
hipBLAS Documentation

Release 2.1.0

Advanced Micro Devices, Inc.

Apr 16, 2024

CONTENTS

1	Building and Installing	3
1.1	Prerequisites	3
1.2	Installing pre-built packages	3
1.3	hipBLAS build	3
1.4	Dependencies	4
1.5	Manual build (all supported platforms)	5
2	Library Source Code Organization	7
2.1	The <i>library</i> directory	7
2.2	The <i>clients</i> directory	8
2.3	Infrastructure	8
3	hipBLAS API	9
3.1	hipBLAS Interface	9
3.2	Naming conventions	9
3.3	Notations	9
3.4	ILP64 Interface	10
3.5	HIPBLAS_V2 and Deprecations	10
3.6	bfloat 16 Datatype	10
3.7	Complex Datatypes	10
3.8	Atomic Operations	11
4	hipBLAS Types	13
4.1	Definitions	13
4.2	Enums	14
5	hipBLAS Functions	21
5.1	Level 1 BLAS	21
5.2	Level 2 BLAS	51
5.3	Level 3 BLAS	123
5.4	BLAS Extensions	175
5.5	SOLVER API	219
5.6	Auxiliary	237
6	Clients	243
6.1	hipblas-bench	243
6.2	hipblas-test	244
7	Deprecations by version	245
7.1	Announced in hipBLAS 0.49	245
7.2	Announced in hipBLAS 0.53	245

7.3	Announced in hipBLAS 1.0	245
7.4	Removed in hipBLAS 1.0	246
7.5	Announced in hipBLAS 2.0	247
7.6	Removed in hipBLAS 2.0	247
8	Contributing	249
8.1	Pull-request guidelines	249
8.2	Coding Guidelines:	249
8.3	Static Code Analysis	250
9	License	251
	Index	253

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.

BUILDING AND INSTALLING

1.1 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](#).

1.2 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 hipBLAS build

1.3.1 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 <code>install.sh</code> 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 hipBLAS in your local directory, you do not need the <code>-i</code> flag.

1.3.2 Build library dependencies + client dependencies + library + client

The client contains executables in the table below.

executable name	description
<code>hipblas-test</code>	runs Google Tests to test the library
<code>hipblas-bench</code>	executable to benchmark or test individual functions
<code>example-sscal</code>	example C code calling <code>hipblas_sscal</code> 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 <code>-d</code> flag only needs to be used once. For subsequent invocations of <code>install.sh</code> 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 <code>-i</code> flag. If you want to keep hipBLAS in your local directory, you do not need the <code>-i</code> 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 <code>-i</code> flag.

1.4 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.

However, for the test and benchmark clients' host reference functions you must manually download and install AMD's ILP64 version of the AOCL libraries, version 4.1 or 4.0, from <https://www.amd.com/en/developer/aocl.html>. The *aocl-linux-** packages include AOCL-BLAS and AOCL-LAPACK. If you download and install the full AOCL packages into their default locations then this reference LAPACK and BLAS should be found by the clients CMakeLists.txt. Note, if you only use the *install.sh -d* dependency script and change the default CMake option `LINK_BLIS=ON`, you may experience *hipblas-test* stress test failures due to 32-bit integer overflow on the host unless you exclude the stress tests via command line argument `-gtest_filter=-*stress*`.

1.5 Manual build (all supported platforms)

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

1.5.1 Dependencies For Building Library

1.5.2 Build Library Using Individual Commands

1.5.3 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.

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

2.1 The *library* directory

2.1.1 *library/include*

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

2.1.2 *library/src/amd_detail*

Implementation of hipBLAS interface compatible with rocBLAS APIs.

2.1.3 *library/src/nvidia_detail*

Implementation of hipBLAS interface compatible with cuBLAS-v2 APIs.

2.1.4 *library/src/include*

Internal include files for:

- Converting C++ exceptions to hipBLAS status.

2.2 The *clients* directory

2.2.1 *clients/gtest*

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

2.2.2 *clients/benchmarks*

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

2.2.3 *clients/include*

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

2.2.4 *clients/common*

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

2.2.5 *clients/samples*

Sample code for calling hipBLAS functions.

2.3 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.

HIPBLAS API

3.1 hipBLAS Interface

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:

3.1.1 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 );
```

3.2 Naming conventions

hipBLAS follows the following naming conventions:

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

3.3 Notations

hipBLAS function uses the following notations to denote precisions:

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

3.4 ILP64 Interface

The hipBLAS library Level-1 functions are also provided with ILP64 interfaces. With these interfaces all “int” arguments are replaced by the typename `int64_t`. These ILP64 function names all end with a suffix `_64`. The only output arguments that change are for the `xMAX` and `xMIN` for which the index is now `int64_t`. Function level documentation is not repeated for these API as they are identical in behavior to the LP64 versions, however functions which support this alternate API include the line: `This function supports the 64-bit integer interface.`

3.5 HIPBLAS_V2 and Deprecations

As of hipBLAS version 2.0.0, `hipblasDatatype_t` is deprecated, along with all functions which use this type. In a future release, all uses of `hipblasDatatype_t` will be replaced by `hipDataType`. See the `hipblasGemmEx` documentation for a small exception where `hipblasComputeType_t` replaces `hipblasDatatype_t` for the `computeType` parameter.

While `hipblasDatatype_t` is deprecated, users may use the compiler define or inline `#define HIPBLAS_V2` before including the header file `<hipblas.h>` to access the updated API. In a future release, this define will no longer be needed and deprecated functions will be removed, leaving the updated interface. Please see the documentation for the following functions to see the new interfaces: `hipblasTrsmEx`, `hipblasGemmEx`, `hipblasAxpvEx`, `hipblasDot(c)Ex`, `hipblasNrm2Ex`, `hipblasRotEx`, `hipblasScalEx`, and all batched and strided-batched variants.

3.6 bfloat 16 Datatype

hipBLAS defines a `hipblasBfloat16` datatype. This type is exposed as a struct simply containing 16 bits of data. There is also a C++ `hipblasBfloat16` class defined which gives slightly more functionality, including conversion to and from a 32-bit float datatype. This class can be used in C++11 or greater by defining `HIPBLAS_BFLOAT16_CLASS` before including the header file `hipblas.h`.

Furthermore, there is also an option to interpret the API as using the `hip_bfloat16` datatype. This is provided to avoid casting when using the `hip_bfloat16` datatype. To expose the API using `hip_bfloat16`, `HIPBLAS_USE_HIP_BFLOAT16` can be defined before including the header file `hipblas.h`. Note that the `hip_bfloat16` datatype is only supported on AMD platforms.

3.7 Complex Datatypes

hipBLAS defines `hipblasComplex` and `hipblasDoubleComplex` structs. These types contain x and y components and identical memory layout to `std::complex` for float and double precision.

For simplified usage with Hipified code, there is an option to interpret the API as using `hipFloatComplex` and `hipDoubleComplex` types (i.e. `typedef hipFloatComplex hipblasComplex`). This is provided for users to avoid casting when using the hip complex types in their code. As the memory layout is consistent across all three types, it is safe to cast arguments to API calls between the 3 types: `hipFloatComplex`, `std::complex<float>`, and `hipblasComplex`, as well as for the double precision variants. To expose the API as using the hip defined complex types, users can use either a compiler define or inline `#define ROCM_MATHLIBS_API_USE_HIP_COMPLEX` before including the header file `<hipblas.h>`. Thus, the API is compatible with both forms, but recompilation is required to avoid casting if switching to pass in the hip complex types.

Note that `hipblasComplex`, `hipblasDoubleComplex`, and use of `ROCM_MATHLIBS_API_USE_HIP_COMPLEX` are now deprecated. The API will provide interfaces using only `hipComplex` and `hipDoubleComplex` in the future.

3.8 Atomic Operations

Some functions in hipBLAS may use atomic operations to increase performance which may cause functions to not give bit-wise reproducible results. By default, the rocBLAS backend allows the use of atomics while the cuBLAS backend disallows the use of atomics. To set the desired behavior, users should call [*hipblasSetAtomicMode\(\)*](#). Please see the rocBLAS or cuBLAS documentation for more information regarding specifics of atomic operations in the backend library.

HIPBLAS TYPES

4.1 Definitions

4.1.1 hipblasHandle_t

typedef void ***hipblasHandle_t**

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

4.1.2 hipblasHalf

typedef uint16_t **hipblasHalf**

To specify the datatype to be unsigned short.

4.1.3 hipblasInt8

typedef int8_t **hipblasInt8**

To specify the datatype to be signed char.

4.1.4 hipblasStride

typedef int64_t **hipblasStride**

Stride between matrices or vectors in strided_batched functions.

4.1.5 hipblasBfloat16

struct **hipblasBfloat16**

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

4.1.6 hipblasComplex

struct **hipblasComplex**

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

4.1.7 hipblasDoubleComplex

struct **hipblasDoubleComplex**

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

4.2 Enums

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

4.2.1 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

4.2.2 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.

4.2.3 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.

4.2.4 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**

4.2.5 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.

4.2.6 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**

4.2.7 hipblasDatatype_t

enum **hipblasDatatype_t**

Indicates the precision of data used. hipblasDatatype_t is deprecated as of hipBLAS 2.0.0 and will be removed in a future release as generally replaced by hipDataType.

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

enumerator **HIPBLAS_DATATYPE_INVALID**
Invalid datatype value, do not use

4.2.8 hipblasComputeType_t

enum **hipblasComputeType_t**

The compute type to be used. Currently only used with GemmEx with the HIPBLAS_V2 interface. Note that support for compute types is largely dependent on backend.

Values:

enumerator **HIPBLAS_COMPUTE_16F**
compute will be at least 16-bit precision

enumerator **HIPBLAS_COMPUTE_16F_PEDANTIC**
compute will be exactly 16-bit precision

enumerator **HIPBLAS_COMPUTE_32F**
compute will be at least 32-bit precision

enumerator **HIPBLAS_COMPUTE_32F_PEDANTIC**
compute will be exactly 32-bit precision

enumerator **HIPBLAS_COMPUTE_32F_FAST_16F**
32-bit input can use 16-bit compute

enumerator **HIPBLAS_COMPUTE_32F_FAST_16BF**
32-bit input can is bf16 compute

enumerator **HIPBLAS_COMPUTE_32F_FAST_TF32**
32-bit input can use tensor cores w/ TF32 compute. Only supported with cuBLAS backend currently

enumerator **HIPBLAS_COMPUTE_64F**
compute will be at least 64-bit precision

enumerator **HIPBLAS_COMPUTE_64F_PEDANTIC**

compute will be exactly 64-bit precision

enumerator **HIPBLAS_COMPUTE_32I**

compute will be at least 32-bit integer precision

enumerator **HIPBLAS_COMPUTE_32I_PEDANTIC**

compute will be exactly 32-bit integer precision

4.2.9 hipblasGemmAlgo_t

enum **hipblasGemmAlgo_t**

Indicates if layer is active with bitmask.

Values:

enumerator **HIPBLAS_GEMM_DEFAULT**

enumerator rocblas_gemm_algo_standard

4.2.10 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. By default, the rocBLAS backend will allow atomic operations while the cuBLAS backend will disallow atomic operations. See backend documentation for more detail.

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.

HIPBLAS FUNCTIONS

5.1 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*

5.1.1 hipblasIXamax + 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.

The amax function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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<=0.

The amaxBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 $n \leq 0$, $\text{incx} \leq 0$.

The amaxStridedBatched function supports the 64-bit integer interface. Refer to section *ILP64 Interface*.

5.1.2 hipblasIXamin + 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.

The amin function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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_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 pointers to array of batchCount size for results. return is 0 if n, incx<=0.

The aminBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 $n \leq 0$, $\text{incx} \leq 0$.

The aminStridedBatched function supports the 64-bit integer interface. Refer to section *ILP64 Interface*.

5.1.3 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 $n \leq 0$.

The asum function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 $i = 1, \dots, \text{batchCount}$.

- 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 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 $n, \text{incx} \leq 0$.

The asumBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 $\text{stride_x} \geq n * \text{incx}$.
- **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 if $n, \text{incx} \leq 0$.

The asumStridedBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.1.4 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.

The axpy function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

hipblasStatus_t **hipblasHaxpyBatched**(*hipblasHandle_t* handle, int n, const *hipblasHalf* *alpha, const *hipblasHalf* *const x[], int incx, *hipblasHalf* *const y[], int incy, int batchSize)

hipblasStatus_t **hipblasSaxpyBatched**(*hipblasHandle_t* handle, int n, const float *alpha, const float *const x[], int incx, float *const y[], int incy, int batchSize)

hipblasStatus_t **hipblasDaxpyBatched**(*hipblasHandle_t* handle, int n, const double *alpha, const double *const x[], int incx, double *const y[], int incy, int batchSize)

hipblasStatus_t **hipblasCaxpyBatched**(*hipblasHandle_t* handle, int n, const *hipblasComplex* *alpha, const *hipblasComplex* *const x[], int incx, *hipblasComplex* *const y[], int incy, int batchSize)

hipblasStatus_t **hipblasZaxpyBatched**(*hipblasHandle_t* handle, int n, const *hipblasDoubleComplex* *alpha, const *hipblasDoubleComplex* *const x[], int incx, *hipblasDoubleComplex* *const y[], int incy, int batchSize)

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

The axpyBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 batchSize)

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 batchSize)

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

The axpyStridedBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.1.5 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.

The copy function supports the 64-bit integer interface. Refer to section *ILP64 Interface*.

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

The copyBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

hipblasStatus_t **hipblasCopyStridedBatched**(*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 $\text{stride_x} \geq n * \text{incx}$.
- **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 $\text{stride_y} \geq n * \text{incy}$. stridey should be non zero.
- **batchCount** – [in] [int] number of instances in the batch

The copyStridedBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.1.6 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.

The dot function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 batchSize, double *result)

hipblasStatus_t **hipblasCdotcBatched**(*hipblasHandle_t* handle, int n, const *hipblasComplex* *const x[], int incx, const *hipblasComplex* *const y[], int incy, int batchSize, *hipblasComplex* *result)

hipblasStatus_t **hipblasCdotuBatched**(*hipblasHandle_t* handle, int n, const *hipblasComplex* *const x[], int incx, const *hipblasComplex* *const y[], int incy, int batchSize, *hipblasComplex* *result)

hipblasStatus_t **hipblasZdotcBatched**(*hipblasHandle_t* handle, int n, const *hipblasDoubleComplex* *const x[], int incx, const *hipblasDoubleComplex* *const y[], int incy, int batchSize, *hipblasDoubleComplex* *result)

hipblasStatus_t **hipblasZdotuBatched**(*hipblasHandle_t* handle, int n, const *hipblasDoubleComplex* *const x[], int incx, const *hipblasDoubleComplex* *const y[], int incy, int batchSize, *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, ..., batchSize

- 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 $n \leq 0$.

The dotBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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

$$\text{result_i} = \mathbf{x_i} \cdot \mathbf{y_i};$$

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

$$\text{result_i} = \text{conjugate}(\mathbf{x_i}) \cdot \mathbf{y_i};$$

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 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 $n \leq 0$.

The dotStridedBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.1.7 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.

The nrm2 function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 $n \leq 0$, $incx \leq 0$.

The nrm2Batched function supports the 64-bit integer interface. Refer to section *ILP64 Interface*.

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 $n \leq 0$, $\text{incx} \leq 0$.

The nrm2StridedBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.1.8 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.

The rot function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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.

The rotBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 batchSize)

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 batchSize)

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 batchSize)

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 batchSize)

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 batchSize)

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 batchSize)

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.

The rotStridedBatched function supports the 64-bit integer interface. Refer to section *ILP64 Interface*.

5.1.9 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.

The rotg function supports the 64-bit integer interface. Refer to section *ILP64 Interface*.

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 $i = 1, \dots, \text{batchCount}$. 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 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).

The `rotgBatched` function supports the 64-bit integer interface. Refer to section *ILP64 Interface*.

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).

The rotgStridedBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.1.10 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 = (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.

The rotm function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 = (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 batched version of this function.
- **batchCount** – [in] [int] the number of x and y arrays, i.e. the number of batches.

The rotmBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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.

The rotmStridedBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.1.11 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 => $H = \begin{pmatrix} H11 & H12 & H21 & H22 \end{pmatrix}$ 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.

The rotmg function supports the 64-bit integer interface. Refer to section *ILP64 Interface*.

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 => $H = \begin{pmatrix} H11 & H12 & H21 & H22 \end{pmatrix}$ 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.
- **batchCount** – [in] [int] the number of instances in the batch.

The `rotmgBatched` function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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.
- **stridedx1** – [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. $\text{param}[0] = \text{flag}$ $\text{param}[1] = H11$ $\text{param}[2] = H21$ $\text{param}[3] = H12$ $\text{param}[4] = H22$ The flag parameter defines the form of H: $\text{flag} = -1 \Rightarrow H = \begin{pmatrix} H11 & H12 \\ H21 & H22 \end{pmatrix}$ $\text{flag} = 0 \Rightarrow H = \begin{pmatrix} 1.0 & H12 & H21 & 1.0 \end{pmatrix}$ $\text{flag} = 1 \Rightarrow H = \begin{pmatrix} H11 & 1.0 & -1.0 & H22 \end{pmatrix}$ $\text{flag} = -2 \Rightarrow H = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$ 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.

The `rotmgStridedBatched` function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.1.12 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`.

The `scal` function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 **hipblasCscalBatched**(*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.

The scalBatched function supports the 64-bit integer interface. Refer to section *ILP64 Interface*.

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 **hipblasCscalStridedBatched**(*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 := 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 .
- **stridedx** – [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 .

The scalStridedBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.1.13 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.

The swap function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

hipblasStatus_t **hipblasSswapBatched**(*hipblasHandle_t* handle, int n, float *const x[], int incx, float *const y[], int incy, int batchCount)

hipblasStatus_t **hipblasDswapBatched**(*hipblasHandle_t* handle, int n, double *const x[], int incx, double *const y[], int incy, int batchCount)

hipblasStatus_t **hipblasCswapBatched**(*hipblasHandle_t* handle, int n, *hipblasComplex* *const x[], int incx, *hipblasComplex* *const y[], int incy, int batchCount)

hipblasStatus_t **hipblasZswapBatched**(*hipblasHandle_t* handle, int n, *hipblasDoubleComplex* *const x[], int incx, *hipblasDoubleComplex* *const y[], int incy, int batchCount)

BLAS Level 1 API.

swapBatched 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 array of device pointers storing 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 .
- **incy** – [in] [int] specifies the increment for the elements of each y_i .
- **batchCount** – [in] [int] number of instances in the batch.

The swapBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

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 $stride_x \geq n * incx$.
- **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 $stride_y \geq n * incy$. $stridey$ should be non zero.
- **batchCount** – [in] [int] number of instances in the batch.

The swapStridedBatched function supports the 64-bit integer interface. Refer to section [ILP64 Interface](#).

5.2 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*

5.2.1 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 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 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 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 0 2 2 2 2
2 5 4 1 2 3 0 0 -&#8212;> 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 0
0 0 0 0 5 4 1 2 0 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 A. Must be $\geq (kl + ku + 1)$
- **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 batchSize)

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 batchSize)

BLAS Level 2 API.

gbmvBatched performs one of the 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. α and β 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 α .
- **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: $(m = n = 7; ku = 2, kl = 2)$ $\begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 3 & 3 & 4 \\ 1 & 2 & 3 & 0 & 0 & 0 & 2 \\ 2 & 2 & 2 & 2 & 2 & 5 & 4 \\ 1 & 2 & 3 & 0 & 0 & 0 & 0 \end{bmatrix}$ 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 β .
- **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,   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 banded matrix with kl sub-diagonals and ku super-diagonals, 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 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 to first banded matrix (A_1). 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 3 4 1 2 3 0 0 0 0 2 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 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 A. Must be >= (kl + ku + 1)
- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A_i) and the next one (A_i+1)

- **x** – [in] device pointer to 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})
- **beta** – [in] device pointer or host pointer to scalar beta.
- **y** – [inout] device pointer to 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] specifies the number of instances in the batch.

5.2.2 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, ..., 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.
- **trans** – [in] [hipblasOperation_t] indicates whether matrices A_i are transposed (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. α and β are scalars, 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.
- **transA** – [in] [hipblasOperation_t] indicates whether matrices A_i are transposed (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 $\geq n * incx$, otherwise stridex $\geq m * incx$.
- **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 $\geq m * incy$, otherwise stridey $\geq n * incy$. stridey should be non zero.
- **batchCount** – [in] [int] number of instances in the batch

5.2.3 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 batchSize)

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 batchSize)

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 batchSize)

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 batchSize)

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 batchSize)


```
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_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, ..., 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 array of device pointers storing each vector x_i.
- **incx** – [in] [int] specifies the increment for the elements of each vector 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 vector y_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 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. α 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 α .
- **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

5.2.4 hipblasXhbm + Batched, StridedBatched

hipblasStatus_t **hipblasChbm**(*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 **hipblasZhbm**(*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.

hbm 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 >= 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)
 (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 batchCount)

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 batchCount)

BLAS Level 2 API.

hbmVBatched 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 ≥ 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 (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 $\geq \max(1, n)$
- **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 α and β 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 ≥ 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 .
- **stridx** – [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.

