The unit diagonal elements of  $L_i$  are not stored.
- **lda** – [in] `int`.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **strideA** – [in] `hipblasStride`.  
Stride from the start of one matrix  $A_i$  to the next one  $A_{i+1}$ . There is no restriction for the value of `strideA`. Normal use case is `strideA`  $\geq lda * n$
- **ipiv** – [out] pointer to `int`. Array on the GPU (the size depends on the value of `strideP`).  
Contains the vectors of pivots indices `ipiv_i` (corresponding to  $A_i$ ). Dimension of `ipiv_i` is  $n$ . Elements of `ipiv_i` are 1-based indices. For each instance  $A_i$  in the batch and for  $1 \leq j \leq n$ , the row  $j$  of the matrix  $A_i$  was interchanged with row `ipiv_i[j]`. Matrix  $P_i$  of the factorization can be derived from `ipiv_i`. The factorization here can be done without pivoting if `ipiv` is passed in as a `nullptr`.
- **strideP** – [in] `hipblasStride`.  
Stride from the start of one vector `ipiv_i` to the next one `ipiv_{i+1}`. There is no restriction for the value of `strideP`. Normal use case is `strideP`  $\geq n$ .
- **info** – [out] pointer to `int`. Array of `batchCount` integers on the GPU.  
If `info[i] = 0`, successful exit for factorization of  $A_i$ . If `info[i] = j > 0`,  $U_i$  is singular.  $U_i[j,j]$  is the first zero pivot.
- **batchCount** – [in] `int`.  $batchCount \geq 0$ .  
Number of matrices in the batch.

### 1.8.4.2 hipblasXgetrs + Batched, stridedBatched

*hipblasStatus\_t* **hipblasSgetrs**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, float \*A, const int lda, const int \*ipiv, float \*B, const int ldb, int \*info)

*hipblasStatus\_t* **hipblasDgetrs**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, double \*A, const int lda, const int \*ipiv, double \*B, const int ldb, int \*info)

*hipblasStatus\_t* **hipblasCgetrs**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, *hipblasComplex* \*A, const int lda, const int \*ipiv, *hipblasComplex* \*B, const int ldb, int \*info)

*hipblasStatus\_t* **hipblasZgetrs**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, *hipblasDoubleComplex* \*A, const int lda, const int \*ipiv, *hipblasDoubleComplex* \*B, const int ldb, int \*info)

SOLVER API.

getrs solves a system of n linear equations on n variables in its factorized form.

It solves one of the following systems, depending on the value of trans:

$$\begin{aligned} AX &= B && \text{not transposed,} \\ A^T X &= B && \text{transposed, or} \\ A^H X &= B && \text{conjugate transposed.} \end{aligned}$$

Matrix A is defined by its triangular factors as returned by *getrf*.

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] *hipblasHandle\_t*.
- **trans** – [in] *hipblasOperation\_t*.  
Specifies the form of the system of equations.
- **n** – [in] int.  $n \geq 0$ .  
The order of the system, i.e. the number of columns and rows of A.
- **nrhs** – [in] int.  $\text{nrhs} \geq 0$ .  
The number of right hand sides, i.e., the number of columns of the matrix B.
- **A** – [in] pointer to type. Array on the GPU of dimension  $\text{lda} \times n$ .  
The factors L and U of the factorization  $A = P * L * U$  returned by *getrf*.
- **lda** – [in] int.  $\text{lda} \geq n$ .  
The leading dimension of A.
- **ipiv** – [in] pointer to int. Array on the GPU of dimension n.  
The pivot indices returned by *getrf*.

- **B** – [inout] pointer to type. Array on the GPU of dimension ldb\*nrhs.  
On entry, the right hand side matrix B. On exit, the solution matrix X.
- **ldb** – [in] int. ldb >= n.  
The leading dimension of B.
- **info** – [out] pointer to a int on the host.  
If info = 0, successful exit. If info = j < 0, the j-th argument is invalid.

*hipblasStatus\_t* **hipblasSgetrsBatched**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, float \*const A[], const int lda, const int \*ipiv, float \*const B[], const int ldb, int \*info, const int batchSize)

*hipblasStatus\_t* **hipblasDgetrsBatched**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, double \*const A[], const int lda, const int \*ipiv, double \*const B[], const int ldb, int \*info, const int batchSize)

*hipblasStatus\_t* **hipblasCgetrsBatched**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, *hipblasComplex* \*const A[], const int lda, const int \*ipiv, *hipblasComplex* \*const B[], const int ldb, int \*info, const int batchSize)

*hipblasStatus\_t* **hipblasZgetrsBatched**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, *hipblasDoubleComplex* \*const A[], const int lda, const int \*ipiv, *hipblasDoubleComplex* \*const B[], const int ldb, int \*info, const int batchSize)

#### SOLVER API.

getrsBatched solves a batch of systems of n linear equations on n variables in its factorized forms.

For each instance i in the batch, it solves one of the following systems, depending on the value of trans:

$$\begin{array}{ll} A_i X_i = B_i & \text{not transposed,} \\ A_i^T X_i = B_i & \text{transposed, or} \\ A_i^H X_i = B_i & \text{conjugate transposed.} \end{array}$$

Matrix  $A_i$  is defined by its triangular factors as returned by *getrfBatched*.

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] *hipblasHandle\_t*.
- **trans** – [in] *hipblasOperation\_t*.  
Specifies the form of the system of equations of each instance in the batch.
- **n** – [in] int. n >= 0.  
The order of the system, i.e. the number of columns and rows of all  $A_i$  matrices.

- **nrhs** – [in] int. nrhs  $\geq 0$ .

The number of right hand sides, i.e., the number of columns of all the matrices  $B_i$ .

- **A** – [in] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda \times n$ .

The factors  $L_i$  and  $U_i$  of the factorization  $A_i = P_i * L_i * U_i$  returned by *getrfBatched*.

- **lda** – [in] int.  $lda \geq n$ .

The leading dimension of matrices  $A_i$ .

- **ipiv** – [in] pointer to int. Array on the GPU.

Contains the vectors  $ipiv_i$  of pivot indices returned by *getrfBatched*.

- **B** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldb \times nrhs$ .

On entry, the right hand side matrices  $B_i$ . On exit, the solution matrix  $X_i$  of each system in the batch.

- **ldb** – [in] int.  $ldb \geq n$ .

The leading dimension of matrices  $B_i$ .

- **info** – [out] pointer to a int on the host.

If  $info = 0$ , successful exit. If  $info = j < 0$ , the  $j$ -th argument is invalid.

- **batchCount** – [in] int.  $batchCount \geq 0$ .

Number of instances (systems) in the batch.

*hipblasStatus\_t* **hipblasSgetrsStridedBatched**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, float \*A, const int lda, const *hipblasStride* strideA, const int \*ipiv, const *hipblasStride* strideP, float \*B, const int ldb, const *hipblasStride* strideB, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasDgetrsStridedBatched**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, double \*A, const int lda, const *hipblasStride* strideA, const int \*ipiv, const *hipblasStride* strideP, double \*B, const int ldb, const *hipblasStride* strideB, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasCgetrsStridedBatched**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, *hipblasComplex* \*A, const int lda, const *hipblasStride* strideA, const int \*ipiv, const *hipblasStride* strideP, *hipblasComplex* \*B, const int ldb, const *hipblasStride* strideB, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasZgetrsStridedBatched**(*hipblasHandle\_t* handle, const *hipblasOperation\_t* trans, const int n, const int nrhs, *hipblasDoubleComplex* \*A, const int lda, const *hipblasStride* strideA, const int \*ipiv, const *hipblasStride* strideP, *hipblasDoubleComplex* \*B, const int ldb, const *hipblasStride* strideB, int \*info, const int batchCount)

SOLVER API.

getrsStridedBatched solves a batch of systems of  $n$  linear equations on  $n$  variables in its factorized forms.

For each instance  $i$  in the batch, it solves one of the following systems, depending on the value of `trans`:

$$\begin{aligned} A_i X_i &= B_i && \text{not transposed,} \\ A_i^T X_i &= B_i && \text{transposed, or} \\ A_i^H X_i &= B_i && \text{conjugate transposed.} \end{aligned}$$

Matrix  $A_i$  is defined by its triangular factors as returned by *getrfStridedBatched*.

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] hipblasHandle\_t.
- **trans** – [in] hipblasOperation\_t.  
Specifies the form of the system of equations of each instance in the batch.
- **n** – [in] int.  $n \geq 0$ .  
The order of the system, i.e. the number of columns and rows of all  $A_i$  matrices.
- **nrhs** – [in] int.  $\text{nrhs} \geq 0$ .  
The number of right hand sides, i.e., the number of columns of all the matrices  $B_i$ .
- **A** – [in] pointer to type. Array on the GPU (the size depends on the value of `strideA`).  
The factors  $L_i$  and  $U_i$  of the factorization  $A_i = P_i * L_i * U_i$  returned by *getrfStridedBatched*.
- **lda** – [in] int.  $\text{lda} \geq n$ .  
The leading dimension of matrices  $A_i$ .
- **strideA** – [in] hipblasStride.  
Stride from the start of one matrix  $A_i$  to the next one  $A_{(i+1)}$ . There is no restriction for the value of `strideA`. Normal use case is  $\text{strideA} \geq \text{lda} * n$ .
- **ipiv** – [in] pointer to int. Array on the GPU (the size depends on the value of `strideP`).  
Contains the vectors `ipiv_i` of pivot indices returned by *getrfStridedBatched*.
- **strideP** – [in] hipblasStride.  
Stride from the start of one vector `ipiv_i` to the next one `ipiv_{(i+1)}`. There is no restriction for the value of `strideP`. Normal use case is  $\text{strideP} \geq n$ .
- **B** – [inout] pointer to type. Array on the GPU (size depends on the value of `strideB`).  
On entry, the right hand side matrices  $B_i$ . On exit, the solution matrix  $X_i$  of each system in the batch.
- **ldb** – [in] int.  $\text{ldb} \geq n$ .  
The leading dimension of matrices  $B_i$ .

- **strideB** – [in] hipblasStride.

Stride from the start of one matrix  $B_i$  to the next one  $B_{i+1}$ . There is no restriction for the value of strideB. Normal use case is  $\text{strideB} \geq \text{ldb} * \text{nrhs}$ .

- **info** – [out] pointer to a int on the host.

If  $\text{info} = 0$ , successful exit. If  $\text{info} = j < 0$ , the  $j$ -th argument is invalid.

- **batchCount** – [in] int.  $\text{batchCount} \geq 0$ .

Number of instances (systems) in the batch.

### 1.8.4.3 hipblasXgetri + Batched, stridedBatched

*hipblasStatus\_t* **hipblasSgetriBatched**(*hipblasHandle\_t* handle, const int n, float \*const A[], const int lda, int \*ipiv, float \*const C[], const int ldc, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasDgetriBatched**(*hipblasHandle\_t* handle, const int n, double \*const A[], const int lda, int \*ipiv, double \*const C[], const int ldc, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasCgetriBatched**(*hipblasHandle\_t* handle, const int n, *hipblasComplex* \*const A[], const int lda, int \*ipiv, *hipblasComplex* \*const C[], const int ldc, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasZgetriBatched**(*hipblasHandle\_t* handle, const int n, *hipblasDoubleComplex* \*const A[], const int lda, int \*ipiv, *hipblasDoubleComplex* \*const C[], const int ldc, int \*info, const int batchCount)

SOLVER API.

getriBatched computes the inverse  $C_i = A_i^{-1}$  of a batch of general  $n$ -by- $n$  matrices  $A_i$ .

The inverse is computed by solving the linear system

$$A_i C_i = I$$

where  $I$  is the identity matrix, and  $A_i$  is factorized as  $A_i = P_i L_i U_i$  as given by *getrfBatched*.

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] hipblasHandle\_t.

- **n** – [in] int.  $n \geq 0$ .

The number of rows and columns of all matrices  $A_i$  in the batch.

- **A** – [in] array of pointers to type. Each pointer points to an array on the GPU of dimension  $\text{lda} * n$ .

The factors  $L_i$  and  $U_i$  of the factorization  $A_i = P_i * L_i * U_i$  returned by *getrfBatched*.



- **lda** – [in] int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **ipiv** – [in] pointer to int. Array on the GPU (the size depends on the value of strideP).  
The pivot indices returned by *getrfBatched*. ipiv can be passed in as a nullptr, this will assume that getrfBatched was called without partial pivoting.
- **C** – [out] array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldc \cdot n$ .  
If  $info[i] = 0$ , the inverse of matrices  $A_i$ . Otherwise, undefined.
- **ldc** – [in] int.  $ldc \geq n$ .  
Specifies the leading dimension of  $C_i$ .
- **info** – [out] pointer to int. Array of batchCount integers on the GPU.  
If  $info[i] = 0$ , successful exit for inversion of  $A_i$ . If  $info[i] = j > 0$ ,  $U_i$  is singular.  $U_i[j,j]$  is the first zero pivot.
- **batchCount** – [in] int.  $batchCount \geq 0$ .  
Number of matrices in the batch.

#### 1.8.4.4 hipblasXgeqrf + Batched, stridedBatched

*hipblasStatus\_t* **hipblasSgeqrf**(*hipblasHandle\_t* handle, const int m, const int n, float \*A, const int lda, float \*ipiv, int \*info)

*hipblasStatus\_t* **hipblasDgeqrf**(*hipblasHandle\_t* handle, const int m, const int n, double \*A, const int lda, double \*ipiv, int \*info)

*hipblasStatus\_t* **hipblasCgeqrf**(*hipblasHandle\_t* handle, const int m, const int n, *hipblasComplex* \*A, const int lda, *hipblasComplex* \*ipiv, int \*info)

*hipblasStatus\_t* **hipblasZgeqrf**(*hipblasHandle\_t* handle, const int m, const int n, *hipblasDoubleComplex* \*A, const int lda, *hipblasDoubleComplex* \*ipiv, int \*info)

SOLVER API.

geqrf computes a QR factorization of a general m-by-n matrix A.

The factorization has the form

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where R is upper triangular (upper trapezoidal if  $m < n$ ), and Q is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q = H_1 H_2 \cdots H_k, \quad \text{with } k = \min(m, n)$$

Each Householder matrix  $H_i$  is given by

$$H_i = I - \text{ipiv}[i] \cdot v_i v_i'$$

where the first  $i-1$  elements of the Householder vector  $v_i$  are zero, and  $v_i[i] = 1$ .

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] hipblasHandle\_t.
- **m** – [in] int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $\text{lda} \times n$ .  
On entry, the  $m$ -by- $n$  matrix to be factored. On exit, the elements on and above the diagonal contain the factor R; the elements below the diagonal are the last  $m - i$  elements of Householder vector  $v_i$ .
- **lda** – [in] int.  $\text{lda} \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [out] pointer to type. Array on the GPU of dimension  $\min(m,n)$ .  
The Householder scalars.
- **info** – [out] pointer to a int on the host.  
If  $\text{info} = 0$ , successful exit. If  $\text{info} = j < 0$ , the  $j$ -th argument is invalid.

*hipblasStatus\_t* **hipblasSgeqrfBatched**(*hipblasHandle\_t* handle, const int m, const int n, float \*const A[], const int lda, float \*const ipiv[], int \*info, const int batchSize)

*hipblasStatus\_t* **hipblasDgeqrfBatched**(*hipblasHandle\_t* handle, const int m, const int n, double \*const A[], const int lda, double \*const ipiv[], int \*info, const int batchSize)

*hipblasStatus\_t* **hipblasCgeqrfBatched**(*hipblasHandle\_t* handle, const int m, const int n, *hipblasComplex* \*const A[], const int lda, *hipblasComplex* \*const ipiv[], int \*info, const int batchSize)

*hipblasStatus\_t* **hipblasZgeqrfBatched**(*hipblasHandle\_t* handle, const int m, const int n, *hipblasDoubleComplex* \*const A[], const int lda, *hipblasDoubleComplex* \*const ipiv[], int \*info, const int batchSize)

SOLVER API.

geqrfBatched computes the QR factorization of a batch of general  $m$ -by- $n$  matrices.

The factorization of matrix  $A_i$  in the batch has the form

$$A_i = Q_i \begin{bmatrix} R_i \\ 0 \end{bmatrix}$$

where  $R_i$  is upper triangular (upper trapezoidal if  $m < n$ ), and  $Q_i$  is a  $m$ -by- $m$  orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_i = H_{i_1} H_{i_2} \cdots H_{i_k}, \quad \text{with } k = \min(m, n)$$

Each Householder matrix  $H_{i_j}$  is given by

$$H_{i_j} = I - \text{ipiv}_i[j] \cdot v_{i_j} v_{i_j}'$$

where the first  $j-1$  elements of Householder vector  $v_{i_j}$  are zero, and  $v_{i_j}[j] = 1$ .

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z

#### Parameters

- **handle** – [in] hipblasHandle\_t.
- **m** – [in] int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_i$  in the batch.
- **n** – [in] int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_i$  in the batch.
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $\text{lda} \times n$ .  
On entry, the  $m$ -by- $n$  matrices  $A_i$  to be factored. On exit, the elements on and above the diagonal contain the factor  $R_i$ . The elements below the diagonal are the last  $m - j$  elements of Householder vector  $v_{i_j}$ .
- **lda** – [in] int.  $\text{lda} \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **ipiv** – [out] array of pointers to type. Each pointer points to an array on the GPU of dimension  $\min(m, n)$ .  
Contains the vectors  $\text{ipiv}_i$  of corresponding Householder scalars.
- **info** – [out] pointer to a int on the host.  
If  $\text{info} = 0$ , successful exit. If  $\text{info} = k < 0$ , the  $k$ -th argument is invalid.
- **batchCount** – [in] int.  $\text{batchCount} \geq 0$ .  
Number of matrices in the batch.

*hipblasStatus\_t* **hipblasSgeqrfStridedBatched**(*hipblasHandle\_t* handle, const int m, const int n, float \*A, const int lda, const *hipblasStride* strideA, float \*ipiv, const *hipblasStride* strideP, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasDgeqrfStridedBatched**(*hipblasHandle\_t* handle, const int m, const int n, double \*A, const int lda, const *hipblasStride* strideA, double \*ipiv, const *hipblasStride* strideP, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasCgeqrfStridedBatched**(*hipblasHandle\_t* handle, const int m, const int n, *hipblasComplex* \*A, const int lda, const *hipblasStride* strideA, *hipblasComplex* \*ipiv, const *hipblasStride* strideP, int \*info, const int batchCount)

*hipblasStatus\_t* **hipblasZgeqrfStridedBatched**(*hipblasHandle\_t* handle, const int m, const int n, *hipblasDoubleComplex* \*A, const int lda, const *hipblasStride* strideA, *hipblasDoubleComplex* \*ipiv, const *hipblasStride* strideP, int \*info, const int batchCount)

#### SOLVER API.

geqrfStridedBatched computes the QR factorization of a batch of general m-by-n matrices.

The factorization of matrix  $A_i$  in the batch has the form

$$A_i = Q_i \begin{bmatrix} R_i \\ 0 \end{bmatrix}$$

where  $R_i$  is upper triangular (upper trapezoidal if  $m < n$ ), and  $Q_i$  is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_i = H_{i_1} H_{i_2} \cdots H_{i_k}, \quad \text{with } k = \min(m, n)$$

Each Householder matrix  $H_{i_j}$  is given by

$$H_{i_j} = I - \text{ipiv}_j[j] \cdot v_{i_j} v_{i_j}'$$

where the first j-1 elements of Householder vector  $v_{i_j}$  are zero, and  $v_{i_j}[j] = 1$ .

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : No support

#### Parameters

- **handle** – [in] *hipblasHandle\_t*.
- **m** – [in] int.  $m \geq 0$ .

The number of rows of all the matrices  $A_i$  in the batch.

- **n** – [in] int.  $n \geq 0$ .

The number of columns of all the matrices  $A_i$  in the batch.

- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of strideA).

On entry, the m-by-n matrices  $A_i$  to be factored. On exit, the elements on and above the diagonal contain the factor  $R_i$ . The elements below the diagonal are the last  $m - j$  elements of Householder vector  $v_{(i,j)}$ .

- **lda** – [in] int.  $lda \geq m$ .

Specifies the leading dimension of matrices  $A_i$ .

- **strideA** – [in] hipblasStride.

Stride from the start of one matrix  $A_i$  to the next one  $A_{(i+1)}$ . There is no restriction for the value of strideA. Normal use case is  $strideA \geq lda * n$ .

- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).

Contains the vectors  $ipiv_i$  of corresponding Householder scalars.

- **strideP** – [in] hipblasStride.

Stride from the start of one vector  $ipiv_i$  to the next one  $ipiv_{(i+1)}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq \min(m, n)$ .

- **info** – [out] pointer to a int on the host.

If  $info = 0$ , successful exit. If  $info = k < 0$ , the k-th argument is invalid.

- **batchCount** – [in] int.  $batchCount \geq 0$ .

Number of matrices in the batch.

### 1.8.5 BLAS Extensions

#### List of BLAS Extension Functions

- *hipblasGemmEx + Batched, StridedBatched*
- *hipblasTrsmEx + Batched, StridedBatched*
- *hipblasApyEx + Batched, StridedBatched*
- *hipblasDotEx + Batched, StridedBatched*
- *hipblasDotcEx + Batched, StridedBatched*
- *hipblasNrm2Ex + Batched, StridedBatched*
- *hipblasRotEx + Batched, StridedBatched*
- *hipblasScalEx + Batched, StridedBatched*

### 1.8.5.1 hipblasGemmEx + Batched, StridedBatched

*hipblasStatus\_t* **hipblasGemmEx**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const void \*alpha, const void \*A, *hipblasDatatype\_t* aType, int lda, const void \*B, *hipblasDatatype\_t* bType, int ldb, const void \*beta, void \*C, *hipblasDatatype\_t* cType, int ldc, *hipblasDatatype\_t* computeType, *hipblasGemmAlgo\_t* algo)

BLAS EX API.

gemmEx performs one of the matrix-matrix operations

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

where  $\text{op}(X)$  is one of

$\text{op}(X) = X$             **or**  
 $\text{op}(X) = X^{**T}$        **or**  
 $\text{op}(X) = X^{**H},$

alpha and beta are scalars, and A, B, and C are matrices, with  $\text{op}(A)$  an m by k matrix,  $\text{op}(B)$  a k by n matrix and C is a m by n matrix.

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

Note for int8 users - For rocBLAS backend, please read rocblas\_gemm\_ex documentation on int8 data layout requirements. hipBLAS makes the assumption that the data layout is in the preferred format for a given device as documented in rocBLAS.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **transA** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(A)$ .
- **transB** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(B)$ .
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **k** – [in] [int] matrix dimension k.
- **alpha** – [in] [const void \*] device pointer or host pointer specifying the scalar alpha. Same datatype as computeType.
- **A** – [in] [void \*] device pointer storing matrix A.
- **aType** – [in] [hipblasDatatype\_t] specifies the datatype of matrix A.
- **lda** – [in] [int] specifies the leading dimension of A.
- **B** – [in] [void \*] device pointer storing matrix B.
- **bType** – [in] [hipblasDatatype\_t] specifies the datatype of matrix B.
- **ldb** – [in] [int] specifies the leading dimension of B.
- **beta** – [in] [const void \*] device pointer or host pointer specifying the scalar beta. Same datatype as computeType.
- **C** – [in] [void \*] device pointer storing matrix C.
- **cType** – [in] [hipblasDatatype\_t] specifies the datatype of matrix C.

- **ldc** – [in] [int] specifies the leading dimension of C.
- **computeType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.
- **algo** – [in] [hipblasGemmAlgo\_t] enumerant specifying the algorithm type.

*hipblasStatus\_t* **hipblasGemmBatchedEx**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const void \*alpha, const void \*A[], *hipblasDatatype\_t* aType, int lda, const void \*B[], *hipblasDatatype\_t* bType, int ldb, const void \*beta, void \*C[], *hipblasDatatype\_t* cType, int ldc, int batchCount, *hipblasDatatype\_t* computeType, *hipblasGemmAlgo\_t* algo)

#### BLAS EX API.

gemmBatchedEx performs one of the batched matrix-matrix operations  $C_i = \alpha * \text{op}(A_i) * \text{op}(B_i) + \beta * C_i$ , for  $i = 1, \dots, \text{batchCount}$ . where  $\text{op}(X)$  is one of  $\text{op}(X) = X$  or  $\text{op}(X) = X^{**T}$  or  $\text{op}(X) = X^{**H}$ , alpha and beta are scalars, and A, B, and C are batched pointers to matrices, with  $\text{op}(A)$  an m by k by batchCount batched matrix,  $\text{op}(B)$  a k by n by batchCount batched matrix and C a m by n by batchCount batched matrix. The batched matrices are an array of pointers to matrices. The number of pointers to matrices is batchCount.

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

Note for int8 users - For rocBLAS backend, please read rocblas\_gemm\_batched\_ex documentation on int8 data layout requirements. hipBLAS makes the assumption that the data layout is in the preferred format for a given device as documented in rocBLAS.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **transA** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(A)$ .
- **transB** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(B)$ .
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **k** – [in] [int] matrix dimension k.
- **alpha** – [in] [const void \*] device pointer or host pointer specifying the scalar alpha. Same datatype as computeType.
- **A** – [in] [void \*] device pointer storing array of pointers to each matrix  $A_i$ .
- **aType** – [in] [hipblasDatatype\_t] specifies the datatype of each matrix  $A_i$ .
- **lda** – [in] [int] specifies the leading dimension of each  $A_i$ .
- **B** – [in] [void \*] device pointer storing array of pointers to each matrix  $B_i$ .
- **bType** – [in] [hipblasDatatype\_t] specifies the datatype of each matrix  $B_i$ .
- **ldb** – [in] [int] specifies the leading dimension of each  $B_i$ .
- **beta** – [in] [const void \*] device pointer or host pointer specifying the scalar beta. Same datatype as computeType.
- **C** – [in] [void \*] device array of device pointers to each matrix  $C_i$ .
- **cType** – [in] [hipblasDatatype\_t] specifies the datatype of each matrix  $C_i$ .
- **ldc** – [in] [int] specifies the leading dimension of each  $C_i$ .

- **batchCount** – [in] [int] number of gemm operations in the batch.
- **computeType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.
- **algo** – [in] [hipblasGemmAlgo\_t] enumerant specifying the algorithm type.

*hipblasStatus\_t* **hipblasGemmStridedBatchedEx**(*hipblasHandle\_t* handle, *hipblasOperation\_t* transA, *hipblasOperation\_t* transB, int m, int n, int k, const void \*alpha, const void \*A, *hipblasDatatype\_t* aType, int lda, *hipblasStride* strideA, const void \*B, *hipblasDatatype\_t* bType, int ldb, *hipblasStride* strideB, const void \*beta, void \*C, *hipblasDatatype\_t* cType, int ldc, *hipblasStride* strideC, int batchCount, *hipblasDatatype\_t* computeType, *hipblasGemmAlgo\_t* algo)

BLAS EX API.

gemmStridedBatchedEx performs one of the strided\_batched matrix-matrix operations

$$C_i = \alpha * \text{op}(A_i) * \text{op}(B_i) + \beta * C_i, \text{ for } i = 1, \dots, \text{batchCount}$$

where  $\text{op}(X)$  is one of

$$\begin{aligned} \text{op}(X) &= X && \text{or} \\ \text{op}(X) &= X^{**T} && \text{or} \\ \text{op}(X) &= X^{**H}, \end{aligned}$$

alpha and beta are scalars, and A, B, and C are strided\_batched matrices, with  $\text{op}(A)$  an m by k by batchCount strided\_batched matrix,  $\text{op}(B)$  a k by n by batchCount strided\_batched matrix and C a m by n by batchCount strided\_batched matrix.

The strided\_batched matrices are multiple matrices separated by a constant stride. The number of matrices is batchCount.

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

Note for int8 users - For rocBLAS backend, please read rocblas\_gemm\_strided\_batched\_ex documentation on int8 data layout requirements. hipBLAS makes the assumption that the data layout is in the preferred format for a given device as documented in rocBLAS.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **transA** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(A)$ .
- **transB** – [in] [hipblasOperation\_t] specifies the form of  $\text{op}(B)$ .
- **m** – [in] [int] matrix dimension m.
- **n** – [in] [int] matrix dimension n.
- **k** – [in] [int] matrix dimension k.
- **alpha** – [in] [const void \*] device pointer or host pointer specifying the scalar alpha. Same datatype as computeType.
- **A** – [in] [void \*] device pointer pointing to first matrix A<sub>1</sub>.
- **aType** – [in] [hipblasDatatype\_t] specifies the datatype of each matrix A<sub>i</sub>.
- **lda** – [in] [int] specifies the leading dimension of each A<sub>i</sub>.