5.2.5 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 >= 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, 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 >= 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, 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 >= 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.

5.2.6 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 α 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 α .
- **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.

5.2.7 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 α 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 α .
- **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 $\text{uplo} == \text{HIPBLAS_FILL_MODE_UPPER}$: 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 $\text{uplo} == \text{HIPBLAS_FILL_MODE_LOWER}$: 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 $\max(lda, 1)$.
- **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 stridx, 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 α 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 α .
- **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.

5.2.8 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 ≥ 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) –—> [(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) –—> [(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) –> [(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.
- **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.

5.2.9 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 batchCount)

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) –—> [(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.

- **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.

5.2.10 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) –—> [(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.

5.2.11 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 *const 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 *const 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, ..., 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. α and β 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 α
- **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 β
- **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

5.2.12 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 *const 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 *const 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, ..., 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. α and β 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 α
- **AP** – [in] Device pointer to the first matrix A_1 on the GPU
- **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
- **stridx** – [in] [hipblasStride] stride from the start of one vector (x_i) and the next one (x_{i+1}). There are no restrictions placed on stridx, however the user should take care to ensure that stridx is of appropriate size. This typically means $\text{stridx} \geq n * \text{incx}$. stridx should be non zero.
- **beta** – [in] device pointer or host pointer to scalar β
- **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

5.2.13 hipblasXsopr + 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 α 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 α .
- **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 symmetric 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 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 –—> [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 –—> [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, ..., 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 –—> [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 –—> [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.

5.2.14 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 **hipblasDsspr2**(*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 $i = 1, \dots, \text{batchCount}$.

- 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_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 symmetric 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 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 –—> [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: $AP(0) = A(0,0)$ $AP(1) = A(1,0)$ $AP(n) = A(2,1)$, 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_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 : 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_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 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 –—> [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: $AP(0) = A(0,0)$ $AP(1) = A(1,0)$ $AP(n) = A(2,1)$, 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_i) and the next (A_{i+1})
- **batchCount** – [in] [int] number of instances in the batch.

5.2.15 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 *const y[], int incy, int batchCount)

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 *const y[], int incy, int batchCount)

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* *const y[], int incy, int batchCount)

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* *const 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. α and β 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 α
- **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 β
- **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, ..., batchCount. A_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 A₁ 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 x₁ 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

5.2.16 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

5.2.17 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 batchSize)

hipblasStatus_t **hipblasDsyr2Batched**(*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 batchSize)

```
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_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 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 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 hipblasCsyr2StridedBatched(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 hipblasZsyr2StridedBatched(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_i) and the next one (A_i+1)
- **batchCount** – [in] [int] number of instances in the batch

5.2.18 hipblasXtbmv + Batched, StridedBatched

hipblasStatus_t **hipblasStbmv**(*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 **hipblasDtbmv**(*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 **hipblasCtbmv**(*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 **hipblasZtbmv**(*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.

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 n by n 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.
- n** – [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; n = 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 represnted 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; n = 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 n, 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 n, 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 n, 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 n, 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

$x_i := A_i * x_i$	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 n 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.
- **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.

- **n** – [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 $k > 0$ && $k < lda$.
- **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; $n = 5$; $k = 2$) 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; $n = 5$; $k = 2$) 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 $lda > k$.
- **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 n, 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 n, 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 n, 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 n, int k, const *hipblasDoubleComplex* *AP, int lda, *hipblasStride* strideA, *hipblasDoubleComplex* *x, int incx, *hipblasStride* stridex, 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 n 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.
- **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.
- **n** – [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; $n = 5$; $k = 2$) 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; $n = 5$; $k = 2$) 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 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.

5.2.19 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($\text{uplo} == \text{HIPBLAS_FILL_MODE_UPPER}$) k specifies the number of super-diagonals of each A_i . if($\text{uplo} == \text{HIPBLAS_FILL_MODE_LOWER}$) 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 . $\text{lda} \geq (k + 1)$.
- **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.

5.2.20 hipblasXtpmv + Batched, StridedBatched

hipblasStatus_t **hipblasStpmv**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const float *AP, float *x, int incx)

hipblasStatus_t **hipblasDtpmv**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const double *AP, double *x, int incx)

hipblasStatus_t **hipblasCtpmv**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const *hipblasComplex* *AP, *hipblasComplex* *x, int incx)

hipblasStatus_t **hipblasZtpmv**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, 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.
- **n** – [in] [int] `n` specifies the number of rows of `A`. `n` \geq 0.
- **AP** – [in] device pointer storing matrix `A`, of dimension at least $(n * (n + 1) / 2)$. 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 n, 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 n, 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 n, 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 n, 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.
- **n** – [in] [int] n specifies the number of rows of matrices A_i . $n \geq 0$.
- **AP** – [in] device pointer storing pointer of matrices A_i , of dimension (lda, n)
- **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 n, 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 n, 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 n, 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 n, 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

$$x_i = A_i * x_i \text{ or } x_i = A_i^T * x_i, \quad 0 \leq i < \text{batchCount}$$

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.
- **n** – [in] [int] n specifies the number of rows of matrices A_i . $n \geq 0$.
- **AP** – [in] device pointer of the matrix A_0 , of dimension (lda, n)
- **strideA** – [in] [*hipblasStride*] stride from the start of one A_i matrix to the next A_{i+1}
- **x** – [in] device pointer storing the vector x_0 .
- **incx** – [in] [int] specifies the increment for the elements of one vector x .
- **stridex** – [in] [*hipblasStride*] stride from the start of one x_i vector to the next x_{i+1}
- **batchCount** – [in] [int] The number of batched matrices/vectors.

5.2.21 hipblasXtpsv + Batched, StridedBatched

hipblasStatus_t **hipblasStpsv**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const float *AP, float *x, int incx)

hipblasStatus_t **hipblasDtpsv**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const double *AP, double *x, int incx)

hipblasStatus_t **hipblasCtpsv**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const *hipblasComplex* *AP, *hipblasComplex* *x, int incx)

hipblasStatus_t **hipblasZtpsv**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, 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.
- **n** – [in] [int] n specifies the number of rows of b. $n \geq 0$.
- **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 n, const float *const AP[], float *const x[], int incx, int batchSize)


```
hipblasStatus_t hipblasDtpsvBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, hipblasOperation_t
    transA, hipblasDiagType_t diag, int n, const double *const AP[], double
    *const x[], int incx, int batchCount)
```

```
hipblasStatus_t hipblasCtpsvBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, hipblasOperation_t
    transA, hipblasDiagType_t diag, int n, const hipblasComplex *const AP[],
    hipblasComplex *const x[], int incx, int batchCount)
```

```
hipblasStatus_t hipblasZtpsvBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, hipblasOperation_t
    transA, hipblasDiagType_t diag, int n, const hipblasDoubleComplex
    *const AP[], hipblasDoubleComplex *const x[], int incx, int batchCount)
```

BLAS Level 2 API.

tpsvBatched 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.
- **n** – [in] [int] n specifies the number of rows of each b_i . $n \geq 0$.
- **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 n, 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 n, 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 n, 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 n, 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.
- **n** – [in] [int] n specifies the number of rows of each b_i . $n \geq 0$.
- **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 .
- **stridex** – [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.

5.2.22 hipblasXtrmv + Batched, StridedBatched

hipblasStatus_t **hipblasStrmv**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, 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 n, 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 n, 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 n, const *hipblasDoubleComplex* *AP, int lda, *hipblasDoubleComplex* *x, int incx)

BLAS Level 2 API.

trmv performs one of the matrix-vector operations

$$\mathbf{x} = \mathbf{A} * \mathbf{x} \text{ or } \mathbf{x} = \mathbf{A}^{**T} * \mathbf{x},$$

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.
- **n** – [in] [int] n specifies the number of rows of A. $n \geq 0$.
- **AP** – [in] device pointer storing matrix A, of dimension (lda, n)
- **lda** – [in] [int] specifies the leading dimension of A. $lda = \max(1, n)$.
- **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 n, const float *const AP[], int lda, float *const x[], int incx, int batchSize)

hipblasStatus_t **hipblasDtrmvBatched**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const double *const AP[], int lda, double *const x[], int incx, int batchSize)

```
hipblasStatus_t hipblasCtrmvBatched(hipblasHandle_t handle, hipblasFillMode_t uplo, hipblasOperation_t
    transA, hipblasDiagType_t diag, int n, 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 n, 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

$$\mathbf{x}_i = \mathbf{A}_i * \mathbf{x}_i \text{ or } \mathbf{x}_i = \mathbf{A}^{**T} * \mathbf{x}_i, \quad 0 \leq i < \text{batchCount}$$

where \mathbf{x}_i is an n element vector and \mathbf{A}_i is an n by n (unit, or non-unit, upper or lower triangular matrix)

The vectors \mathbf{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: \mathbf{A}_i is an upper triangular matrix. HIPBLAS_FILL_MODE_LOWER: \mathbf{A}_i is a lower triangular matrix.
- **transA** – [in] [hipblasOperation_t]
- **diag** – [in] [hipblasDiagType_t] HIPBLAS_DIAG_UNIT: \mathbf{A}_i is assumed to be unit triangular. HIPBLAS_DIAG_NON_UNIT: \mathbf{A}_i is not assumed to be unit triangular.
- **n** – [in] [int] n specifies the number of rows of matrices \mathbf{A}_i . $n \geq 0$.
- **AP** – [in] device pointer storing pointer of matrices \mathbf{A}_i , of dimension (lda, n)
- **lda** – [in] [int] specifies the leading dimension of \mathbf{A}_i . $\text{lda} \geq \max(1, n)$.
- **x** – [in] device pointer storing vectors \mathbf{x}_i .
- **incx** – [in] [int] specifies the increment for the elements of vectors \mathbf{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 n, 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 n, 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 n, 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 n, 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.
- **n** – [in] [int] n specifies the number of rows of matrices A_i . $n \geq 0$.
- **AP** – [in] device pointer of the matrix A_0 , of dimension (lda, n)
- **lda** – [in] [int] specifies the leading dimension of A_i . $lda \geq \max(1, n)$.
- **strideA** – [in] [*hipblasStride*] stride from the start of one A_i matrix to the next A_{i+1}
- **x** – [in] device pointer storing the vector x_0 .
- **incx** – [in] [int] specifies the increment for the elements of one vector x .
- **stridex** – [in] [*hipblasStride*] stride from the start of one x_i vector to the next x_{i+1}
- **batchCount** – [in] [int] The number of batched matrices/vectors.

5.2.23 hipblasXtrsv + Batched, StridedBatched

```
hipblasStatus_t hipblasStrsv(hipblasHandle_t handle, hipblasFillMode_t uplo, hipblasOperation_t transA,
                              hipblasDiagType_t diag, int n, 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 n, 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 n, 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 n, 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.
- **n** – [in] [int] n specifies the number of rows of b. $n \geq 0$.
- **AP** – [in] device pointer storing matrix A, of dimension (lda, n)
- **lda** – [in] [int] specifies the leading dimension of A. $lda = \max(1, n)$.
- **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 n, const float *const AP[], int lda, float *const x[], int incx, int batchSize)

hipblasStatus_t **hipblasDtrsvBatched**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const double *const AP[], int lda, double *const x[], int incx, int batchSize)

hipblasStatus_t **hipblasCtrsvBatched**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const *hipblasComplex* *const AP[], int lda, *hipblasComplex* *const x[], int incx, int batchSize)

hipblasStatus_t **hipblasZtrsvBatched**(*hipblasHandle_t* handle, *hipblasFillMode_t* uplo, *hipblasOperation_t* transA, *hipblasDiagType_t* diag, int n, const *hipblasDoubleComplex* *const AP[], int lda, *hipblasDoubleComplex* *const x[], int incx, int batchSize)

BLAS Level 2 API.

trsvBatched solves

$$A_i * x_i = b_i \text{ or } A_i^{**T} x_i = b_i,$$

where (A_i, x_i, b_i) is the i -th instance of the batch. x_i and b_i are vectors and A_i is an n by n 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.
- **n** – [in] [int] n specifies the number of rows of b . $n \geq 0$.
- **AP** – [in] device array of device pointers storing each matrix A_i .
- **lda** – [in] [int] specifies the leading dimension of each A_i . $lda = \max(1, n)$
- **x** – [in] device array of device pointers storing each vector x_i .
- **incx** – [in] [int] specifies the increment for the elements of 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 n, 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 n, 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 n, 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 n, 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_i, x_i, b_i) is the i -th instance of the batch. x_i and b_i are vectors and A_i is an n by n triangular matrix, for $i = 1, \dots, \text{batchCount}$.

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.
- **n** – [in] [int] n specifies the number of rows of each b_i . $n \geq 0$.
- **AP** – [in] device pointer to the first matrix (A_1) in the batch, of dimension (lda, n)
- **strideA** – [in] [hipblasStride] stride from the start of one A_i matrix to the next $A_{(i+1)}$
- **lda** – [in] [int] specifies the leading dimension of each A_i . $\text{lda} = \max(1, n)$.
- **x** – [inout] device pointer to the first vector (x_1) in the batch.
- **stridx** – [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

5.3 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*

5.3.1 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

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

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 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 op(A)
- **transB** – [in] [hipblasOperation_t] specifies the form of op(B)
- **m** – [in] [int] number or rows of matrices op(A) and C
- **n** – [in] [int] number of columns of matrices op(B) and C
- **k** – [in] [int] number of columns of matrix op(A) and number of rows of matrix 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.

gemmBatched 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 $\text{op}(X) = X^{**}H$, α and β 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 α .
- **AP** – [in] device array of device pointers storing each matrix A_i .
- **lda** – [in] [int] specifies the leading dimension of each A_i .
- **BP** – [in] device array of device pointers storing each matrix B_i .
- **ldb** – [in] [int] specifies the leading dimension of each B_i .
- **beta** – [in] device pointer or host pointer specifying the scalar β .
- **CP** – [inout] device array of device pointers storing 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

hipblasStatus_t **hipblasHgemmStridedBatched**(*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 **hipblasSgemmStridedBatched**(*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 **hipblasDgemmStridedBatched**(*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.

gemmStridedBatched performs one of the strided batched matrix-matrix operations

```
C_i = alpha*op( A_i )*op( B_i ) + beta*C_i, for i = 1, ..., batchCount.
```

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 batchCount strided_batched matrix, 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 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 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 C₁.
- **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

5.3.2 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: op(A) = A^H HIPBLAS_OP_N: op(A) = A
- **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. 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: $\text{op}(A) = A^H$ HIPBLAS_OP_N: $\text{op}(A) = A$
- **n** – [in] [int] n specifies the number of rows and columns of C_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, $\text{lda} \geq \max(1, n)$, 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] 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. $\text{ldc} \geq \max(1, n)$.
- **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_t strideA, const float
                                           *beta, hipblasComplex *CP, int ldc, hipblasStride_t 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_t strideA, const
                                           double *beta, hipblasDoubleComplex *CP, int ldc, hipblasStride_t
                                           strideC, int batchCount)
```

BLAS Level 3 API.

herkStridedBatched performs a batch of the matrix-matrix operations for a Hermitian rank-k update

$$C_i := \alpha * \text{op}(A_i) * \text{op}(A_i)^H + \beta * C_i$$

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: $\text{op}(A) = A^H$ HIPBLAS_OP_N: $\text{op}(A) = A$
- **n** – [in] [int] n specifies the number of rows and columns of C_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, $\text{lda} \geq \max(1, n)$, otherwise $\text{lda} \geq \max(1, k)$.
- **strideA** – [in] [hipblasStride_t] 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. $\text{ldc} \geq \max(1, n)$.

- **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.

5.3.3 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, $ldb \geq \max(1, n)$, 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] 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. $ldc \geq \max(1, n)$.

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
 ↪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. $n \geq 0$.
- **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 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 >= 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, 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: $\text{op}(A_i) = A_i^H$, $\text{op}(B_i) = B_i^H$
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_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.

5.3.4 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, $\text{op}(A)$ and $\text{op}(B)$ are n by k matrices, and C is a $n \times n$ Hermitian matrix stored as either upper or lower.

$\text{op}(A) = A$, $\text{op}(B) = B$, and A and B are n by k if $\text{trans} == \text{HIPBLAS_OP_N}$
 $\text{op}(A) = A^H$, $\text{op}(B) = B^H$, and A and B are k by n if $\text{trans} == \text{HIPBLAS_OP_C}$

- 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 $\text{trans} = \text{HIPBLAS_OP_N}$, otherwise (lda, n) only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if $\text{trans} = \text{HIPBLAS_OP_N}$, $\text{lda} \geq \max(1, n)$, otherwise $\text{lda} \geq \max(1, k)$.
- **BP** – [in] pointer storing matrix B on the GPU. Martrix dimension is (ldb, k) when if $\text{trans} = \text{HIPBLAS_OP_N}$, otherwise (ldb, n) only the upper/lower triangular part is accessed.
- **ldb** – [in] [int] ldb specifies the first dimension of B. if $\text{trans} = \text{HIPBLAS_OP_N}$, $\text{ldb} \geq \max(1, n)$, 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 batchCount)

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 batchCount)

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 $n \times n$ 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: $\text{op}(A) = A^H$ HIPBLAS_OP_N: $\text{op}(A) = A$
- **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 $\text{op}(A)$. $k \geq 0$.
- **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 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 $\text{lda} \geq \max(1, k)$.
- **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_i . 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_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 . $\text{ldc} \geq \max(1, n)$.
- **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_t strideA, const
                                             hipblasDoubleComplex *BP, int ldb, hipblasStride_t strideB,
                                             const double *beta, hipblasDoubleComplex *CP, int ldc,
                                             hipblasStride_t 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, 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.

```
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_i) = A_i^H, op(B_i) = B_i^H 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 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, lda >= max(1, n), otherwise lda >= 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, ldb >= max(1, n), otherwise ldb >= 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.

5.3.5 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 batchSize)

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 batchSize)

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 batchSize)

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 batchSize)

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 ≥ 0.
- **n** – [in] [int] n specifies the number of columns of B_i and C_i. n ≥ 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 α and β 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] α specifies the scalar α . When α 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] β specifies the scalar β . When β 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.

5.3.6 hipblasXsyrk + Batched, StridedBatched

hipblasStatus_t **hipblasSsyrk**(*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 **hipblasDsyrk**(*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 **hipblasCsyrk**(*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 **hipblasZsyrk**(*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.

syrk 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

$$C_i := \alpha \cdot \text{op}(A_i) \cdot \text{op}(A_i)^T + \beta \cdot C_i$$

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 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.
- **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 **hipblasSsyrcStridedBatched**(*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 **hipblasDsyrcStridedBatched**(*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 **hipblasCsyrcStridedBatched**(*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 **hipblasZsyrcStridedBatched**(*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.

syrcStridedBatched performs a batch of the matrix-matrix operations for a symmetric rank-k update

$C_i := \alpha * op(A_i) * op(A_i)^T + \beta * C_i$

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.

5.3.7 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

$$C := \alpha * (op(A) * op(B)^T + op(B) * op(A)^T) + \beta * C$$

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.

$\text{op}(A) = A$, $\text{op}(B) = B$, and A and B are n by k if $\text{trans} == \text{HIPBLAS_OP_N}$
 $\text{op}(A) = A^T$, $\text{op}(B) = B^T$, and A and B are k by n if $\text{trans} == \text{HIPBLAS_OP_T}$

- 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: $\text{op}(A) = A^T$, $\text{op}(B) = B^T$ 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 $\text{op}(A)$ and $\text{op}(B)$. $k \geq 0$.
- **alpha** – [in] α specifies the scalar α . When α 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 $\text{trans} = \text{HIPBLAS_OP_N}$, otherwise (lda, n) only the upper/lower triangular part is accessed.
- **lda** – [in] [int] lda specifies the first dimension of A. if $\text{trans} = \text{HIPBLAS_OP_N}$, $\text{lda} \geq \max(1, n)$, otherwise $\text{lda} \geq \max(1, k)$.
- **BP** – [in] pointer storing matrix B on the GPU. Martrix dimension is (ldb, k) when if $\text{trans} = \text{HIPBLAS_OP_N}$, otherwise (ldb, n) only the upper/lower triangular part is accessed.
- **ldb** – [in] [int] ldb specifies the first dimension of B. if $\text{trans} = \text{HIPBLAS_OP_N}$, $\text{ldb} \geq \max(1, n)$, otherwise $\text{ldb} \geq \max(1, k)$.
- **beta** – [in] β specifies the scalar β . When β 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 **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 batchSize)

hipblasStatus_t **hipblasDsyr2kBatched**(*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 batchSize)

hipblasStatus_t **hipblasCsyr2kBatched**(*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 batchSize)

```
hipblasStatus_t hipblasZsyr2kBatched(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.

syr2kBatched 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_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 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
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 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, lda >= max(1, n), otherwise lda >= 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, ldb >= max(1, n), otherwise ldb >= 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. ldc >= 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.