- **strideA** – [in] [hipblasStride] specifies stride from start of one A<sub>i</sub> matrix to the next A<sub>(i + 1)</sub>.
- **B** – [in] [void \*] device pointer pointing to first matrix B<sub>1</sub>.
- **bType** – [in] [hipblasDatatype\_t] specifies the datatype of each matrix B<sub>i</sub>.
- **ldb** – [in] [int] specifies the leading dimension of each B<sub>i</sub>.
- **strideB** – [in] [hipblasStride] specifies stride from start of one B<sub>i</sub> matrix to the next B<sub>(i + 1)</sub>.
- **beta** – [in] [const void \*] device pointer or host pointer specifying the scalar beta. Same datatype as computeType.
- **C** – [in] [void \*] device pointer pointing to first matrix C<sub>1</sub>.
- **cType** – [in] [hipblasDatatype\_t] specifies the datatype of each matrix C<sub>i</sub>.
- **ldc** – [in] [int] specifies the leading dimension of each C<sub>i</sub>.
- **strideC** – [in] [hipblasStride] specifies stride from start of one C<sub>i</sub> matrix to the next C<sub>(i + 1)</sub>.
- **batchCount** – [in] [int] number of gemm operations in the batch.
- **computeType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.
- **algo** – [in] [hipblasGemmAlgo\_t] enumerant specifying the algorithm type.

### 1.8.5.2 hipblasTrsmEx + Batched, StridedBatched

*hipblasStatus\_t* **hipblasTrsmEx**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const void \*alpha, void \*A, int lda, void \*B, int ldb, const void \*invA, int invASize, *hipblasDatatype\_t* computeType)

BLAS EX API

trsmEx solves

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

where alpha is a scalar, X and B are m by n matrices, A is triangular matrix and op(A) is one of

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

The matrix X is overwritten on B.

This function gives the user the ability to reuse the invA matrix between runs. If invA == NULL, hipblasTrsmEx will automatically calculate invA on every run.

Setting up invA: The accepted invA matrix consists of the packed 128x128 inverses of the diagonal blocks of matrix A, followed by any smaller diagonal block that remains. To set up invA it is recommended that hipblas-TrtriBatched be used with matrix A as the input.

Device memory of size 128 x k should be allocated for invA ahead of time, where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT. The actual number of elements in invA should be passed as invASize.

To begin, hipblasTrtriBatched must be called on the full 128x128 sized diagonal blocks of matrix A. Below are the restricted parameters:

- n = 128

- `ldinvA = 128`
- `stride_invA = 128x128`
- `batchCount = k / 128,`

Then any remaining block may be added:

- `n = k % 128`
- `invA = invA + stride_invA * previousBatchCount`
- `ldinvA = 128`
- `batchCount = 1`

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] HIPBLAS\_SIDE\_LEFT:  $\text{op}(A)*X = \alpha*B$ . HIPBLAS\_SIDE\_RIGHT:  $X*\text{op}(A) = \alpha*B$ .
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: A is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: A is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_N:  $\text{op}(A) = A$ . HIPBLAS\_OP\_T:  $\text{op}(A) = A^T$ . HIPBLAS\_ON\_C:  $\text{op}(A) = A^H$ .
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: A is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: A is not assumed to be unit triangular.
- **m** – [in] [int] m specifies the number of rows of B.  $m \geq 0$ .
- **n** – [in] [int] n specifies the number of columns of B.  $n \geq 0$ .
- **alpha** – [in] [void \*] device pointer or host pointer specifying the scalar alpha. When alpha is `&zero` then A is not referenced, and B need not be set before entry.
- **A** – [in] [void \*] device pointer storing matrix A. of dimension ( lda, k ), where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if side = HIPBLAS\_SIDE\_LEFT,  $\text{lda} \geq \max(1, m)$ , if side = HIPBLAS\_SIDE\_RIGHT,  $\text{lda} \geq \max(1, n)$ .
- **B** – [inout] [void \*] device pointer storing matrix B. B is of dimension ( ldb, n ). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.
- **ldb** – [in] [int] ldb specifies the first dimension of B.  $\text{ldb} \geq \max(1, m)$ .
- **invA** – [in] [void \*] device pointer storing the inverse diagonal blocks of A. invA is of dimension ( ld\_invA, k ), where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT. ld\_invA must be equal to 128.
- **invASize** – [in] [int] invASize specifies the number of elements of device memory in invA.
- **computeType** – [in] [hipblasDatatype\_t] specifies the datatype of computation

*hipblasStatus\_t* **hipblasTrsmBatchedEx**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const void \*alpha, void \*A, int lda, void \*B, int ldb, int batchCount, const void \*invA, int invASize, *hipblasDatatype\_t* computeType)

BLAS EX API

trsmBatchedEx solves

$$\text{op}(A_i) * X_i = \alpha * B_i \text{ or } X_i * \text{op}(A_i) = \alpha * B_i,$$

for  $i = 1, \dots, \text{batchCount}$ ; and where  $\alpha$  is a scalar,  $X$  and  $B$  are arrays of  $m$  by  $n$  matrices,  $A$  is an array of triangular matrix and each  $\text{op}(A_i)$  is one of

$$\text{op}(A_i) = A_i \text{ or } \text{op}(A_i) = A_i^T \text{ or } \text{op}(A_i) = A_i^H.$$

Each matrix  $X_i$  is overwritten on  $B_i$ .

This function gives the user the ability to reuse the  $\text{invA}$  matrix between runs. If  $\text{invA} == \text{NULL}$ , `hipblasTrsmBatchedEx` will automatically calculate each  $\text{invA}_i$  on every run.

Setting up  $\text{invA}$ : Each accepted  $\text{invA}_i$  matrix consists of the packed  $128 \times 128$  inverses of the diagonal blocks of matrix  $A_i$ , followed by any smaller diagonal block that remains. To set up each  $\text{invA}_i$  it is recommended that `hipblasTrtriBatched` be used with matrix  $A_i$  as the input.  $\text{invA}$  is an array of pointers of  $\text{batchCount}$  length holding each  $\text{invA}_i$ .

Device memory of size  $128 \times k$  should be allocated for each  $\text{invA}_i$  ahead of time, where  $k$  is  $m$  when `HIPBLAS_SIDE_LEFT` and is  $n$  when `HIPBLAS_SIDE_RIGHT`. The actual number of elements in each  $\text{invA}_i$  should be passed as  $\text{invASize}$ .

To begin, `hipblasTrtriBatched` must be called on the full  $128 \times 128$  sized diagonal blocks of each matrix  $A_i$ . Below are the restricted parameters:

- $n = 128$
- $\text{ldinvA} = 128$
- $\text{stride\_invA} = 128 \times 128$
- $\text{batchCount} = k / 128$ ,

Then any remaining block may be added:

- $n = k \% 128$
- $\text{invA} = \text{invA} + \text{stride\_invA} * \text{previousBatchCount}$
- $\text{ldinvA} = 128$
- $\text{batchCount} = 1$

#### Parameters

- **handle** – [in] [`hipblasHandle_t`] handle to the `hipblas` library context queue.
- **side** – [in] [`hipblasSideMode_t`] `HIPBLAS_SIDE_LEFT`:  $\text{op}(A) * X = \alpha * B$ . `HIPBLAS_SIDE_RIGHT`:  $X * \text{op}(A) = \alpha * B$ .
- **uplo** – [in] [`hipblasFillMode_t`] `HIPBLAS_FILL_MODE_UPPER`: each  $A_i$  is an upper triangular matrix. `HIPBLAS_FILL_MODE_LOWER`: each  $A_i$  is a lower triangular matrix.
- **transA** – [in] [`hipblasOperation_t`] `HIPBLAS_OP_N`:  $\text{op}(A) = A$ . `HIPBLAS_OP_T`:  $\text{op}(A) = A^T$ . `HIPBLAS_OP_C`:  $\text{op}(A) = A^H$ .
- **diag** – [in] [`hipblasDiagType_t`] `HIPBLAS_DIAG_UNIT`: each  $A_i$  is assumed to be unit triangular. `HIPBLAS_DIAG_NON_UNIT`: each  $A_i$  is not assumed to be unit triangular.
- **m** – [in] [int]  $m$  specifies the number of rows of each  $B_i$ .  $m \geq 0$ .
- **n** – [in] [int]  $n$  specifies the number of columns of each  $B_i$ .  $n \geq 0$ .

- **alpha** – [in] [void \*] device pointer or host pointer alpha specifying the scalar alpha. When alpha is &zero then A is not referenced, and B need not be set before entry.
- **A** – [in] [void \*] device array of device pointers storing each matrix A<sub>i</sub>. each A<sub>i</sub> is of dimension ( lda, k ), where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of each A<sub>i</sub>. if side = HIPBLAS\_SIDE\_LEFT, lda >= max( 1, m ), if side = HIPBLAS\_SIDE\_RIGHT, lda >= max( 1, n ).
- **B** – [inout] [void \*] device array of device pointers storing each matrix B<sub>i</sub>. each B<sub>i</sub> is of dimension ( ldb, n ). Before entry, the leading m by n part of the array B<sub>i</sub> must contain the right-hand side matrix B<sub>i</sub>, and on exit is overwritten by the solution matrix X<sub>i</sub>
- **ldb** – [in] [int] ldb specifies the first dimension of each B<sub>i</sub>. ldb >= max( 1, m ).
- **batchCount** – [in] [int] specifies how many batches.
- **invA** – [in] [void \*] device array of device pointers storing the inverse diagonal blocks of each A<sub>i</sub>. each invA<sub>i</sub> is of dimension ( ld\_invA, k ), where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT. ld\_invA must be equal to 128.
- **invASize** – [in] [int] invASize specifies the number of elements of device memory in each invA<sub>i</sub>.
- **computeType** – [in] [hipblasDatatype\_t] specifies the datatype of computation

*hipblasStatus\_t* **hipblasTrsmStridedBatchedEx**(*hipblasHandle\_t* handle, *hipblasSideMode\_t* side, *hipblasFillMode\_t* uplo, *hipblasOperation\_t* transA, *hipblasDiagType\_t* diag, int m, int n, const void \*alpha, void \*A, int lda, *hipblasStride* strideA, void \*B, int ldb, *hipblasStride* strideB, int batchCount, const void \*invA, int invASize, *hipblasStride* strideInvA, *hipblasDatatype\_t* computeType)

#### BLAS EX API

trsmStridedBatchedEx solves

$$\text{op}(A_i) * X_i = \alpha * B_i \text{ or } X_i * \text{op}(A_i) = \alpha * B_i,$$

for  $i = 1, \dots, \text{batchCount}$ ; and where alpha is a scalar, X and B are strided batched m by n matrices, A is a strided batched triangular matrix and op(A<sub>i</sub>) is one of

$$\text{op}(A_i) = A_i \text{ or } \text{op}(A_i) = A_i^T \text{ or } \text{op}(A_i) = A_i^H.$$

Each matrix X<sub>i</sub> is overwritten on B<sub>i</sub>.

This function gives the user the ability to reuse each invA<sub>i</sub> matrix between runs. If invA == NULL, hipblas-TrsmStridedBatchedEx will automatically calculate each invA<sub>i</sub> on every run.

Setting up invA: Each accepted invA<sub>i</sub> matrix consists of the packed 128x128 inverses of the diagonal blocks of matrix A<sub>i</sub>, followed by any smaller diagonal block that remains. To set up invA<sub>i</sub> it is recommended that hipblasTrtriBatched be used with matrix A<sub>i</sub> as the input. invA is a contiguous piece of memory holding each invA<sub>i</sub>.

Device memory of size 128 x k should be allocated for each invA<sub>i</sub> ahead of time, where k is m when HIPBLAS\_SIDE\_LEFT and is n when HIPBLAS\_SIDE\_RIGHT. The actual number of elements in each invA<sub>i</sub> should be passed as invASize.

To begin, hipblasTrtriBatched must be called on the full 128x128 sized diagonal blocks of each matrix A<sub>i</sub>. Below are the restricted parameters:

- $n = 128$
- $ld_{invA} = 128$
- $stride_{invA} = 128 \times 128$
- $batchCount = k / 128$ ,

Then any remaining block may be added:

- $n = k \% 128$
- $invA = invA + stride_{invA} * previousBatchCount$
- $ld_{invA} = 128$
- $batchCount = 1$

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **side** – [in] [hipblasSideMode\_t] HIPBLAS\_SIDE\_LEFT:  $op(A)*X = \alpha*B$ . HIPBLAS\_SIDE\_RIGHT:  $X*op(A) = \alpha*B$ .
- **uplo** – [in] [hipblasFillMode\_t] HIPBLAS\_FILL\_MODE\_UPPER: each  $A_i$  is an upper triangular matrix. HIPBLAS\_FILL\_MODE\_LOWER: each  $A_i$  is a lower triangular matrix.
- **transA** – [in] [hipblasOperation\_t] HIPBLAS\_OP\_N:  $op(A) = A$ . HIPBLAS\_OP\_T:  $op(A) = A^T$ . HIPBLAS\_OP\_C:  $op(A) = A^H$ .
- **diag** – [in] [hipblasDiagType\_t] HIPBLAS\_DIAG\_UNIT: each  $A_i$  is assumed to be unit triangular. HIPBLAS\_DIAG\_NON\_UNIT: each  $A_i$  is not assumed to be unit triangular.
- **m** – [in] [int]  $m$  specifies the number of rows of each  $B_i$ .  $m \geq 0$ .
- **n** – [in] [int]  $n$  specifies the number of columns of each  $B_i$ .  $n \geq 0$ .
- **alpha** – [in] [void \*] device pointer or host pointer specifying the scalar alpha. When alpha is &zero then A is not referenced, and B need not be set before entry.
- **A** – [in] [void \*] device pointer storing matrix A. of dimension (  $lda, k$  ), where  $k$  is  $m$  when HIPBLAS\_SIDE\_LEFT and is  $n$  when HIPBLAS\_SIDE\_RIGHT only the upper/lower triangular part is accessed.
- **lda** – [in] [int]  $lda$  specifies the first dimension of A. if  $side = HIPBLAS_SIDE_LEFT$ ,  $lda \geq \max(1, m)$ , if  $side = HIPBLAS_SIDE_RIGHT$ ,  $lda \geq \max(1, n)$ .
- **strideA** – [in] [hipblasStride] The stride between each A matrix.
- **B** – [inout] [void \*] device pointer pointing to first matrix  $B_i$ . each  $B_i$  is of dimension (  $ldb, n$  ). Before entry, the leading  $m$  by  $n$  part of each array  $B_i$  must contain the right-hand side of matrix  $B_i$ , and on exit is overwritten by the solution matrix  $X_i$ .
- **ldb** – [in] [int]  $ldb$  specifies the first dimension of each  $B_i$ .  $ldb \geq \max(1, m)$ .
- **strideB** – [in] [hipblasStride] The stride between each  $B_i$  matrix.
- **batchCount** – [in] [int] specifies how many batches.
- **invA** – [in] [void \*] device pointer storing the inverse diagonal blocks of each  $A_i$ .  $invA$  points to the first  $invA_1$ . each  $invA_i$  is of dimension (  $ld_{invA}, k$  ), where  $k$  is  $m$  when HIPBLAS\_SIDE\_LEFT and is  $n$  when HIPBLAS\_SIDE\_RIGHT.  $ld_{invA}$  must be equal to 128.