5.3.8 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 hipblasZsyrkxBatched(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.

syrkxBatched 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 **hipblasDsyrkxStridedBatched**(*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 **hipblasCsyrkxStridedBatched**(*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 **hipblasZsyrkxStridedBatched**(*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

$C_i := \alpha * op(A_i) * op(B_i)^T + \beta * C_i$

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). $k \geq 0$.
- **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, $lda \geq \max(1, n)$, 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)
- **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, $ldb \geq \max(1, n)$, otherwise $ldb \geq \max(1, k)$.
- **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. $ldc \geq \max(1, n)$.
- **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.

5.3.9 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
$\text{op}(X) = X^{**H}$,	

α and β 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 α .
- **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 β .
- **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

$$C_i = \alpha * \text{op}(A_i) + \beta * \text{op}(B_i) \quad \text{for } i = 0, 1, \dots, \text{batchCount} - 1$$

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

$$\begin{aligned} \text{op}(X) &= X && \text{or} \\ \text{op}(X) &= X^{**T} \end{aligned}$$

- 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*op( A_i ) + beta*op( B_i )  for i = 0, 1, ... 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

```
op( X ) = X      or
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 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 pointer to the first matrix A_0 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.
- **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 transB == HIPBLAS_OP_N and is n when transB == 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.

5.3.10 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. $\text{ldc} \geq \max(1, m)$

hipblasStatus_t **hipblasChemvBatched**(*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 batchCount)

hipblasStatus_t **hipblasZhemvBatched**(*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 batchCount)

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, $\text{lda} \geq \max(1, m)$, otherwise $\text{lda} \geq \max(1, n)$.
- **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 . $\text{ldb} \geq \max(1, m)$

- **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 **hipblasZhemmmStridedBatched**(*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.

hemmmStridedBatched 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 >= max(1, m), otherwise lda >= 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 B₁ of dimension (ldb, n) on the GPU
- **ldb** – [in] [int] ldb specifies the first dimension of B_i. if side = HIPBLAS_OP_N, ldb >= max(1, m), otherwise ldb >= 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 C₁ of dimension (ldc, n) on the GPU.
- **ldc** – [in] [int] ldc specifies the first dimension of C. ldc >= 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

5.3.11 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 *A, int lda, const float *B, int ldb, float *C, int ldc)

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 *A, int lda, const double *B, int ldb, double *C, int ldc)

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* *A, int lda, const *hipblasComplex* *B, int ldb, *hipblasComplex* *C, int ldc)

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* *A, int lda, const *hipblasDoubleComplex* *B, int ldb, *hipblasDoubleComplex* *C, int ldc)

BLAS Level 3 API.

trmm performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * B$, or $C := \alpha * B * \text{op}(A)$

where alpha is a scalar, B and C are an m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

op(A) = A **or** op(A) = A^T **or** op(A) = A^H.

Note that trmm can provide in-place functionality by passing in the same address for both matrices B and C and by setting ldb equal to ldc.

- 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 `op(A)` multiplies `B` from the left or right as follows: `HIPBLAS_SIDE_LEFT`: $C := \alpha * op(A) * B$. `HIPBLAS_SIDE_RIGHT`: $C := \alpha * B * op(A)$.
- **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 `op(A)` to be used in the matrix multiplication as follows: `HIPBLAS_OP_N`: $op(A) = A$. `HIPBLAS_OP_T`: $op(A) = A^T$. `HIPBLAS_OP_C`: $op(A) = A^H$.
- **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` 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` is not referenced and `B` need not be set before entry.
- **A** – [in] Device pointer to matrix `A` on the GPU. `A` has 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`, $lda \geq \max(1, n)$.
- **B** – [inout] Device pointer to the matrix `B` of dimension (ldb, n) on the GPU.
- **ldb** – [in] [int] `ldb` specifies the first dimension of `B`. $ldb \geq \max(1, m)$.
- **C** – [in] Device pointer to the matrix `C` of dimension (ldc, n) on the GPU. Users can pass in the same matrix `B` to parameter `C` to achieve in-place functionality of `trmm`.
- **ldc** – [in] [int] `ldc` specifies the first dimension of `C`. $ldc \geq \max(1, m)$.

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 A[], int lda, const float *const B[], int ldb, float *const C[], int ldc, 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 A[], int lda, const double *const B[],
                                     int ldb, double *const C[], int ldc, 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 A[], int lda, const
                                     hipblasComplex *const B[], int ldb, hipblasComplex *const C[], int ldc, 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 A[],
                                     int lda, const hipblasDoubleComplex *const B[], int ldb,
                                     hipblasDoubleComplex *const C[], int ldc, int batchCount)
```

BLAS Level 3 API.

trmmBatched performs one of the batched matrix-matrix operations

$C_i := \alpha * \text{op}(A_i) * B_i$, or $C_i := \alpha * B_i * \text{op}(A_i)$ for $i = 0, 1, \dots, \text{batchCount} - 1$

where α is a scalar, B_i and C_i are m by n matrices, A_i is a unit, or non-unit, upper or lower triangular matrix and $\text{op}(A_i)$ is one of

$\text{op}(A_i) = A_i$ **or** $\text{op}(A_i) = A_i^T$ **or** $\text{op}(A_i) = A_i^H$.

Note that trmmBatched can provide in-place functionality by passing in the same address for both matrices B and C and by setting ldb equal to ldc .

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

When $\text{uplo} == \text{HIPBLAS_FILL_MODE_UPPER}$ 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 $\text{uplo} == \text{HIPBLAS_FILL_MODE_LOWER}$ 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 $\text{diag} == \text{HIPBLAS_DIAG_UNIT}$ 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 : $B_i := \alpha * \text{op}(A_i) * B_i$. $\text{HIPBLAS_SIDE_RIGHT}$: $B_i := \alpha * B_i * \text{op}(A_i)$.
- **uplo** – [in] [hipblasFillMode_t] Specifies whether the matrix A is an upper or lower triangular matrix as follows: $\text{HIPBLAS_FILL_MODE_UPPER}$: A is an upper triangular matrix. $\text{HIPBLAS_FILL_MODE_LOWER}$: 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 and C_i. m >= 0.
- **n** – [in] [int] n specifies the number of columns of B_i and C_i. n >= 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.
- **A** – [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 >= max(1, m), if side == HIPBLAS_SIDE_RIGHT, lda >= max(1, n).
- **B** – [inout] device array of device pointers storing each matrix B_i of dimension (ldb, n) on the GPU.
- **ldb** – [in] [int] ldb specifies the first dimension of B_i. ldb >= max(1, m).
- **C** – [in] device array of device pointers storing each matrix C_i of dimension (ldc, n) on the GPU. Users can pass in the same matrices B to parameter C to achieve in-place functionality of trmmBatched.
- **ldc** – [in] [int] ldc specifies the first dimension of C_i. ldc >= 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 *A, int lda, hipblasStride_t strideA, const float *B, int ldb,
hipblasStride_t strideB, float *C, int ldc, hipblasStride_t strideC, 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 *A, int lda, hipblasStride_t strideA, const double *B, int
ldb, hipblasStride_t strideB, double *C, int ldc, hipblasStride_t
strideC, 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 *A, int lda, hipblasStride_t strideA,
const hipblasComplex *B, int ldb, hipblasStride_t strideB,
hipblasComplex *C, int ldc, hipblasStride_t strideC, 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
*A, int lda, hipblasStride_t strideA, const hipblasDoubleComplex
*B, int ldb, hipblasStride_t strideB, hipblasDoubleComplex *C, int
ldc, hipblasStride_t strideC, int batchCount)
```


BLAS Level 3 API.

trmmStridedBatched performs one of the strided_batched matrix-matrix operations

$C_i := \alpha * \text{op}(A_i) * B_i$, or $C_i := \alpha * B_i * \text{op}(A_i)$ for $i = 0, 1, \dots, \text{batchCount} - 1$

where α is a scalar, B_i and C_i are m by n matrices, A_i is a unit, or non-unit, upper or lower triangular matrix and $\text{op}(A_i)$ is one of

$\text{op}(A_i) = A_i$ **or** $\text{op}(A_i) = A_i^T$ **or** $\text{op}(A_i) = A_i^H$.

Note that trmmStridedBatched can provide in-place functionality by passing in the same address for both matrices B and C and by setting ldb equal to ldc .

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

When $\text{uplo} == \text{HIPBLAS_FILL_MODE_UPPER}$ 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 $\text{uplo} == \text{HIPBLAS_FILL_MODE_LOWER}$ 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 $\text{diag} == \text{HIPBLAS_DIAG_UNIT}$ 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: $C_i := \alpha * \text{op}(A_i) * B_i$. HIPBLAS_SIDE_RIGHT: $C_i := \alpha * B_i * \text{op}(A_i)$.
- **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 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] α specifies the scalar α . When α is zero then A_i is not referenced and B_i need not be set before entry.
- **A** – [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 $\text{side} == \text{HIPBLAS_SIDE_LEFT}$, $\text{lda} \geq \max(1, m)$, if $\text{side} == \text{HIPBLAS_SIDE_RIGHT}$, $\text{lda} \geq \max(1, n)$.

- **strideA** – [in] [hipblasStride] stride from the start of one matrix (A_i) and the next one (A_i+1)
- **B** – [inout] Device pointer to the first matrix B₀ on the GPU. Each B_i is of dimension (ldb, n)
- **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)
- **C** – [in] Device pointer to the first matrix C₀ on the GPU. Each C_i is of dimension (ldc, n).
- **ldc** – [in] [int] ldc specifies the first dimension of C_i. $ldc \geq \max(1, m)$.
- **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.

5.3.12 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, const 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, const 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, const *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, const *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_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: 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, $\text{lda} \geq \max(1, m)$, if side = HIPBLAS_SIDE_RIGHT, $\text{lda} \geq \max(1, n)$.
- **BP** – [inout] device pointer storing matrix B.
- **ldb** – [in] [int] ldb specifies the first dimension of B. $\text{ldb} \geq \max(1, m)$.

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, const float *const AP[], int lda, float *const BP[], int ldb, int batchSize)

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, const double *const AP[], int lda, double *const BP[], int ldb, int batchSize)

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, const *hipblasComplex* *const AP[], int lda, *hipblasComplex* *const BP[], int ldb, int batchSize)

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, const *hipblasDoubleComplex* *const AP[], int lda, *hipblasDoubleComplex* *const 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_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: 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, const 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, const 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, const *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, const *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 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, ..., 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, $\text{lda} \geq \max(1, m)$, if side = HIPBLAS_SIDE_RIGHT, $\text{lda} \geq \max(1, n)$.
- **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 . $\text{ldb} \geq \max(1, m)$.
- **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.

5.3.13 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 batchCount)

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 batchCount)

hipblasStatus_t **hipblasCtrtriBatched**(*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 **hipblasZtrtriBatched**(*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, ..., 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_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 pointer pointing to address of first matrix A_1 .
- **lda** – [in] [int] specifies the leading dimension of each A.
- **strideA** – [in] [*hipblasStride*] “batch stride a”: stride from the start of one A_i matrix to the next A_{i+1} .

- **invA** – [out] device pointer storing the inverses 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.
- **stride_invA** – [in] [hipblasStride] “batch stride invA”: stride from the start of one invA_i matrix to the next invA_(i + 1).
- **batchCount** – [in] [int] numbers of matrices in the batch

5.3.14 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 **hipblasZdgmm**(*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

$C = A * \text{diag}(x) \text{ if } \text{side} == \text{HIPBLAS_SIDE_RIGHT}$ $C = \text{diag}(x) * A \text{ if } \text{side} == \text{HIPBLAS_SIDE_LEFT}$
--

where C and A are m by n dimensional matrices. $\text{diag}(x)$ is a diagonal matrix and x is vector of dimension n if $\text{side} == \text{HIPBLAS_SIDE_RIGHT}$ and dimension m if $\text{side} == \text{HIPBLAS_SIDE_LEFT}$.

- 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 $\text{diag}(x)$
- **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 batchSize)

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 batchSize)

hipblasStatus_t **hipblasCdgmBatched**(*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 batchSize)

hipblasStatus_t **hipblasZdgmBatched**(*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 batchSize)

BLAS Level 3 API.

dgmmBatched performs one of the batched matrix-matrix operations

```
C_i = A_i * diag(x_i) for i = 0, 1, ... batchSize-1 if side == HIPBLAS_SIDE_RIGHT
C_i = diag(x_i) * A_i for i = 0, 1, ... batchSize-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 **hipblasSdgmmStridedBatched**(*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_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 pointer to the first matrix A₀ on the GPU. Each A_i 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_i) and the next one (A_{i+1})

- **x** – [in] pointer to the first vector x_0 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
- **stridex** – [in] [hipblasStride] stride from the start of one vector(x_i) and the next one (x_{i+1})
- **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.

5.4 BLAS Extensions

List of BLAS Extension Functions

- *hipblasGemmEx + Batched, StridedBatched*
- *hipblasTrsmEx + Batched, StridedBatched*
- *hipblasAxpvEx + Batched, StridedBatched*
- *hipblasDotEx + Batched, StridedBatched*
- *hipblasDotcEx + Batched, StridedBatched*
- *hipblasNrm2Ex + Batched, StridedBatched*
- *hipblasRotEx + Batched, StridedBatched*
- *hipblasScalEx + Batched, StridedBatched*

5.4.1 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

$$\begin{aligned} \text{op}(X) &= X && \text{or} \\ \text{op}(X) &= X^{**T} && \text{or} \\ \text{op}(X) &= X^{**H}, \end{aligned}$$

alpha and beta are scalars, and A, B, and C are matrices, with op(A) an m by k matrix, op(B) a k by n matrix and C is a m by n matrix.

- Supported types are determined by the backend. See cuBLAS documentation for cuBLAS backend. For rocBLAS backend, conversion from hipblasComputeType_t to rocblas_datatype_t happens within hipBLAS. Supported types are as follows:

aType	bType	cType	computeType
HIP_R_16F	HIP_R_16F	HIP_R_16F	HIPBLAS_COMPUTE_16F
HIP_R_16F	HIP_R_16F	HIP_R_16F	HIPBLAS_COMPUTE_32F
HIP_R_16F	HIP_R_16F	HIP_R_32F	HIPBLAS_COMPUTE_32F
HIP_R_16BF	HIP_R_16BF	HIP_R_16BF	HIPBLAS_COMPUTE_32F
HIP_R_16BF	HIP_R_16BF	HIP_R_32F	HIPBLAS_COMPUTE_32F
HIP_R_32F	HIP_R_32F	HIP_R_32F	HIPBLAS_COMPUTE_32F
HIP_R_64F	HIP_R_64F	HIP_R_64F	HIPBLAS_COMPUTE_64F
HIP_R_8I	HIP_R_8I	HIP_R_32I	HIPBLAS_COMPUTE_32I
HIP_C_32F	HIP_C_32F	HIP_C_32F	HIPBLAS_COMPUTE_32F
HIP_C_64F	HIP_C_64F	HIP_C_64F	HIPBLAS_COMPUTE_64F

hipblasGemmExWithFlags is also available which is identical to hipblasGemmEx with the addition of a “flags” parameter which controls flags used in Tensile to control gemm algorithms with the rocBLAS backend. When using a cuBLAS backend this parameter is ignored.

With HIPBLAS_V2 define, hipblasGemmEx accepts hipDataType for aType, bType, and cType. It also accepts hipblasComputeType_t for computeType. hipblasGemmEx will no longer support hipblasDataType_t for these parameters in a future release. hipblasGemmEx follows the same convention.

```
typedef hipblasStatus_t hipblasGemmEx(hipblasHandle_t handle,
```

```
hipblasOperation_t transA,
hipblasOperation_t transB,
int m,
int n,
int k,
const void* alpha,
const void* A,
hipDataType aType,
int lda,
const void* B,
hipDataType bType,
int ldb,
const void* beta,
void* C,
hipDataType cType,
int ldc,
hipblasComputeType_t computeType,
hipblasGemmAlgo_t algo)
```

```
hipblasStatus_t hipblasGemmExWithFlags(hipblasHandle_t handle,
hipblasOperation_t transA,
```

(continues on next page)

(continued from previous page)

```

hipblasOperation_t  transB,
int                m,
int                n,
int                k,
const void*        alpha,
const void*        A,
hipDataType        aType,
int                lda,
const void*        B,
hipDataType        bType,
int                ldb,
const void*        beta,
void*              C,
hipDataType        cType,
int                ldc,
hipblasComputeType_t computeType,
hipblasGemmAlgo_t  algo,
hipblasGemmFlags_t flags)

#else // [DEPRECATED]

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)

hipblasStatus_t hipblasGemmExWithFlags(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,

```

(continues on next page)

(continued from previous page)

```

hipblasDatatype_t bType,
int ldb,
const void* beta,
void* C,
hipblasDatatype_t cType,
int ldc,
hipblasDatatype_t computeType,
hipblasGemmAlgo_t algo,
hipblasGemmFlags_t flags)

#endif

```

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] [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] [DEPRECATED] specifies the datatype of matrix A.
[hipDataType] 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] [DEPRECATED] specifies the datatype of matrix B.
[hipDataType] 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] [DEPRECATED] specifies the datatype of matrix C.
[hipDataType] specifies the datatype of matrix C.
- **ldc** – [in] [int] specifies the leading dimension of C.
- **computeType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipblasComputeType_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 $\text{op}(X) = X^{**H}$, α and β 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.

hipblasGemmBatchedExWithFlags is also available which is identical to hipblasGemmBatchedEx with the addition of a “flags” parameter which controls flags used in Tensile to control gemm algorithms with the rocBLAS backend. When using a cuBLAS backend this parameter is ignored.