- **invASize** – [in] [int] invASize specifies the number of elements of device memory in each invA\_i.
- **strideInvA** – [in] [hipblasStride] The stride between each invA matrix.
- **computeType** – [in] [hipblasDatatype\_t] specifies the datatype of computation

### 1.8.5.3 hipblasAxyEx + Batched, StridedBatched

*hipblasStatus\_t* **hipblasAxyEx**(*hipblasHandle\_t* handle, int n, const void \*alpha, *hipblasDatatype\_t* alphaType, const void \*x, *hipblasDatatype\_t* xType, int incx, void \*y, *hipblasDatatype\_t* yType, int incy, *hipblasDatatype\_t* executionType)

BLAS EX API.

axyEx computes constant alpha multiplied by vector x, plus vector y

$y := \alpha * x + y$

– Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x and y.
- **alpha** – [in] device pointer or host pointer to specify the scalar alpha.
- **alphaType** – [in] [hipblasDatatype\_t] specifies the datatype of alpha.
- **x** – [in] device pointer storing vector x.
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [inout] device pointer storing vector y.
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasAxyBatchedEx**(*hipblasHandle\_t* handle, int n, const void \*alpha, *hipblasDatatype\_t* alphaType, const void \*x, *hipblasDatatype\_t* xType, int incx, void \*y, *hipblasDatatype\_t* yType, int incy, int batchCount, *hipblasDatatype\_t* executionType)

BLAS EX API.

axyBatchedEx computes constant alpha multiplied by vector x, plus vector y over a set of batched vectors.

 $y := \alpha * x + y$ 

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each x\_i and y\_i.

- **alpha** – [in] device pointer or host pointer to specify the scalar alpha.
- **alphaType** – [in] [hipblasDatatype\_t] specifies the datatype of alpha.
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **y** – [inout] device array of device pointers storing each vector  $y_i$ .
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **batchCount** – [in] [int] number of instances in the batch.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasAxpYStridedBatchedEx**(*hipblasHandle\_t* handle, int n, const void \*alpha, *hipblasDatatype\_t* alphaType, const void \*x, *hipblasDatatype\_t* xType, int incx, *hipblasStride* stridex, void \*y, *hipblasDatatype\_t* yType, int incy, *hipblasStride* stridey, int batchCount, *hipblasDatatype\_t* executionType)

BLAS EX API.

axpyStridedBatchedEx computes constant alpha multiplied by vector x, plus vector y over a set of strided batched vectors.

$y := \alpha * x + y$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x_i$  and  $y_i$ .
- **alpha** – [in] device pointer or host pointer to specify the scalar alpha.
- **alphaType** – [in] [hipblasDatatype\_t] specifies the datatype of alpha.
- **x** – [in] device pointer to the first vector  $x_1$ .
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) to the next one ( $x_{i+1}$ ). There are no restrictions placed on stridex, however the user should take care to ensure that stridex is of appropriate size, for a typical case this means  $\text{stridex} \geq n * \text{incx}$ .
- **y** – [inout] device pointer to the first vector  $y_1$ .
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **stridey** – [in] [hipblasStride] stride from the start of one vector ( $y_i$ ) to the next one ( $y_{i+1}$ ). There are no restrictions placed on stridey, however the user should take care to ensure that stridey is of appropriate size, for a typical case this means  $\text{stridey} \geq n * \text{incy}$ .
- **batchCount** – [in] [int] number of instances in the batch.

- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

#### 1.8.5.4 hipblasDotEx + Batched, StridedBatched

*hipblasStatus\_t* **hipblasDotEx**(*hipblasHandle\_t* handle, int n, const void \*x, *hipblasDatatype\_t* xType, int incx, const void \*y, *hipblasDatatype\_t* yType, int incy, void \*result, *hipblasDatatype\_t* resultType, *hipblasDatatype\_t* executionType)

BLAS EX API.

dotEx performs the dot product of vectors x and y

```
result = x * y;
```

dotcEx performs the dot product of the conjugate of complex vector x and complex vector y

```
result = conjugate (x) * y;
```

– Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

##### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x and y.
- **x** – [in] device pointer storing vector x.
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **y** – [in] device pointer storing vector y.
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **result** – [inout] device pointer or host pointer to store the dot product. return is 0.0 if n <= 0.
- **resultType** – [in] [hipblasDatatype\_t] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasDotBatchedEx**(*hipblasHandle\_t* handle, int n, const void \*x, *hipblasDatatype\_t* xType, int incx, const void \*y, *hipblasDatatype\_t* yType, int incy, int batchCount, void \*result, *hipblasDatatype\_t* resultType, *hipblasDatatype\_t* executionType)

BLAS EX API.

dotBatchedEx performs a batch of dot products of vectors x and y

```
result_i = x_i * y_i;
```

dotcBatchedEx performs a batch of dot products of the conjugate of complex vector x and complex vector y

```
result_i = conjugate (x_i) * y_i;
```

where (x\_i, y\_i) is the i-th instance of the batch. x\_i and y\_i are vectors, for i = 1, ..., batchCount



- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x_i$  and  $y_i$ .
- **x** – [in] device array of device pointers storing each vector  $x_i$ .
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **y** – [in] device array of device pointers storing each vector  $y_i$ .
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [inout] device array or host array of batchCount size to store the dot products of each batch. return 0.0 for each element if  $n \leq 0$ .
- **resultType** – [in] [hipblasDatatype\_t] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasDotStridedBatchedEx**(*hipblasHandle\_t* handle, int n, const void \*x, *hipblasDatatype\_t* xType, int incx, *hipblasStride* stridex, const void \*y, *hipblasDatatype\_t* yType, int incy, *hipblasStride* stridey, int batchCount, void \*result, *hipblasDatatype\_t* resultType, *hipblasDatatype\_t* executionType)

BLAS EX API.

dotStridedBatchedEx performs a batch of dot products of vectors  $x$  and  $y$

```
result_i = x_i * y_i;
```

dotc\_strided\_batched\_ex performs a batch of dot products of the conjugate of complex vector  $x$  and complex vector  $y$

```
result_i = conjugate (x_i) * y_i;
```

where  $(x_i, y_i)$  is the  $i$ -th instance of the batch.  $x_i$  and  $y_i$  are vectors, for  $i = 1, \dots, \text{batchCount}$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x_i$  and  $y_i$ .
- **x** – [in] device pointer to the first vector ( $x_1$ ) in the batch.
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .

- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ )
- **y** – [in] device pointer to the first vector ( $y_1$ ) in the batch.
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **stridey** – [in] [hipblasStride] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [inout] device array or host array of batchCount size to store the dot products of each batch. return 0.0 for each element if  $n \leq 0$ .
- **resultType** – [in] [hipblasDatatype\_t] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

### 1.8.5.5 hipblasDotcEx + Batched, StridedBatched

*hipblasStatus\_t* **hipblasDotcEx**(*hipblasHandle\_t* handle, int n, const void \*x, *hipblasDatatype\_t* xType, int incx, const void \*y, *hipblasDatatype\_t* yType, int incy, void \*result, *hipblasDatatype\_t* resultType, *hipblasDatatype\_t* executionType)

BLAS EX API.

dotEx performs the dot product of vectors x and y

```
result = x * y;
```

dotcEx performs the dot product of the conjugate of complex vector x and complex vector y

```
result = conjugate (x) * y;
```

– Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x and y.
- **x** – [in] device pointer storing vector x.
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of vector x.
- **incx** – [in] [int] specifies the increment for the elements of y.
- **y** – [in] device pointer storing vector y.
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **result** – [inout] device pointer or host pointer to store the dot product. return is 0.0 if  $n \leq 0$ .
- **resultType** – [in] [hipblasDatatype\_t] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasDotcBatchedEx**(*hipblasHandle\_t* handle, int n, const void \*x, *hipblasDatatype\_t* xType, int incx, const void \*y, *hipblasDatatype\_t* yType, int incy, int batchCount, void \*result, *hipblasDatatype\_t* resultType, *hipblasDatatype\_t* executionType)

BLAS EX API.

dotBatchedEx performs a batch of dot products of vectors x and y

```
result_i = x_i * y_i;
```

dotcBatchedEx performs a batch of dot products of the conjugate of complex vector x and complex vector y

```
result_i = conjugate (x_i) * y_i;
```

where (x\_i, y\_i) is the i-th instance of the batch. x\_i and y\_i are vectors, for i = 1, ..., batchCount

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [*hipblasHandle\_t*] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each x\_i and y\_i.
- **x** – [in] device array of device pointers storing each vector x\_i.
- **xType** – [in] [*hipblasDatatype\_t*] specifies the datatype of each vector x\_i.
- **incx** – [in] [int] specifies the increment for the elements of each x\_i.
- **y** – [in] device array of device pointers storing each vector y\_i.
- **yType** – [in] [*hipblasDatatype\_t*] specifies the datatype of each vector y\_i.
- **incy** – [in] [int] specifies the increment for the elements of each y\_i.
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [inout] device array or host array of batchCount size to store the dot products of each batch. return 0.0 for each element if n <= 0.
- **resultType** – [in] [*hipblasDatatype\_t*] specifies the datatype of the result.
- **executionType** – [in] [*hipblasDatatype\_t*] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasDotcStridedBatchedEx**(*hipblasHandle\_t* handle, int n, const void \*x, *hipblasDatatype\_t* xType, int incx, *hipblasStride* stridex, const void \*y, *hipblasDatatype\_t* yType, int incy, *hipblasStride* stridey, int batchCount, void \*result, *hipblasDatatype\_t* resultType, *hipblasDatatype\_t* executionType)

BLAS EX API.

dotStridedBatchedEx performs a batch of dot products of vectors x and y

```
result_i = x_i * y_i;
```

dotc\_strided\_batched\_ex performs a batch of dot products of the conjugate of complex vector x and complex vector y

```
result_i = conjugate (x_i) * y_i;
```

where  $(x_i, y_i)$  is the  $i$ -th instance of the batch.  $x_i$  and  $y_i$  are vectors, for  $i = 1, \dots, \text{batchCount}$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in each  $x_i$  and  $y_i$ .
- **x** – [in] device pointer to the first vector ( $x_1$ ) in the batch.
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ )
- **y** – [in] device pointer to the first vector ( $y_1$ ) in the batch.
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment for the elements of each  $y_i$ .
- **stridey** – [in] [hipblasStride] stride from the start of one vector ( $y_i$ ) and the next one ( $y_{i+1}$ )
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [inout] device array or host array of batchCount size to store the dot products of each batch. return 0.0 for each element if  $n \leq 0$ .
- **resultType** – [in] [hipblasDatatype\_t] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

#### 1.8.5.6 hipblasNrm2Ex + Batched, StridedBatched

*hipblasStatus\_t* **hipblasNrm2Ex**(*hipblasHandle\_t* handle, int n, const void \*x, *hipblasDatatype\_t* xType, int incx, void \*result, *hipblasDatatype\_t* resultType, *hipblasDatatype\_t* executionType)

BLAS\_EX API.

nrm2Ex computes the euclidean norm of a real or complex vector

```
result := sqrt( x'*x ) for real vectors
result := sqrt( x**H*x ) for complex vectors
```

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x.
- **x** – [in] device pointer storing vector x.
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of the vector x.
- **incx** – [in] [int] specifies the increment for the elements of y.

- **result** – [inout] device pointer or host pointer to store the nrm2 product. return is 0.0 if n, incx<=0.
- **resultType** – [in] [hipblasDatatype\_t] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasNrm2BatchedEx**(*hipblasHandle\_t* handle, int n, const void \*x, *hipblasDatatype\_t* xType, int incx, int batchCount, void \*result, *hipblasDatatype\_t* resultType, *hipblasDatatype\_t* executionType)

BLAS\_EX API.

nrm2BatchedEx computes the euclidean norm over a batch of real or complex vectors

```
result := sqrt( x_i'*x_i ) for real vectors x, for i = 1, ..., batchCount
result := sqrt( x_i**H*x_i ) for complex vectors x, for i = 1, ..., batchCount
```

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each x\_i.
- **x** – [in] device array of device pointers storing each vector x\_i.
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector x\_i.
- **incx** – [in] [int] specifies the increment for the elements of each x\_i. incx must be > 0.
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [out] device pointer or host pointer to array of batchCount size for nrm2 results. return is 0.0 for each element if n <= 0, incx<=0.
- **resultType** – [in] [hipblasDatatype\_t] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasNrm2StridedBatchedEx**(*hipblasHandle\_t* handle, int n, const void \*x, *hipblasDatatype\_t* xType, int incx, *hipblasStride* stridex, int batchCount, void \*result, *hipblasDatatype\_t* resultType, *hipblasDatatype\_t* executionType)

BLAS\_EX API.

nrm2StridedBatchedEx computes the euclidean norm over a batch of real or complex vectors

```
:= sqrt( x_i'*x_i ) for real vectors x, for i = 1, ..., batchCount
:= sqrt( x_i**H*x_i ) for complex vectors, for i = 1, ..., batchCount
```

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each x\_i.

- **x** – [in] device pointer to the first vector  $x_1$ .
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ . incx must be  $> 0$ .
- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) and the next one ( $x_{i+1}$ ). There are no restrictions placed on stride\_x, however the user should take care to ensure that stride\_x is of appropriate size, for a typical case this means  $\text{stride\_x} \geq n * \text{incx}$ .
- **batchCount** – [in] [int] number of instances in the batch
- **result** – [out] device pointer or host pointer to array for storing contiguous batchCount results. return is 0.0 for each element if  $n \leq 0$ ,  $\text{incx} \leq 0$ .
- **resultType** – [in] [hipblasDatatype\_t] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

### 1.8.5.7 hipblasRotEx + Batched, StridedBatched

*hipblasStatus\_t hipblasRotEx(hipblasHandle\_t handle, int n, void \*x, hipblasDatatype\_t xType, int incx, void \*y, hipblasDatatype\_t yType, int incy, const void \*c, const void \*s, hipblasDatatype\_t csType, hipblasDatatype\_t executionType)*

BLAS EX API.

rotEx applies the Givens rotation matrix defined by  $c=\cos(\alpha)$  and  $s=\sin(\alpha)$  to vectors  $x$  and  $y$ . Scalars  $c$  and  $s$  may be stored in either host or device memory, location is specified by calling `hipblasSetPointerMode`.

In the case where `cs_type` is real:  $x := c * x + s * y$   $y := c * y - s * x$

In the case where `cs_type` is complex, the imaginary part of  $c$  is ignored:  $x := \text{real}(c) * x + s * y$   $y := \text{real}(c) * y - \text{conj}(s) * x$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in the  $x$  and  $y$  vectors.
- **x** – [inout] device pointer storing vector  $x$ .
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of vector  $x$ .
- **incx** – [in] [int] specifies the increment between elements of  $x$ .
- **y** – [inout] device pointer storing vector  $y$ .
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of vector  $y$ .
- **incy** – [in] [int] specifies the increment between elements of  $y$ .
- **c** – [in] device pointer or host pointer storing scalar cosine component of the rotation matrix.
- **s** – [in] device pointer or host pointer storing scalar sine component of the rotation matrix.
- **csType** – [in] [hipblasDatatype\_t] specifies the datatype of  $c$  and  $s$ .
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasRotBatchedEx**(*hipblasHandle\_t* handle, int n, void \*x, *hipblasDatatype\_t* xType, int incx, void \*y, *hipblasDatatype\_t* yType, int incy, const void \*c, const void \*s, *hipblasDatatype\_t* csType, int batchCount, *hipblasDatatype\_t* executionType)

BLAS EX API.

rotBatchedEx applies the Givens rotation matrix defined by  $c=\cos(\alpha)$  and  $s=\sin(\alpha)$  to batched vectors  $x_i$  and  $y_i$ , for  $i = 1, \dots, \text{batchCount}$ . Scalars  $c$  and  $s$  may be stored in either host or device memory, location is specified by calling `hipblasSetPointerMode`.

In the case where `cs_type` is real:  $x := c * x + s * y$   $y := c * y - s * x$

In the case where `cs_type` is complex, the imaginary part of  $c$  is ignored:  $x := \text{real}(c) * x + s * y$   $y := \text{real}(c) * y - \text{conj}(s) * x$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [`hipblasHandle_t`] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each  $x_i$  and  $y_i$  vectors.
- **x** – [inout] device array of device pointers storing each vector  $x_i$ .
- **xType** – [in] [`hipblasDatatype_t`] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment between elements of each  $x_i$ .
- **y** – [inout] device array of device pointers storing each vector  $y_i$ .
- **yType** – [in] [`hipblasDatatype_t`] specifies the datatype of each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment between elements of each  $y_i$ .
- **c** – [in] device pointer or host pointer to scalar cosine component of the rotation matrix.
- **s** – [in] device pointer or host pointer to scalar sine component of the rotation matrix.
- **csType** – [in] [`hipblasDatatype_t`] specifies the datatype of  $c$  and  $s$ .
- **batchCount** – [in] [int] the number of  $x$  and  $y$  arrays, i.e. the number of batches.
- **executionType** – [in] [`hipblasDatatype_t`] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasRotStridedBatchedEx**(*hipblasHandle\_t* handle, int n, void \*x, *hipblasDatatype\_t* xType, int incx, *hipblasStride* stridex, void \*y, *hipblasDatatype\_t* yType, int incy, *hipblasStride* stridey, const void \*c, const void \*s, *hipblasDatatype\_t* csType, int batchCount, *hipblasDatatype\_t* executionType)

BLAS Level 1 API.

rotStridedBatchedEx applies the Givens rotation matrix defined by  $c=\cos(\alpha)$  and  $s=\sin(\alpha)$  to strided batched vectors  $x_i$  and  $y_i$ , for  $i = 1, \dots, \text{batchCount}$ . Scalars  $c$  and  $s$  may be stored in either host or device memory, location is specified by calling `hipblasSetPointerMode`.

In the case where `cs_type` is real:  $x := c * x + s * y$   $y := c * y - s * x$

In the case where `cs_type` is complex, the imaginary part of  $c$  is ignored:  $x := \text{real}(c) * x + s * y$   $y := \text{real}(c) * y - \text{conj}(s) * x$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

**Parameters**

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] number of elements in each  $x_i$  and  $y_i$  vectors.
- **x** – [inout] device pointer to the first vector  $x_1$ .
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment between elements of each  $x_i$ .
- **stridex** – [in] [hipblasStride] specifies the increment from the beginning of  $x_i$  to the beginning of  $x_{i+1}$
- **y** – [inout] device pointer to the first vector  $y_1$ .
- **yType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $y_i$ .
- **incy** – [in] [int] specifies the increment between elements of each  $y_i$ .
- **stridey** – [in] [hipblasStride] specifies the increment from the beginning of  $y_i$  to the beginning of  $y_{i+1}$
- **c** – [in] device pointer or host pointer to scalar cosine component of the rotation matrix.
- **s** – [in] device pointer or host pointer to scalar sine component of the rotation matrix.
- **csType** – [in] [hipblasDatatype\_t] specifies the datatype of  $c$  and  $s$ .
- **batchCount** – [in] [int] the number of  $x$  and  $y$  arrays, i.e. the number of batches.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

**1.8.5.8 hipblasScalEx + Batched, StridedBatched**

*hipblasStatus\_t* **hipblasScalEx**(*hipblasHandle\_t* handle, int n, const void \*alpha, *hipblasDatatype\_t* alphaType, void \*x, *hipblasDatatype\_t* xType, int incx, *hipblasDatatype\_t* executionType)

BLAS EX API.

scalEx scales each element of vector  $x$  with scalar  $\alpha$ .

$$x := \alpha * x$$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

**Parameters**

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in  $x$ .
- **alpha** – [in] device pointer or host pointer for the scalar  $\alpha$ .
- **alphaType** – [in] [hipblasDatatype\_t] specifies the datatype of  $\alpha$ .
- **x** – [inout] device pointer storing vector  $x$ .
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of vector  $x$ .
- **incx** – [in] [int] specifies the increment for the elements of  $x$ .



- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasScalBatchedEx**(*hipblasHandle\_t* handle, int n, const void \*alpha, *hipblasDatatype\_t* alphaType, void \*x, *hipblasDatatype\_t* xType, int incx, int batchCount, *hipblasDatatype\_t* executionType)

BLAS EX API.

scalBatchedEx scales each element of each vector  $x_i$  with scalar alpha.

$$x_i := \alpha * x_i$$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x.
- **alpha** – [in] device pointer or host pointer for the scalar alpha.
- **alphaType** – [in] [hipblasDatatype\_t] specifies the datatype of alpha.
- **x** – [inout] device array of device pointers storing each vector  $x_i$ .
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .
- **batchCount** – [in] [int] number of instances in the batch.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

*hipblasStatus\_t* **hipblasScalStridedBatchedEx**(*hipblasHandle\_t* handle, int n, const void \*alpha, *hipblasDatatype\_t* alphaType, void \*x, *hipblasDatatype\_t* xType, int incx, *hipblasStride* stridex, int batchCount, *hipblasDatatype\_t* executionType)

BLAS EX API.

scalStridedBatchedEx scales each element of vector x with scalar alpha over a set of strided batched vectors.

$x := \alpha * x$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

#### Parameters

- **handle** – [in] [hipblasHandle\_t] handle to the hipblas library context queue.
- **n** – [in] [int] the number of elements in x.
- **alpha** – [in] device pointer or host pointer for the scalar alpha.
- **alphaType** – [in] [hipblasDatatype\_t] specifies the datatype of alpha.
- **x** – [inout] device pointer to the first vector  $x_1$ .
- **xType** – [in] [hipblasDatatype\_t] specifies the datatype of each vector  $x_i$ .
- **incx** – [in] [int] specifies the increment for the elements of each  $x_i$ .

- **stridex** – [in] [hipblasStride] stride from the start of one vector ( $x_i$ ) to the next one ( $x_{i+1}$ ). There are no restrictions placed on stridex, however the user should take care to ensure that stridex is of appropriate size, for a typical case this means  $\text{stridex} \geq n * \text{incx}$ .
- **batchCount** – [in] [int] number of instances in the batch.
- **executionType** – [in] [hipblasDatatype\_t] specifies the datatype of computation.

## 1.8.6 Auxiliary

### 1.8.6.1 hipblasCreate

*hipblasStatus\_t* **hipblasCreate**(*hipblasHandle\_t* \*handle)  
Create hipblas handle.

### 1.8.6.2 hipblasDestroy

*hipblasStatus\_t* **hipblasDestroy**(*hipblasHandle\_t* handle)  
Destroys the library context created using *hipblasCreate()*

### 1.8.6.3 hipblasSetStream

*hipblasStatus\_t* **hipblasSetStream**(*hipblasHandle\_t* handle, *hipStream\_t* streamId)  
Set stream for handle.

### 1.8.6.4 hipblasGetStream

*hipblasStatus\_t* **hipblasGetStream**(*hipblasHandle\_t* handle, *hipStream\_t* \*streamId)  
Get stream[0] for handle.

### 1.8.6.5 hipblasSetPointerMode

*hipblasStatus\_t* **hipblasSetPointerMode**(*hipblasHandle\_t* handle, *hipblasPointerMode\_t* mode)  
Set hipblas pointer mode.

### 1.8.6.6 hipblasGetPointerMode

*hipblasStatus\_t* **hipblasGetPointerMode**(*hipblasHandle\_t* handle, *hipblasPointerMode\_t* \*mode)  
Get hipblas pointer mode.

### 1.8.6.7 hipblasSetVector

*hipblasStatus\_t* **hipblasSetVector**(int n, int elemSize, const void \*x, int incx, void \*y, int incy)  
copy vector from host to device

#### Parameters

- **n** – [in] [int] number of elements in the vector
- **elemSize** – [in] [int] Size of both vectors in bytes
- **x** – [in] pointer to vector on the host

- **incx** – [in] [int] specifies the increment for the elements of the vector
- **y** – [out] pointer to vector on the device
- **incy** – [in] [int] specifies the increment for the elements of the vector

#### 1.8.6.8 hipblasGetVector

*hipblasStatus\_t* **hipblasGetVector**(int n, int elemSize, const void \*x, int incx, void \*y, int incy)  
copy vector from device to host

##### Parameters

- **n** – [in] [int] number of elements in the vector
- **elemSize** – [in] [int] Size of both vectors in bytes
- **x** – [in] pointer to vector on the device
- **incx** – [in] [int] specifies the increment for the elements of the vector
- **y** – [out] pointer to vector on the host
- **incy** – [in] [int] specifies the increment for the elements of the vector

#### 1.8.6.9 hipblasSetMatrix

*hipblasStatus\_t* **hipblasSetMatrix**(int rows, int cols, int elemSize, const void \*AP, int lda, void \*BP, int ldb)  
copy matrix from host to device

##### Parameters

- **rows** – [in] [int] number of rows in matrices
- **cols** – [in] [int] number of columns in matrices
- **elemSize** – [in] [int] number of bytes per element in the matrix
- **AP** – [in] pointer to matrix on the host
- **lda** – [in] [int] specifies the leading dimension of A, lda >= rows
- **BP** – [out] pointer to matrix on the GPU
- **ldb** – [in] [int] specifies the leading dimension of B, ldb >= rows

#### 1.8.6.10 hipblasGetMatrix

*hipblasStatus\_t* **hipblasGetMatrix**(int rows, int cols, int elemSize, const void \*AP, int lda, void \*BP, int ldb)  
copy matrix from device to host

##### Parameters

- **rows** – [in] [int] number of rows in matrices
- **cols** – [in] [int] number of columns in matrices
- **elemSize** – [in] [int] number of bytes per element in the matrix
- **AP** – [in] pointer to matrix on the GPU
- **lda** – [in] [int] specifies the leading dimension of A, lda >= rows
- **BP** – [out] pointer to matrix on the host

- **ldb** – **[in]** [int] specifies the leading dimension of B,  $\text{ldb} \geq \text{rows}$

#### 1.8.6.11 hipblasSetVectorAsync

*hipblasStatus\_t* **hipblasSetVectorAsync**(int n, int elemSize, const void \*x, int incx, void \*y, int incy, hipStream\_t stream)

asynchronously copy vector from host to device

hipblasSetVectorAsync copies a vector from pinned host memory to device memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

##### Parameters

- **n** – **[in]** [int] number of elements in the vector
- **elemSize** – **[in]** [int] number of bytes per element in the matrix
- **x** – **[in]** pointer to vector on the host
- **incx** – **[in]** [int] specifies the increment for the elements of the vector
- **y** – **[out]** pointer to vector on the device
- **incy** – **[in]** [int] specifies the increment for the elements of the vector
- **stream** – **[in]** specifies the stream into which this transfer request is queued

#### 1.8.6.12 hipblasGetVectorAsync

*hipblasStatus\_t* **hipblasGetVectorAsync**(int n, int elemSize, const void \*x, int incx, void \*y, int incy, hipStream\_t stream)

asynchronously copy vector from device to host

hipblasGetVectorAsync copies a vector from pinned host memory to device memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

##### Parameters

- **n** – **[in]** [int] number of elements in the vector
- **elemSize** – **[in]** [int] number of bytes per element in the matrix
- **x** – **[in]** pointer to vector on the device
- **incx** – **[in]** [int] specifies the increment for the elements of the vector
- **y** – **[out]** pointer to vector on the host
- **incy** – **[in]** [int] specifies the increment for the elements of the vector
- **stream** – **[in]** specifies the stream into which this transfer request is queued

### 1.8.6.13 hipblasSetMatrixAsync

*hipblasStatus\_t* **hipblasSetMatrixAsync**(int rows, int cols, int elemSize, const void \*AP, int lda, void \*BP, int ldb, hipStream\_t stream)

asynchronously copy matrix from host to device

hipblasSetMatrixAsync copies a matrix from pinned host memory to device memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

#### Parameters

- **rows** – [in] [int] number of rows in matrices
- **cols** – [in] [int] number of columns in matrices
- **elemSize** – [in] [int] number of bytes per element in the matrix
- **AP** – [in] pointer to matrix on the host
- **lda** – [in] [int] specifies the leading dimension of A, lda >= rows
- **BP** – [out] pointer to matrix on the GPU
- **ldb** – [in] [int] specifies the leading dimension of B, ldb >= rows
- **stream** – [in] specifies the stream into which this transfer request is queued

### 1.8.6.14 hipblasGetMatrixAsync

*hipblasStatus\_t* **hipblasGetMatrixAsync**(int rows, int cols, int elemSize, const void \*AP, int lda, void \*BP, int ldb, hipStream\_t stream)

asynchronously copy matrix from device to host

hipblasGetMatrixAsync copies a matrix from device memory to pinned host memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

#### Parameters

- **rows** – [in] [int] number of rows in matrices
- **cols** – [in] [int] number of columns in matrices
- **elemSize** – [in] [int] number of bytes per element in the matrix
- **AP** – [in] pointer to matrix on the GPU
- **lda** – [in] [int] specifies the leading dimension of A, lda >= rows
- **BP** – [out] pointer to matrix on the host
- **ldb** – [in] [int] specifies the leading dimension of B, ldb >= rows
- **stream** – [in] specifies the stream into which this transfer request is queued

### 1.8.6.15 hipblasSetAtomsMode

*hipblasStatus\_t* **hipblasSetAtomsMode**(*hipblasHandle\_t* handle, *hipblasAtomsMode\_t* atoms\_mode)  
Set hipblasSetAtomsMode.