With HIPBLAS_V2 define, hipblasGemmBatchedEx accepts hipDataType for aType, bType, and cType. It also accepts hipblasComputeType_t for computeType. hipblasGemmBatchedEx will no longer support hipblasDatatype_t for these parameters in a future release. hipblasGemmBatchedExWithFlags follows the same convention.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2
```

```
hipblasStatus_t hipblasGemmBatchedEx(hipblasHandle_t    handle,
                                     hipblasOperation_t  transA,
                                     hipblasOperation_t  transB,
                                     int                  m,
                                     int                  n,
                                     int                  k,
                                     const void*         alpha,
                                     const void*         A[],
                                     hipDataType          aType,
                                     int                  lda,
                                     const void*         B[],
                                     hipDataType          bType,
                                     int                  ldb,
                                     const void*         beta,
                                     void*               C[],
                                     hipDataType          cType,
                                     int                  ldc,
                                     int                  batchCount,
                                     hipblasComputeType_t computeType,
                                     hipblasGemmAlgo_t    algo)

hipblasStatus_t hipblasGemmBatchedExWithFlags(hipblasHandle_t    handle,
                                               hipblasOperation_t  transA,
                                               hipblasOperation_t  transB,
                                               int                  m,
                                               int                  n,
```

(continues on next page)

(continued from previous page)

```

        int k,
        const void* alpha,
        const void* A[],
        hipDataType aType,
        int lda,
        const void* B[],
        hipDataType bType,
        int ldb,
        const void* beta,
        void* C[],
        hipDataType cType,
        int ldc,
        int batchSize,
        hipblasComputeType_t computeType,
        hipblasGemmAlgo_t algo,
        hipblasGemmFlags_t flags)

#else // [DEPRECATED]

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 batchSize,
        hipblasDatatype_t computeType,
        hipblasGemmAlgo_t algo)

hipblasStatus_t hipblasGemmBatchedExWithFlags(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,

```

(continues on next page)

(continued from previous page)

```

int          ldb,
const void*  beta,
void*        C[],
hipblasDatatype_t cType,
int          ldc,
int          batchCount,
hipblasDatatype_t computeType,
hipblasGemmAlgo_t algo,
hipblasGemmFlags_t flags)

#endif

```

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] [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] [DEPRECATED] specifies the datatype of each matrix A_i.
[hipDataType] 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] [DEPRECATED] specifies the datatype of each matrix B_i.
[hipDataType] 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] [DEPRECATED] specifies the datatype of each matrix C_i.
[hipDataType] 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] [DEPRECATED] specifies the datatype of computation.

[hipblasComputeType_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*op(A_i)*op(B_i) + beta*C_i, for i = 1, ..., batchCount
```

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 batchCount strided_batched matrix, 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.

hipblasGemmStridedBatchedExWithFlags is also available which is identical to hipblasStridedBatchedGemmEx with the addition of a “flags” parameter which controls flags used in Tensile to control gemm algorithms with the rocBLAS backend. When using a cuBLAS backend this parameter is ignored.

With HIPBLAS_V2 define, hipblasGemmStridedBatchedEx accepts hipDataType for aType, bType, and cType. It also accepts hipblasComputeType_t for computeType. hipblasGemmStridedBatchedEx will no longer support hipblasDatatype_t for these parameters in a future release. hipblasGemmStridedBatchedExWithFlags follows the same convention.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2
```

```
hipblasStatus_t hipblasGemmStridedBatchedEx(hipblasHandle_t    handle,
                                             hipblasOperation_t transA,
                                             hipblasOperation_t transB,
                                             int                 m,
                                             int                 n,
                                             int                 k,
                                             const void*         alpha,
                                             const void*         A,
                                             hipDataType          aType,
                                             int                 lda,
                                             hipblasStride        strideA,
```

(continues on next page)

(continued from previous page)

```

        const void*      B,
        hipDataType      bType,
        int              ldb,
        hipblasStride    strideB,
        const void*      beta,
        void*            C,
        hipDataType      cType,
        int              ldc,
        hipblasStride    strideC,
        int              batchCount,
        hipblasComputeType_t computeType,
        hipblasGemmAlgo_t algo)

    hipblasStatus_t hipblasGemmStridedBatchedExWithFlags(hipblasHandle_t
↪handle,
                                                         hipblasOperation_t
↪transA,
                                                         hipblasOperation_t
↪transB,
                                                         int          m,
                                                         int          n,
                                                         int          k,
                                                         const void*   alpha,
                                                         const void*   A,
                                                         hipDataType   aType,
                                                         int           lda,
                                                         hipblasStride
↪strideA,
                                                         const void*   B,
                                                         hipDataType   bType,
                                                         int           ldb,
                                                         hipblasStride
↪strideB,
                                                         const void*   beta,
                                                         void*        C,
                                                         hipDataType   cType,
                                                         int           ldc,
                                                         hipblasStride
↪strideC,
                                                         int
↪batchCount,
                                                         hipblasComputeType_t
↪computeType,
                                                         hipblasGemmAlgo_t   algo,
                                                         hipblasGemmFlags_t flags)

#else // [DEPRECATED]

    hipblasStatus_t hipblasGemmStridedBatchedEx(hipblasHandle_t  handle,
        hipblasOperation_t transA,
        hipblasOperation_t transB,
        int                m,

```

(continues on next page)

(continued from previous page)

```

        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)

    hipblasStatus_t hipblasGemmStridedBatchedExWithFlags(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,
        hipblasGemmFlags_t flags)

```

(continues on next page)

(continued from previous page)

#endif

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] [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_1.
- **aType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of each matrix A_i.
[hipDataType] specifies the datatype of each matrix A_i.
- **lda** – [in] [int] specifies the leading dimension of each A_i.
- **strideA** – [in] [hipblasStride] specifies stride from start of one A_i matrix to the next A_(i + 1).
- **B** – [in] [void *] device pointer pointing to first matrix B_1.
- **bType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of each matrix B_i.
[hipDataType] specifies the datatype of each matrix B_i.
- **ldb** – [in] [int] specifies the leading dimension of each B_i.
- **strideB** – [in] [hipblasStride] specifies stride from start of one B_i matrix to the next B_(i + 1).
- **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_1.
- **cType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of each matrix C_i.
[hipDataType] specifies the datatype of each matrix C_i.
- **ldc** – [in] [int] specifies the leading dimension of each C_i.
- **strideC** – [in] [hipblasStride] specifies stride from start of one C_i matrix to the next C_(i + 1).
- **batchCount** – [in] [int] number of gemm operations in the batch.
- **computeType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipblasComputeType_t] specifies the datatype of computation.

- **algo** – [in] [hipblasGemmAlgo_t] enumerant specifying the algorithm type.

5.4.2 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

$$\text{op}(A) * X = \alpha * B \quad \text{or} \quad X * \text{op}(A) = \alpha * B,$$

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.

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 hipblasTrtriBatched 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

With HIPBLAS_V2 define, hipblasTrsmEx accepts hipDataType for computeType rather than hipblasDatatype_t. hipblasTrsmEx will only accept hipDataType in a future release.

```
#ifndef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

hipblasStatus_t hipblasTrsmEx(hipblasHandle_t    handle,
                              hipblasSideMode_t  side,
                              hipblasFillMode_t  uplo,
```

(continues on next page)

(continued from previous page)

```

        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,
        hipDataType        computeType)

#else // [DEPRECATED]

    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)

#endif

```

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, lda $\geq \max(1, m)$, if side = HIPBLAS_SIDE_RIGHT, 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. 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] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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 invA matrix between runs. If invA == NULL, hipblasTrsmBatchedEx will automatically calculate each invA_i on every run.

Setting up invA: Each accepted invA_i matrix consists of the packed 128x128 inverses of the diagonal blocks of matrix A_i , followed by any smaller diagonal block that remains. To set up each invA_i it is recommended that hipblasTrtriBatched be used with matrix A_i as the input. invA is an array of pointers of batchCount length holding each invA_i .

Device memory of size $128 \times k$ should be allocated for each 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 invA_i should be passed as invASize.

To begin, hipblasTrtriBatched must be called on the full 128x128 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:

- $n = k \% 128$
- $\text{invA} = \text{invA} + \text{stride_invA} * \text{previousBatchCount}$
- $\text{ldinvA} = 128$
- $\text{batchCount} = 1$

With HIPBLAS_V2 define, hipblasTrsmBatchedEx accepts hipDataType for computeType rather than hipblas-Datatype_t. hipblasTrsmBatchedEx will only accept hipDataType in a future release.

```
#ifndef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

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,
                                     hipDataType           computeType)

#else // [DEPRECATED]

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)

#endif
```

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_i . each A_i 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_i . 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 array of device pointers storing each matrix B_i . each B_i is of dimension (ldb, n). Before entry, the leading m by n part of the array B_i must contain the right-hand side 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 . $\text{ldb} \geq \max(1, m)$.
- **batchCount** – [in] [int] specifies how many batches.
- **invA** – [in] [void *] device array of device pointers storing the inverse diagonal blocks of each A_i . 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 .
- **computeType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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 α is a scalar, X and B are strided batched m by n matrices, A is a strided batched triangular matrix and $\text{op}(A_i)$ is one of

$\text{op}(A_i) = A_i$ or $\text{op}(A_i) = A_i^T$ 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 each invA_i matrix between runs. If $\text{invA} == \text{NULL}$, `hipblasTrsmStridedBatchedEx` will automatically calculate each invA_i on every run.

Setting up invA : Each accepted invA_i matrix consists of the packed 128×128 inverses of the diagonal blocks of matrix A_i , followed by any smaller diagonal block that remains. To set up invA_i it is recommended that `hipblasTrtriBatched` be used with matrix A_i as the input. invA is a contiguous piece of memory holding each invA_i .

Device memory of size $128 \times k$ should be allocated for each 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 invA_i should be passed as invASize .

To begin, `hipblasTrtriBatched` must be called on the full 128×128 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:

- $n = k \% 128$
- $\text{invA} = \text{invA} + \text{stride_invA} * \text{previousBatchCount}$
- $\text{ldinvA} = 128$
- $\text{batchCount} = 1$

With `HIPBLAS_V2` define, `hipblasStridedBatchedTrsmEx` accepts `hipDataType` for `computeType` rather than `hipblasDatatype_t`. `hipblasTrsmStridedBatchedEx` will only accept `hipDataType` in a future release.

```

#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    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,

```

(continues on next page)

(continued from previous page)

```

                                const void*    invA,
                                int             invASize,
                                hipblasStride   strideInvA,
                                hipDataType     computeType);

#else // [DEPRECATED]

    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)

#endif

```

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 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] [DEPRECATED] specifies the datatype of computation.
[hipDataType] specifies the datatype of computation.

5.4.3 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.

With HIPBLAS_V2 define, hipblasAxyEx accepts hipDataType for alphaType, xType, yType, and executionType rather than hipblasDatatype_t. hipblasAxyEx will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

hipblasStatus_t hipblasAxyEx(hipblasHandle_t handle,
                             int n,
                             const void* alpha,
                             hipDataType alphaType,
                             const void* x,
                             hipDataType xType,
                             int incx,
                             void* y,
                             hipDataType yType,
                             int incy,
```

(continues on next page)

(continued from previous page)

```

hipDataType      executionType)

#else // [DEPRECATED]

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)

#endif

```

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] [DEPRECATED] specifies the datatype of alpha.
[hipDataType] specifies the datatype of alpha.
- **x** – [in] device pointer storing vector x.
- **xType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of vector x.
[hipDataType] 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] [DEPRECATED] specifies the datatype of vector y.
[hipDataType] specifies the datatype of vector y.
- **incy** – [in] [int] specifies the increment for the elements of y.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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.