### 1.8.6.16 hipblasGetAtomsMode

*hipblasStatus\_t* **hipblasGetAtomsMode**(*hipblasHandle\_t* handle, *hipblasAtomsMode\_t* \*atoms\_mode)  
Get hipblasSetAtomsMode.

### 1.8.6.17 hipblasStatusToString

const char \***hipblasStatusToString**(*hipblasStatus\_t* status)

HIPBLAS Auxiliary API

hipblasStatusToString

Returns string representing hipblasStatus\_t value

**Parameters** **status** – [in] [hipblasStatus\_t] hipBLAS status to convert to string

## 1.9 Contributing

### 1.9.1 Pull-request guidelines

Our code contribution guidelines closely follows the model of [GitHub pull-requests](#). The hipBLAS repository follows a workflow which dictates a /master branch where releases are cut, and a /develop branch which serves as an integration branch for new code. Pull requests should:

- target the **develop** branch for integration
- ensure code builds successfully.
- do not break existing test cases
- new unit tests should integrate within the existing googletest framework.
- tests must have good code coverage
- code must also have benchmark tests, and performance must approach the compute bound limit or memory bound limit.

### 1.9.2 Coding Guidelines:

- Do not use unnamed namespaces inside of header files.
- Use either **template** or **inline** (or both) for functions defined outside of classes in header files.
- Do not declare namespace-scope (not class-scope) functions **static** inside of header files unless there is a very good reason, that the function does not have any non-const **static** local variables, and that it is acceptable that each compilation unit will have its own independent definition of the function and its **static** local variables. (static class member functions defined in header files are okay.)
- Use **static** for **constexpr** **template** variables until C++17, after which **constexpr** variables become **inline** variables, and thus can be defined in multiple compilation units. It is okay if the **constexpr** variables remain **static** in C++17; it just means there might be a little bit of redundancy between compilation units.

### 1.9.2.1 Format

C and C++ code is formatted using `clang-format`. To run `clang-format` use the version in the `/opt/rocm/llvm/bin` directory. Please do not use your system's built-in `clang-format`, as this may be an older version that will result in different results.

To format a file, use:

```
/opt/rocm/llvm/bin/clang-format -style=file -i <path-to-source-file>
```

To format all files, run the following script in rocBLAS directory:

```
#!/bin/bash
git ls-files -z *.cc *.cpp *.h *.hpp *.cl *.h.in *.hpp.in *.cpp.in | xargs -0 /opt/rocm/
  ↪ llvm/bin/clang-format -style=file -i
```

Also, githooks can be installed to format the code per-commit:

```
./githooks/install
```

### 1.9.3 Static Code Analysis

`cppcheck` is an open-source static analysis tool. This project uses this tool for performing static code analysis.

Users can use the following command to run `cppcheck` locally to generate the report for all files.

```
$ cd hipBLAS
$ cppcheck --enable=all --inconclusive --library=googletest --inline-suppr -i./build --
  ↪ suppressions-list=./CppCheckSuppressions.txt --template="{file}:{line}: {severity}:
  ↪ {id} :{message}" . 2> cppcheck_report.txt
```

For more information on the command line options, refer to the `cppcheck` manual on the web.





## DESIGN DOCUMENTATION

### 2.1 Clients

There are two client executables that can be used with hipBLAS. They are,

1. hipblas-bench
2. hipblas-test

These two clients can be built by following the instructions in the [Building and Installing hipBLAS github page](#). After building the hipBLAS clients, they can be found in the directory `hipBLAS/build/release/clients/staging`.

The next two sections will cover a brief explanation and the usage of each hipBLAS client.

#### 2.1.1 hipblas-bench

hipblas-bench is used to measure performance and to verify the correctness of hipBLAS functions.

It has a command line interface. For more information:

```
./hipblas-bench --help
```

For example, to measure performance of sgemm:

```
./hipblas-bench -f gemm -r f32_r --transposeA N --transposeB N -m 4096 -n 4096 -k 4096 --  
↪ alpha 1 --lda 4096 --ldb 4096 --beta 0 --ldc 4096
```

On a vega20 machine the above command outputs a performance of 11941.5 Gflops below:

```
transA,transB,M,N,K,alpha,lda,ldb,beta,ldc,hipblas-Gflops,us  
N,N,4096,4096,4096,1,4096,4096,0,4096,11941.5,11509.4
```

A useful way of finding the parameters that can be used with `./hipblas-bench -f gemm` is to turn on logging by setting environment variable `ROCBLAS_LAYER=2`. For example if the user runs:

```
ROCBLAS_LAYER=2 ./hipblas-bench -f gemm -i 1 -j 0
```

The above command will log:

```
./rocblas-bench -f gemm -r f32_r --transposeA N --transposeB N -m 128 -n 128 -k 128 --  
↪ alpha 1 --lda 128 --ldb 128 --beta 0 --ldc 128
```

The user can copy and change the above command. For example, to change the datatype to IEEE-64 bit and the size to 2048:

```
./hipblas-bench -f gemm -r f64_r --transposeA N --transposeB N -m 2048 -n 2048 -k 2048 --  
↪ alpha 1 --lda 2048 --ldb 2048 --beta 0 --ldc 2048
```

Logging affects performance, so only use it to log the command to copy and change, then run the command without logging to measure performance.

Note that hipblas-bench also has the flag `-v 1` for correctness checks.

### 2.1.2 hipblas-test

hipblas-test is used in performing hipBLAS unit tests and it uses Googletest framework.

To run the hipblas tests:

```
./hipblas-test
```

## H

- hipblasAtomicMode\_t (C++ enum), 12
- hipblasAtomicMode\_t::HIPBLAS\_ATOMICS\_ALLOWED (C++ enumerator), 12
- hipblasAtomicMode\_t::HIPBLAS\_ATOMICS\_NOT\_ALLOWED (C++ enumerator), 12
- hipblasAxyBatchedEx (C++ function), 192
- hipblasAxyEx (C++ function), 192
- hipblasAxyStridedBatchedEx (C++ function), 193
- hipblasBfdot (C++ function), 23
- hipblasBfdotBatched (C++ function), 24
- hipblasBfdotStridedBatched (C++ function), 26
- hipblasBfloat16 (C++ struct), 8
- hipblasCaxpy (C++ function), 19
- hipblasCaxpyBatched (C++ function), 19
- hipblasCaxpyStridedBatched (C++ function), 20
- hipblasCcopy (C++ function), 21
- hipblasCcopyBatched (C++ function), 22
- hipblasCcopyStridedBatched (C++ function), 22
- hipblasCdgmm (C++ function), 167
- hipblasCdgmmBatched (C++ function), 168
- hipblasCdgmmStridedBatched (C++ function), 169
- hipblasCdotc (C++ function), 23
- hipblasCdotcBatched (C++ function), 25
- hipblasCdotcStridedBatched (C++ function), 26
- hipblasCdotu (C++ function), 24
- hipblasCdotuBatched (C++ function), 25
- hipblasCdotuStridedBatched (C++ function), 26
- hipblasCgbmv (C++ function), 44
- hipblasCgbmvBatched (C++ function), 45
- hipblasCgbmvStridedBatched (C++ function), 47
- hipblasCgeam (C++ function), 148
- hipblasCgeamBatched (C++ function), 149
- hipblasCgeamStridedBatched (C++ function), 150
- hipblasCgemm (C++ function), 117
- hipblasCgemmBatched (C++ function), 119
- hipblasCgemmStridedBatched (C++ function), 120
- hipblasCgemv (C++ function), 48
- hipblasCgemvBatched (C++ function), 49
- hipblasCgemvStridedBatched (C++ function), 50
- hipblasCgeqrf (C++ function), 179
- hipblasCgeqrfBatched (C++ function), 180
- hipblasCgeqrfStridedBatched (C++ function), 182
- hipblasCgerc (C++ function), 52
- hipblasCgercBatched (C++ function), 53
- hipblasCgercStridedBatched (C++ function), 54
- hipblasCgeru (C++ function), 52
- hipblasCgeruBatched (C++ function), 53
- hipblasCgeruStridedBatched (C++ function), 54
- hipblasCgetrf (C++ function), 170
- hipblasCgetrfBatched (C++ function), 171
- hipblasCgetrfStridedBatched (C++ function), 172
- hipblasCgetriBatched (C++ function), 178
- hipblasCgetrs (C++ function), 174
- hipblasCgetrsBatched (C++ function), 175
- hipblasCgetrsStridedBatched (C++ function), 176
- hipblasChbmv (C++ function), 55
- hipblasChbmvBatched (C++ function), 56
- hipblasChbmvStridedBatched (C++ function), 57
- hipblasChemmm (C++ function), 151
- hipblasChemmmBatched (C++ function), 152
- hipblasChemmmStridedBatched (C++ function), 153
- hipblasChemv (C++ function), 59
- hipblasChemvBatched (C++ function), 60
- hipblasChemvStridedBatched (C++ function), 61
- hipblasCher (C++ function), 62
- hipblasCher2 (C++ function), 64
- hipblasCher2Batched (C++ function), 65
- hipblasCher2k (C++ function), 128
- hipblasCher2kBatched (C++ function), 129
- hipblasCher2kStridedBatched (C++ function), 130
- hipblasCher2StridedBatched (C++ function), 66
- hipblasCherBatched (C++ function), 62
- hipblasCherk (C++ function), 121
- hipblasCherkBatched (C++ function), 122
- hipblasCherkStridedBatched (C++ function), 123
- hipblasCherkx (C++ function), 124
- hipblasCherkxBatched (C++ function), 125
- hipblasCherkxStridedBatched (C++ function), 127
- hipblasCherStridedBatched (C++ function), 63
- hipblasChpmv (C++ function), 67
- hipblasChpmvBatched (C++ function), 68
- hipblasChpmvStridedBatched (C++ function), 69
- hipblasChpr (C++ function), 71

hipblasChpr2 (C++ *function*), 74  
hipblasChpr2Batched (C++ *function*), 74  
hipblasChpr2StridedBatched (C++ *function*), 76  
hipblasChprBatched (C++ *function*), 71  
hipblasChprStridedBatched (C++ *function*), 72  
hipblasComplex (C++ *struct*), 8  
hipblasCreate (C++ *function*), 204  
hipblasCrot (C++ *function*), 29  
hipblasCrotBatched (C++ *function*), 30  
hipblasCrotg (C++ *function*), 32  
hipblasCrotgBatched (C++ *function*), 32  
hipblasCrotgStridedBatched (C++ *function*), 33  
hipblasCrotStridedBatched (C++ *function*), 31  
hipblasCscal (C++ *function*), 39  
hipblasCscalBatched (C++ *function*), 39  
hipblasCscalStridedBatched (C++ *function*), 40  
hipblasCspr (C++ *function*), 82  
hipblasCsprBatched (C++ *function*), 83  
hipblasCsprStridedBatched (C++ *function*), 84  
hipblasCsrot (C++ *function*), 29  
hipblasCsrotBatched (C++ *function*), 30  
hipblasCsrotStridedBatched (C++ *function*), 31  
hipblasCsscal (C++ *function*), 39  
hipblasCsscalBatched (C++ *function*), 40  
hipblasCsscalStridedBatched (C++ *function*), 40  
hipblasCswap (C++ *function*), 41  
hipblasCswapBatched (C++ *function*), 42  
hipblasCswapStridedBatched (C++ *function*), 43  
hipblasCsymm (C++ *function*), 132  
hipblasCsymmBatched (C++ *function*), 133  
hipblasCsymmStridedBatched (C++ *function*), 134  
hipblasCsymv (C++ *function*), 88  
hipblasCsymvBatched (C++ *function*), 89  
hipblasCsymvStridedBatched (C++ *function*), 90  
hipblasCsyrr (C++ *function*), 92  
hipblasCsyrr2 (C++ *function*), 95  
hipblasCsyrr2Batched (C++ *function*), 96  
hipblasCsyrr2k (C++ *function*), 139  
hipblasCsyrr2kBatched (C++ *function*), 141  
hipblasCsyrr2kStridedBatched (C++ *function*), 142  
hipblasCsyrr2StridedBatched (C++ *function*), 97  
hipblasCsyrrBatched (C++ *function*), 93  
hipblasCsyrrk (C++ *function*), 136  
hipblasCsyrrkBatched (C++ *function*), 137  
hipblasCsyrrkStridedBatched (C++ *function*), 138  
hipblasCsyrrkx (C++ *function*), 143  
hipblasCsyrrkxBatched (C++ *function*), 145  
hipblasCsyrrkxStridedBatched (C++ *function*), 146  
hipblasCsyrrStridedBatched (C++ *function*), 94  
hipblasCtbmv (C++ *function*), 98  
hipblasCtbmvBatched (C++ *function*), 99  
hipblasCtbmvStridedBatched (C++ *function*), 100  
hipblasCtbsv (C++ *function*), 102  
hipblasCtbsvBatched (C++ *function*), 103  
hipblasCtbsvStridedBatched (C++ *function*), 104  
hipblasCtpmv (C++ *function*), 105  
hipblasCtpmvBatched (C++ *function*), 106  
hipblasCtpmvStridedBatched (C++ *function*), 107  
hipblasCtpsv (C++ *function*), 108  
hipblasCtpsvBatched (C++ *function*), 109  
hipblasCtpsvStridedBatched (C++ *function*), 110  
hipblasCtrmm (C++ *function*), 155  
hipblasCtrmmBatched (C++ *function*), 156  
hipblasCtrmmStridedBatched (C++ *function*), 158  
hipblasCtrmv (C++ *function*), 111  
hipblasCtrmvBatched (C++ *function*), 112  
hipblasCtrmvStridedBatched (C++ *function*), 113  
hipblasCtrsm (C++ *function*), 159  
hipblasCtrsmBatched (C++ *function*), 161  
hipblasCtrsmStridedBatched (C++ *function*), 162  
hipblasCtrsv (C++ *function*), 114  
hipblasCtrsvBatched (C++ *function*), 115  
hipblasCtrsvStridedBatched (C++ *function*), 116  
hipblasCtrtri (C++ *function*), 164  
hipblasCtrtriBatched (C++ *function*), 164  
hipblasCtrtriStridedBatched (C++ *function*), 165  
hipblasDasum (C++ *function*), 17  
hipblasDasumBatched (C++ *function*), 17  
hipblasDasumStridedBatched (C++ *function*), 18  
hipblasDatatype\_t (C++ *enum*), 11  
hipblasDatatype\_t::HIPBLAS\_C\_16B (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_C\_16F (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_C\_32F (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_C\_32I (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_C\_32U (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_C\_64F (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_C\_8I (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_C\_8U (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_R\_16B (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_R\_16F (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_R\_32F (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_R\_32I (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_R\_32U (C++ *enumerator*), 11  
hipblasDatatype\_t::HIPBLAS\_R\_64F (C++ *enumerator*), 11

hipblasDatatype\_t::HIPBLAS\_R\_8I (C++ *enumerator*), 11  
 hipblasDatatype\_t::HIPBLAS\_R\_8U (C++ *enumerator*), 11  
 hipblasDaxpy (C++ *function*), 19  
 hipblasDaxpyBatched (C++ *function*), 19  
 hipblasDaxpyStridedBatched (C++ *function*), 20  
 hipblasDcopy (C++ *function*), 21  
 hipblasDcopyBatched (C++ *function*), 22  
 hipblasDcopyStridedBatched (C++ *function*), 22  
 hipblasDdgemm (C++ *function*), 167  
 hipblasDdgemmBatched (C++ *function*), 167  
 hipblasDdgemmStridedBatched (C++ *function*), 168  
 hipblasDdot (C++ *function*), 23  
 hipblasDdotBatched (C++ *function*), 24  
 hipblasDdotStridedBatched (C++ *function*), 26  
 hipblasDestroy (C++ *function*), 204  
 hipblasDgbmv (C++ *function*), 44  
 hipblasDgbmvBatched (C++ *function*), 45  
 hipblasDgbmvStridedBatched (C++ *function*), 47  
 hipblasDgeam (C++ *function*), 148  
 hipblasDgeamBatched (C++ *function*), 149  
 hipblasDgeamStridedBatched (C++ *function*), 150  
 hipblasDgemm (C++ *function*), 117  
 hipblasDgemmBatched (C++ *function*), 119  
 hipblasDgemmStridedBatched (C++ *function*), 120  
 hipblasDgemv (C++ *function*), 48  
 hipblasDgemvBatched (C++ *function*), 49  
 hipblasDgemvStridedBatched (C++ *function*), 50  
 hipblasDgeqrf (C++ *function*), 179  
 hipblasDgeqrfBatched (C++ *function*), 180  
 hipblasDgeqrfStridedBatched (C++ *function*), 182  
 hipblasDger (C++ *function*), 52  
 hipblasDgerBatched (C++ *function*), 53  
 hipblasDgerStridedBatched (C++ *function*), 54  
 hipblasDgetrf (C++ *function*), 170  
 hipblasDgetrfBatched (C++ *function*), 171  
 hipblasDgetrfStridedBatched (C++ *function*), 172  
 hipblasDgetriBatched (C++ *function*), 178  
 hipblasDgetrs (C++ *function*), 174  
 hipblasDgetrsBatched (C++ *function*), 175  
 hipblasDgetrsStridedBatched (C++ *function*), 176  
 hipblasDiagType\_t (C++ *enum*), 10  
 hipblasDiagType\_t::HIPBLAS\_DIAG\_NON\_UNIT (C++ *enumerator*), 10  
 hipblasDiagType\_t::HIPBLAS\_DIAG\_UNIT (C++ *enumerator*), 10  
 hipblasDnrm2 (C++ *function*), 27  
 hipblasDnrm2Batched (C++ *function*), 28  
 hipblasDnrm2StridedBatched (C++ *function*), 28  
 hipblasDotBatchedEx (C++ *function*), 194  
 hipblasDotcBatchedEx (C++ *function*), 196  
 hipblasDotcEx (C++ *function*), 196  
 hipblasDotcStridedBatchedEx (C++ *function*), 197  
 hipblasDotEx (C++ *function*), 194  
 hipblasDotStridedBatchedEx (C++ *function*), 195  
 hipblasDoubleComplex (C++ *struct*), 8  
 hipblasDrot (C++ *function*), 29  
 hipblasDrotBatched (C++ *function*), 30  
 hipblasDrotg (C++ *function*), 32  
 hipblasDrotgBatched (C++ *function*), 32  
 hipblasDrotgStridedBatched (C++ *function*), 33  
 hipblasDrotm (C++ *function*), 34  
 hipblasDrotmBatched (C++ *function*), 35  
 hipblasDrotmg (C++ *function*), 36  
 hipblasDrotmgBatched (C++ *function*), 37  
 hipblasDrotmgStridedBatched (C++ *function*), 38  
 hipblasDrotmStridedBatched (C++ *function*), 35  
 hipblasDrotStridedBatched (C++ *function*), 31  
 hipblasDsmbv (C++ *function*), 77  
 hipblasDsmbvBatched (C++ *function*), 78  
 hipblasDsmbvStridedBatched (C++ *function*), 78  
 hipblasDscal (C++ *function*), 39  
 hipblasDscalBatched (C++ *function*), 39  
 hipblasDscalStridedBatched (C++ *function*), 40  
 hipblasDspmv (C++ *function*), 80  
 hipblasDspmvBatched (C++ *function*), 80  
 hipblasDspmvStridedBatched (C++ *function*), 81  
 hipblasDspr (C++ *function*), 82  
 hipblasDspr2 (C++ *function*), 85  
 hipblasDspr2Batched (C++ *function*), 86  
 hipblasDspr2StridedBatched (C++ *function*), 87  
 hipblasDsprBatched (C++ *function*), 83  
 hipblasDsprStridedBatched (C++ *function*), 84  
 hipblasDswap (C++ *function*), 41  
 hipblasDswapBatched (C++ *function*), 42  
 hipblasDswapStridedBatched (C++ *function*), 42  
 hipblasDsymm (C++ *function*), 132  
 hipblasDsymmBatched (C++ *function*), 133  
 hipblasDsymmStridedBatched (C++ *function*), 134  
 hipblasDsymv (C++ *function*), 88  
 hipblasDsymvBatched (C++ *function*), 89  
 hipblasDsymvStridedBatched (C++ *function*), 90  
 hipblasDsyr (C++ *function*), 92  
 hipblasDsyr2 (C++ *function*), 95  
 hipblasDsyr2Batched (C++ *function*), 95  
 hipblasDsyr2k (C++ *function*), 139  
 hipblasDsyr2kBatched (C++ *function*), 140  
 hipblasDsyr2kStridedBatched (C++ *function*), 142  
 hipblasDsyr2StridedBatched (C++ *function*), 96  
 hipblasDsyrBatched (C++ *function*), 93  
 hipblasDsyrk (C++ *function*), 136  
 hipblasDsyrkBatched (C++ *function*), 137  
 hipblasDsyrkStridedBatched (C++ *function*), 138  
 hipblasDsyrkx (C++ *function*), 143  
 hipblasDsyrkxBatched (C++ *function*), 145  
 hipblasDsyrkxStridedBatched (C++ *function*), 146  
 hipblasDsyrStridedBatched (C++ *function*), 93



hipblasDtbmv (C++ function), 98  
hipblasDtbmvBatched (C++ function), 99  
hipblasDtbmvStridedBatched (C++ function), 100  
hipblasDtbsv (C++ function), 102  
hipblasDtbsvBatched (C++ function), 103  
hipblasDtbsvStridedBatched (C++ function), 104  
hipblasDtpmv (C++ function), 105  
hipblasDtpmvBatched (C++ function), 106  
hipblasDtpmvStridedBatched (C++ function), 107  
hipblasDtpsv (C++ function), 108  
hipblasDtpsvBatched (C++ function), 109  
hipblasDtpsvStridedBatched (C++ function), 110  
hipblasDtrmm (C++ function), 155  
hipblasDtrmmBatched (C++ function), 156  
hipblasDtrmmStridedBatched (C++ function), 158  
hipblasDtrmv (C++ function), 111  
hipblasDtrmvBatched (C++ function), 112  
hipblasDtrmvStridedBatched (C++ function), 113  
hipblasDtrsm (C++ function), 159  
hipblasDtrsmBatched (C++ function), 161  
hipblasDtrsmStridedBatched (C++ function), 162  
hipblasDtrsv (C++ function), 114  
hipblasDtrsvBatched (C++ function), 115  
hipblasDtrsvStridedBatched (C++ function), 116  
hipblasDtrtri (C++ function), 164  
hipblasDtrtriBatched (C++ function), 164  
hipblasDtrtriStridedBatched (C++ function), 165  
hipblasDzasum (C++ function), 17  
hipblasDzasumBatched (C++ function), 17  
hipblasDzasumStridedBatched (C++ function), 18  
hipblasDznrm2 (C++ function), 27  
hipblasDznrm2Batched (C++ function), 28  
hipblasDznrm2StridedBatched (C++ function), 28  
hipblasFillMode\_t (C++ enum), 10  
hipblasFillMode\_t::HIPBLAS\_FILL\_MODE\_FULL  
(C++ enumerator), 10  
hipblasFillMode\_t::HIPBLAS\_FILL\_MODE\_LOWER  
(C++ enumerator), 10  
hipblasFillMode\_t::HIPBLAS\_FILL\_MODE\_UPPER  
(C++ enumerator), 10  
hipblasGemmAlgo\_t (C++ enum), 12  
hipblasGemmAlgo\_t::HIPBLAS\_GEMM\_DEFAULT  
(C++ enumerator), 12  
hipblasGemmBatchedEx (C++ function), 185  
hipblasGemmEx (C++ function), 184  
hipblasGemmStridedBatchedEx (C++ function), 186  
hipblasGetAtomicsMode (C++ function), 208  
hipblasGetMatrix (C++ function), 205  
hipblasGetMatrixAsync (C++ function), 207  
hipblasGetPointerMode (C++ function), 204  
hipblasGetStream (C++ function), 204  
hipblasGetVector (C++ function), 205  
hipblasGetVectorAsync (C++ function), 206  
hipblasHalf (C++ type), 7  
hipblasHandle\_t (C++ type), 7  
hipblasHaxpy (C++ function), 19  
hipblasHaxpyBatched (C++ function), 19  
hipblasHaxpyStridedBatched (C++ function), 20  
hipblasHdot (C++ function), 23  
hipblasHdotBatched (C++ function), 24  
hipblasHdotStridedBatched (C++ function), 25  
hipblasHgemv (C++ function), 117  
hipblasHgemvBatched (C++ function), 118  
hipblasHgemvStridedBatched (C++ function), 120  
hipblasIcamax (C++ function), 13  
hipblasIcamaxBatched (C++ function), 13  
hipblasIcamaxStridedBatched (C++ function), 14  
hipblasIcamin (C++ function), 15  
hipblasIcaminBatched (C++ function), 15  
hipblasIcaminStridedBatched (C++ function), 16  
hipblasIdamax (C++ function), 13  
hipblasIdamaxBatched (C++ function), 13  
hipblasIdamaxStridedBatched (C++ function), 14  
hipblasIdamin (C++ function), 15  
hipblasIdaminBatched (C++ function), 15  
hipblasIdaminStridedBatched (C++ function), 16  
hipblasInt8 (C++ type), 7  
hipblasIsamax (C++ function), 13  
hipblasIsamaxBatched (C++ function), 13  
hipblasIsamaxStridedBatched (C++ function), 14  
hipblasIsamin (C++ function), 15  
hipblasIsaminBatched (C++ function), 15  
hipblasIsaminStridedBatched (C++ function), 16  
hipblasIzamax (C++ function), 13  
hipblasIzamaxBatched (C++ function), 13  
hipblasIzamaxStridedBatched (C++ function), 14  
hipblasIzamin (C++ function), 15  
hipblasIzaminBatched (C++ function), 15  
hipblasIzaminStridedBatched (C++ function), 16  
hipblasNrm2BatchedEx (C++ function), 199  
hipblasNrm2Ex (C++ function), 198  
hipblasNrm2StridedBatchedEx (C++ function), 199  
hipblasOperation\_t (C++ enum), 9  
hipblasOperation\_t::HIPBLAS\_OP\_C (C++ enumerator), 9  
hipblasOperation\_t::HIPBLAS\_OP\_N (C++ enumerator), 9  
hipblasOperation\_t::HIPBLAS\_OP\_T (C++ enumerator), 9  
hipblasPointerMode\_t (C++ enum), 9  
hipblasPointerMode\_t::HIPBLAS\_POINTER\_MODE\_DEVICE  
(C++ enumerator), 9  
hipblasPointerMode\_t::HIPBLAS\_POINTER\_MODE\_HOST  
(C++ enumerator), 9  
hipblasRotBatchedEx (C++ function), 200  
hipblasRotEx (C++ function), 200  
hipblasRotStridedBatchedEx (C++ function), 201  
hipblasSasum (C++ function), 17

hipblasSasumBatched (C++ function), 17  
 hipblasSasumStridedBatched (C++ function), 18  
 hipblasSaxpy (C++ function), 19  
 hipblasSaxpyBatched (C++ function), 19  
 hipblasSaxpyStridedBatched (C++ function), 20  
 hipblasScalBatchedEx (C++ function), 203  
 hipblasScalEx (C++ function), 202  
 hipblasScalStridedBatchedEx (C++ function), 203  
 hipblasScasum (C++ function), 17  
 hipblasScasumBatched (C++ function), 17  
 hipblasScasumStridedBatched (C++ function), 18  
 hipblasScnrm2 (C++ function), 27  
 hipblasScnrm2Batched (C++ function), 28  
 hipblasScnrm2StridedBatched (C++ function), 28  
 hipblasScopy (C++ function), 21  
 hipblasScopyBatched (C++ function), 22  
 hipblasScopyStridedBatched (C++ function), 22  
 hipblasSdgm (C++ function), 167  
 hipblasSdgmBatched (C++ function), 167  
 hipblasSdgmStridedBatched (C++ function), 168  
 hipblasSdot (C++ function), 23  
 hipblasSdotBatched (C++ function), 24  
 hipblasSdotStridedBatched (C++ function), 26  
 hipblasSetAtomicsMode (C++ function), 208  
 hipblasSetMatrix (C++ function), 205  
 hipblasSetMatrixAsync (C++ function), 207  
 hipblasSetPointerMode (C++ function), 204  
 hipblasSetStream (C++ function), 204  
 hipblasSetVector (C++ function), 204  
 hipblasSetVectorAsync (C++ function), 206  
 hipblasSgbmv (C++ function), 44  
 hipblasSgbmvBatched (C++ function), 45  
 hipblasSgbmvStridedBatched (C++ function), 47  
 hipblasSgeam (C++ function), 148  
 hipblasSgeamBatched (C++ function), 149  
 hipblasSgeamStridedBatched (C++ function), 150  
 hipblasSgemm (C++ function), 117  
 hipblasSgemmBatched (C++ function), 119  
 hipblasSgemmStridedBatched (C++ function), 120  
 hipblasSgemv (C++ function), 48  
 hipblasSgemvBatched (C++ function), 49  
 hipblasSgemvStridedBatched (C++ function), 50  
 hipblasSgeqrf (C++ function), 179  
 hipblasSgeqrfBatched (C++ function), 180  
 hipblasSgeqrfStridedBatched (C++ function), 181  
 hipblasSger (C++ function), 52  
 hipblasSgerBatched (C++ function), 53  
 hipblasSgerStridedBatched (C++ function), 54  
 hipblasSgetrf (C++ function), 170  
 hipblasSgetrfBatched (C++ function), 171  
 hipblasSgetrfStridedBatched (C++ function), 172  
 hipblasSgetriBatched (C++ function), 178  
 hipblasSgetrs (C++ function), 174  
 hipblasSgetrsBatched (C++ function), 175  
 hipblasSgetrsStridedBatched (C++ function), 176  
 hipblasSideMode\_t (C++ enum), 10  
 hipblasSideMode\_t::HIPBLAS\_SIDE\_BOTH (C++ enumerator), 10  
 hipblasSideMode\_t::HIPBLAS\_SIDE\_LEFT (C++ enumerator), 10  
 hipblasSideMode\_t::HIPBLAS\_SIDE\_RIGHT (C++ enumerator), 10  
 hipblasSnrm2 (C++ function), 27  
 hipblasSnrm2Batched (C++ function), 27  
 hipblasSnrm2StridedBatched (C++ function), 28  
 hipblasSrot (C++ function), 29  
 hipblasSrotBatched (C++ function), 30  
 hipblasSrotg (C++ function), 32  
 hipblasSrotgBatched (C++ function), 32  
 hipblasSrotgStridedBatched (C++ function), 33  
 hipblasSrotm (C++ function), 34  
 hipblasSrotmBatched (C++ function), 35  
 hipblasSrotmg (C++ function), 36  
 hipblasSrotmgBatched (C++ function), 37  
 hipblasSrotmgStridedBatched (C++ function), 38  
 hipblasSrotmStridedBatched (C++ function), 35  
 hipblasSrotStridedBatched (C++ function), 31  
 hipblasSsbmv (C++ function), 77  
 hipblasSsbmvBatched (C++ function), 78  
 hipblasSsbmvStridedBatched (C++ function), 78  
 hipblasSscal (C++ function), 39  
 hipblasSscalBatched (C++ function), 39  
 hipblasSscalStridedBatched (C++ function), 40  
 hipblasSspmv (C++ function), 80  
 hipblasSspmvBatched (C++ function), 80  
 hipblasSspmvStridedBatched (C++ function), 81  
 hipblasSspr (C++ function), 82  
 hipblasSspr2 (C++ function), 85  
 hipblasSspr2Batched (C++ function), 86  
 hipblasSspr2StridedBatched (C++ function), 87  
 hipblasSsprBatched (C++ function), 83  
 hipblasSsprStridedBatched (C++ function), 84  
 hipblasSswap (C++ function), 41  
 hipblasSswapBatched (C++ function), 42  
 hipblasSswapStridedBatched (C++ function), 42  
 hipblasSsymm (C++ function), 132  
 hipblasSsymmBatched (C++ function), 133  
 hipblasSsymmStridedBatched (C++ function), 134  
 hipblasSsymv (C++ function), 88  
 hipblasSsymvBatched (C++ function), 89  
 hipblasSsymvStridedBatched (C++ function), 90  
 hipblasSsy (C++ function), 92  
 hipblasSsy2 (C++ function), 95  
 hipblasSsy2Batched (C++ function), 95  
 hipblasSsy2k (C++ function), 139  
 hipblasSsy2kBatched (C++ function), 140  
 hipblasSsy2kStridedBatched (C++ function), 142  
 hipblasSsy2StridedBatched (C++ function), 96

hipblasSsyBatched (C++ function), 92  
hipblasSsyk (C++ function), 136  
hipblasSsykBatched (C++ function), 137  
hipblasSsykStridedBatched (C++ function), 138  
hipblasSsykx (C++ function), 143  
hipblasSsykxBatched (C++ function), 145  
hipblasSsykxStridedBatched (C++ function), 146  
hipblasSsyStridedBatched (C++ function), 93  
hipblasStatus\_t (C++ enum), 8  
hipblasStatus\_t::HIPBLAS\_STATUS\_ALLOC\_FAILED (C++ enumerator), 8  
hipblasStatus\_t::HIPBLAS\_STATUS\_ARCH\_MISMATCH (C++ enumerator), 9  
hipblasStatus\_t::HIPBLAS\_STATUS\_EXECUTION\_FAILED (C++ enumerator), 8  
hipblasStatus\_t::HIPBLAS\_STATUS\_HANDLE\_IS\_NULL (C++ enumerator), 9  
hipblasStatus\_t::HIPBLAS\_STATUS\_INTERNAL\_ERROR (C++ enumerator), 8  
hipblasStatus\_t::HIPBLAS\_STATUS\_INVALID\_ENUM (C++ enumerator), 9  
hipblasStatus\_t::HIPBLAS\_STATUS\_INVALID\_VALUE (C++ enumerator), 8  
hipblasStatus\_t::HIPBLAS\_STATUS\_MAPPING\_ERROR (C++ enumerator), 8  
hipblasStatus\_t::HIPBLAS\_STATUS\_NOT\_INITIALIZED (C++ enumerator), 8  
hipblasStatus\_t::HIPBLAS\_STATUS\_NOT\_SUPPORTED (C++ enumerator), 8  
hipblasStatus\_t::HIPBLAS\_STATUS\_SUCCESS (C++ enumerator), 8  
hipblasStatus\_t::HIPBLAS\_STATUS\_UNKNOWN (C++ enumerator), 9  
hipblasStatusToString (C++ function), 208  
hipblasStbmv (C++ function), 98  
hipblasStbmvBatched (C++ function), 99  
hipblasStbmvStridedBatched (C++ function), 100  
hipblasStbsv (C++ function), 102  
hipblasStbsvBatched (C++ function), 103  
hipblasStbsvStridedBatched (C++ function), 104  
hipblasStpmv (C++ function), 105  
hipblasStpmvBatched (C++ function), 106  
hipblasStpmvStridedBatched (C++ function), 107  
hipblasStpsv (C++ function), 108  
hipblasStpsvBatched (C++ function), 109  
hipblasStpsvStridedBatched (C++ function), 110  
hipblasStride (C++ type), 7  
hipblasStrmm (C++ function), 155  
hipblasStrmmBatched (C++ function), 156  
hipblasStrmmStridedBatched (C++ function), 157  
hipblasStrmv (C++ function), 111  
hipblasStrmvBatched (C++ function), 112  
hipblasStrmvStridedBatched (C++ function), 113  
hipblasStrsm (C++ function), 159  
hipblasStrsmBatched (C++ function), 161  
hipblasStrsmStridedBatched (C++ function), 162  
hipblasStrsv (C++ function), 114  
hipblasStrsvBatched (C++ function), 115  
hipblasStrsvStridedBatched (C++ function), 116  
hipblasStrtri (C++ function), 164  
hipblasStrtriBatched (C++ function), 164  
hipblasStrtriStridedBatched (C++ function), 165  
hipblasTrsmBatchedEx (C++ function), 188  
hipblasTrsmEx (C++ function), 187  
hipblasTrsmStridedBatchedEx (C++ function), 190  
hipblasZaxpy (C++ function), 19  
hipblasZaxpyBatched (C++ function), 20  
hipblasZaxpyStridedBatched (C++ function), 20  
hipblasZcopy (C++ function), 21  
hipblasZcopyBatched (C++ function), 22  
hipblasZcopyStridedBatched (C++ function), 22  
hipblasZdgemm (C++ function), 167  
hipblasZdgemmBatched (C++ function), 168  
hipblasZdgemmStridedBatched (C++ function), 169  
hipblasZdotc (C++ function), 24  
hipblasZdotcBatched (C++ function), 25  
hipblasZdotcStridedBatched (C++ function), 26  
hipblasZdotu (C++ function), 24  
hipblasZdotuBatched (C++ function), 25  
hipblasZdotuStridedBatched (C++ function), 26  
hipblasZdrot (C++ function), 29  
hipblasZdrotBatched (C++ function), 30  
hipblasZdrotStridedBatched (C++ function), 31  
hipblasZdscal (C++ function), 39  
hipblasZdscalBatched (C++ function), 40  
hipblasZdscalStridedBatched (C++ function), 41  
hipblasZgbmv (C++ function), 44  
hipblasZgbmvBatched (C++ function), 46  
hipblasZgbmvStridedBatched (C++ function), 47  
hipblasZgeam (C++ function), 148  
hipblasZgeamBatched (C++ function), 149  
hipblasZgeamStridedBatched (C++ function), 150  
hipblasZgemm (C++ function), 118  
hipblasZgemmBatched (C++ function), 119  
hipblasZgemmStridedBatched (C++ function), 120  
hipblasZgemv (C++ function), 48  
hipblasZgemvBatched (C++ function), 49  
hipblasZgemvStridedBatched (C++ function), 50  
hipblasZgeqrf (C++ function), 179  
hipblasZgeqrfBatched (C++ function), 180  
hipblasZgeqrfStridedBatched (C++ function), 182  
hipblasZgerc (C++ function), 52  
hipblasZgercBatched (C++ function), 53  
hipblasZgercStridedBatched (C++ function), 54  
hipblasZgeru (C++ function), 52  
hipblasZgeruBatched (C++ function), 53  
hipblasZgeruStridedBatched (C++ function), 54  
hipblasZgetrf (C++ function), 170



hipblasZgetrfBatched (C++ function), 171  
 hipblasZgetrfStridedBatched (C++ function), 172  
 hipblasZgetriBatched (C++ function), 178  
 hipblasZgetrs (C++ function), 174  
 hipblasZgetrsBatched (C++ function), 175  
 hipblasZgetrsStridedBatched (C++ function), 176  
 hipblasZhbmvm (C++ function), 55  
 hipblasZhbmvmBatched (C++ function), 56  
 hipblasZhbmvmStridedBatched (C++ function), 57  
 hipblasZhemm (C++ function), 151  
 hipblasZhemmBatched (C++ function), 152  
 hipblasZhemmStridedBatched (C++ function), 153  
 hipblasZhemv (C++ function), 59  
 hipblasZhemvBatched (C++ function), 60  
 hipblasZhemvStridedBatched (C++ function), 61  
 hipblasZher (C++ function), 62  
 hipblasZher2 (C++ function), 64  
 hipblasZher2Batched (C++ function), 65  
 hipblasZher2k (C++ function), 128  
 hipblasZher2kBatched (C++ function), 129  
 hipblasZher2kStridedBatched (C++ function), 130  
 hipblasZher2StridedBatched (C++ function), 66  
 hipblasZherBatched (C++ function), 63  
 hipblasZherk (C++ function), 121  
 hipblasZherkBatched (C++ function), 122  
 hipblasZherkStridedBatched (C++ function), 123  
 hipblasZherkx (C++ function), 124  
 hipblasZherkxBatched (C++ function), 126  
 hipblasZherkxStridedBatched (C++ function), 127  
 hipblasZherStridedBatched (C++ function), 63  
 hipblasZhpmv (C++ function), 67  
 hipblasZhpmvBatched (C++ function), 68  
 hipblasZhpmvStridedBatched (C++ function), 69  
 hipblasZhpr (C++ function), 71  
 hipblasZhpr2 (C++ function), 74  
 hipblasZhpr2Batched (C++ function), 75  
 hipblasZhpr2StridedBatched (C++ function), 76  
 hipblasZhprBatched (C++ function), 71  
 hipblasZhprStridedBatched (C++ function), 72  
 hipblasZrot (C++ function), 29  
 hipblasZrotBatched (C++ function), 30  
 hipblasZrotg (C++ function), 32  
 hipblasZrotgBatched (C++ function), 33  
 hipblasZrotgStridedBatched (C++ function), 33  
 hipblasZrotStridedBatched (C++ function), 31  
 hipblasZscal (C++ function), 39  
 hipblasZscalBatched (C++ function), 40  
 hipblasZscalStridedBatched (C++ function), 40  
 hipblasZspr (C++ function), 82  
 hipblasZsprBatched (C++ function), 83  
 hipblasZsprStridedBatched (C++ function), 84  
 hipblasZswap (C++ function), 41  
 hipblasZswapBatched (C++ function), 42  
 hipblasZswapStridedBatched (C++ function), 43  
 hipblasZsymm (C++ function), 132  
 hipblasZsymmBatched (C++ function), 133  
 hipblasZsymmStridedBatched (C++ function), 134  
 hipblasZsymv (C++ function), 88  
 hipblasZsymvBatched (C++ function), 89  
 hipblasZsymvStridedBatched (C++ function), 91  
 hipblasZsyr (C++ function), 92  
 hipblasZsyr2 (C++ function), 95  
 hipblasZsyr2Batched (C++ function), 96  
 hipblasZsyr2k (C++ function), 139  
 hipblasZsyr2kBatched (C++ function), 141  
 hipblasZsyr2kStridedBatched (C++ function), 142  
 hipblasZsyr2StridedBatched (C++ function), 97  
 hipblasZsyrBatched (C++ function), 93  
 hipblasZsyrr (C++ function), 136  
 hipblasZsyrrBatched (C++ function), 137  
 hipblasZsyrrStridedBatched (C++ function), 138  
 hipblasZsyrrx (C++ function), 144  
 hipblasZsyrrxBatched (C++ function), 145  
 hipblasZsyrrxStridedBatched (C++ function), 146  
 hipblasZsyrStridedBatched (C++ function), 94  
 hipblasZtbmv (C++ function), 98  
 hipblasZtbmvBatched (C++ function), 99  
 hipblasZtbmvStridedBatched (C++ function), 100  
 hipblasZtbsv (C++ function), 102  
 hipblasZtbsvBatched (C++ function), 103  
 hipblasZtbsvStridedBatched (C++ function), 104  
 hipblasZtpmv (C++ function), 105  
 hipblasZtpmvBatched (C++ function), 106  
 hipblasZtpmvStridedBatched (C++ function), 107  
 hipblasZtpsv (C++ function), 108  
 hipblasZtpsvBatched (C++ function), 109  
 hipblasZtpsvStridedBatched (C++ function), 110  
 hipblasZtrmm (C++ function), 155  
 hipblasZtrmmBatched (C++ function), 156  
 hipblasZtrmmStridedBatched (C++ function), 158  
 hipblasZtrmv (C++ function), 111  
 hipblasZtrmvBatched (C++ function), 112  
 hipblasZtrmvStridedBatched (C++ function), 113  
 hipblasZtrsm (C++ function), 159  
 hipblasZtrsmBatched (C++ function), 161  
 hipblasZtrsmStridedBatched (C++ function), 162  
 hipblasZtrsv (C++ function), 114  
 hipblasZtrsvBatched (C++ function), 115  
 hipblasZtrsvStridedBatched (C++ function), 116  
 hipblasZtrtri (C++ function), 164  
 hipblasZtrtriBatched (C++ function), 165  
 hipblasZtrtriStridedBatched (C++ function), 166