With HIPBLAS_V2 define, hipblasAxyBatchedEx accepts hipDataType for alphaType, xType, yType, and executionType rather than hipblasDatatype_t. hipblasAxyBatchedEx will only accept hipDataType in a future release.

```
#ifndef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasAxyBatchedEx(hipblasHandle_t handle,
                                        int n,
                                        const void* alpha,
                                        hipDataType alphaType,
                                        const void* x,
                                        hipDataType xType,
                                        int incx,
                                        void* y,
                                        hipDataType yType,
                                        int incy,
                                        int batchCount,
                                        hipDataType executionType)

#else // [DEPRECATED]

    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)

#endif
```

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] [DEPRECATED] specifies the datatype of alpha.
[hipDataType] specifies the datatype of alpha.
- x** – [in] device array of device pointers storing each vector x_i .
- xType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of each vector x_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of each vector y_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of computation.
[hipDataType] specifies the datatype of computation.

hipblasStatus_t **hipblasAxyStridedBatchedEx**(*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.

axyStridedBatchedEx 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.

With HIPBLAS_V2 define, hipblasAxyStridedBatchedEx accepts hipDataType for alphaType, xType, yType, and executionType rather than hipblasDatatype_t. hipblasAxyStridedBatchedEx will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasAxyStridedBatchedEx(hipblasHandle_t handle,
                                                int n,
                                                const void* alpha,
                                                hipDataType alphaType,
                                                const void* x,
                                                hipDataType xType,
                                                int incx,
                                                hipblasStride stridex,
                                                void* y,
                                                hipDataType yType,
                                                int incy,
                                                hipblasStride stridey,
                                                int batchCount,
                                                hipDataType executionType)

#else // [DEPRECATED]

    hipblasStatus_t hipblasAxyStridedBatchedEx(hipblasHandle_t handle,
                                                int n,
                                                const void* alpha,
                                                hipblasDatatype_t alphaType,
```

(continues on next page)

(continued from previous page)

```

const void*      x,
hipblasDatatype_t xType,
int             incx,
hipblasStride    stridex,
void*           y,
hipblasDatatype_t yType,
int             incy,
hipblasStride    stridey,
int             batchCount,
hipblasDatatype_t executionType)

#endif

```

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] [DEPRECATED] specifies the datatype of alpha.
[hipDataType] specifies the datatype of alpha.
- **x** – [in] device pointer to the first vector x_1 .
- **xType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of each vector x_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of each vector y_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of computation.
[hipDataType] specifies the datatype of computation.

5.4.4 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.

With HIPBLAS_V2 define, hipblasDot(c)Ex accepts hipDataType for xType, yType, resultType, and executionType rather than hipblasDatatype_t. hipblasDot(c)Ex will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasDotEx(hipblasHandle_t handle,
                                int n,
                                const void* x,
                                hipDataType xType,
                                int incx,
                                const void* y,
                                hipDataType yType,
                                int incy,
                                void* result,
                                hipDataType resultType,
                                hipDataType executionType)

#else // [DEPRECATED]

    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)

#endif
```

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] [DEPRECATED] specifies the datatype of vector x.
[hipDataType] 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] [DEPRECATED] specifies the datatype of vector y.
[hipDataType] 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] [DEPRECATED] specifies the datatype of the result.
[hipDataType] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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.

With HIPBLAS_V2 define, hipblasDot(c)BatchedEx accepts hipDataType for xType, yType, resultType, and executionType rather than hipblasDatatype_t. hipblasDot(c)BatchedEx will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2
```

```
hipblasStatus_t hipblasDotBatchedEx(hipblasHandle_t handle,
                                     int                n,
                                     const void*         x,
                                     hipDataType          xType,
                                     int                  incx,
                                     const void*         y,
                                     hipDataType          yType,
                                     int                  incy,
```

(continues on next page)

(continued from previous page)

```

                                int          batchCount,
                                void*        result,
                                hipDataType   resultType,
                                hipDataType   executionType)

#else // [DEPRECATED]

    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)

#endif

```

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] [DEPRECATED] specifies the datatype of each vector x_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of each vector y_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of the result.
[hipDataType] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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, ..., batchCount

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

With HIPBLAS_V2 define, hipblasDot(c)StridedBatchedEx accepts hipDataType for xType, yType, resultType, and executionType rather than hipblasDatatype_t. hipblasDot(c)StridedBatchedEx will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2
```

```
    hipblasStatus_t hipblasDotStridedBatchedEx(hipblasHandle_t handle,
                                                int n,
                                                const void* x,
                                                hipDataType xType,
                                                int incx,
                                                hipblasStride stridex,
                                                const void* y,
                                                hipDataType yType,
                                                int incy,
                                                hipblasStride stridey,
                                                int batchCount,
                                                void* result,
                                                hipDataType resultType,
                                                hipDataType executionType)
```

```
#else // [DEPRECATED]
```

```
    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,
```

(continues on next page)

(continued from previous page)

```

void*      result,
hipblasDatatype_t resultType,
hipblasDatatype_t executionType)

#endif

```

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] [DEPRECATED] specifies the datatype of each vector x_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of each vector y_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of the result.
[hipDataType] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] specifies the datatype of computation.

5.4.5 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.

With HIPBLAS_V2 define, hipblasDot(c)Ex accepts hipDataType for xType, yType, resultType, and executionType rather than hipblasDatatype_t. hipblasDot(c)Ex will only accept hipDataType in a future release.

```

#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasDotEx(hipblasHandle_t handle,
                                int n,
                                const void* x,
                                hipDataType xType,
                                int incx,
                                const void* y,
                                hipDataType yType,
                                int incy,
                                void* result,
                                hipDataType resultType,
                                hipDataType executionType)

#else // [DEPRECATED]

    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)

#endif

```

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] [DEPRECATED] specifies the datatype of vector x.
[hipDataType] 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] [DEPRECATED] specifies the datatype of vector y.
[hipDataType] 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] [DEPRECATED] specifies the datatype of the result.
[hipDataType] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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 $i = 1, \dots, \text{batchCount}$

– Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

With HIPBLAS_V2 define, hipblasDot(c)BatchedEx accepts hipDataType for xType, yType, resultType, and executionType rather than hipblasDatatype_t. hipblasDot(c)BatchedEx will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasDotBatchedEx(hipblasHandle_t handle,
                                        int n,
                                        const void* x,
                                        hipDataType xType,
                                        int incx,
                                        const void* y,
                                        hipDataType yType,
                                        int incy,
                                        int batchCount,
                                        void* result,
                                        hipDataType resultType,
                                        hipDataType executionType)

#else // [DEPRECATED]

    hipblasStatus_t hipblasDotBatchedEx(hipblasHandle_t handle,
                                        int n,
                                        const void* x,
                                        hipblasDatatype_t xType,
```

(continues on next page)

(continued from previous page)

```

        int incx,
        const void* y,
        hipblasDatatype_t yType,
        int incy,
        int batchCount,
        void* result,
        hipblasDatatype_t resultType,
        hipblasDatatype_t executionType)

#endif

```

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] [DEPRECATED] specifies the datatype of each vector x_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of each vector y_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of the result.
[hipDataType] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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.

With HIPBLAS_V2 define, hipblasDot(c)StridedBatchedEx accepts hipDataType for xType, yType, resultType, and executionType rather than hipblasDatatype_t. hipblasDot(c)StridedBatchedEx will only accept hipDataType in a future release.

```
typedef hipblasStatus_t hipblasDotStridedBatchedEx(hipblasHandle_t handle,
```

```

    int n,
    const void* x,
    hipDataType xType,
    int incx,
    hipblasStride stridex,
    const void* y,
    hipDataType yType,
    int incy,
    hipblasStride stridey,
    int batchCount,
    void* result,
    hipDataType resultType,
    hipDataType executionType)

```

```
typedef 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)

```

```
typedef hipblasStatus_t hipblasDotStridedBatchedEx(hipblasHandle_t handle,
```

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] [DEPRECATED] specifies the datatype of each vector x_i .

[hipDataType] specifies the datatype of each vector x_i .

- **incx** – [in] [int] specifies the increment for the elements of each x_i .
- **stridedx** – [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] [DEPRECATED] specifies the datatype of each vector y_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of the result.

[hipDataType] specifies the datatype of the result.

- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.

[hipDataType] specifies the datatype of computation.

5.4.6 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.

With HIPBLAS_V2 define, hipblasNrm2Ex accepts hipDataType for xType, resultType, and executionType rather than hipblasDatatype_t. hipblasNrm2Ex will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2
```

```
hipblasStatus_t hipblasNrm2Ex(hipblasHandle_t handle,
                              int                n,
                              const void*        x,
                              hipDataType         xType,
                              int                incx,
                              void*              result,
                              hipDataType         resultType,
                              hipDataType         executionType)
```

(continues on next page)

(continued from previous page)

```

else // [DEPRECATED]

    hipblasStatus_t hipblasNrm2Ex(hipblasHandle_t    handle,
                                   int                n,
                                   const void*        x,
                                   hipblasDatatype_t  xType,
                                   int                incx,
                                   void*              result,
                                   hipblasDatatype_t  resultType,
                                   hipblasDatatype_t  executionType)

endif

```

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] [DEPRECATED] specifies the datatype of the vector x.
[hipDataType] 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] [DEPRECATED] specifies the datatype of the result.
[hipDataType] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] specifies the datatype of computation.

hipblasStatus_t **hipblasNrm2BatchedEx**(*hipblasHandle_t* handle, int n, const void *x, *hipblasDatatype_t* xType, int incx, int batchSize, 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, ..., batchSize
result := sqrt( x_i**H*x_i ) for complex vectors x, for i = 1, ..., batchSize

```

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

With HIPBLAS_V2 define, hipblasNrm2BatchedEx accepts hipDataType for xType, resultType, and executionType rather than hipblasDatatype_t. hipblasNrm2BatchedEx will only accept hipDataType in a future release.

```

#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasNrm2BatchedEx(hipblasHandle_t handle,
                                         int n,
                                         const void* x,
                                         hipDataType_t xType,
                                         int incx,
                                         int batchCount,
                                         void* result,
                                         hipDataType_t resultType,
                                         hipDataType_t executionType)

#else // [DEPRECATED]

    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)

#endif

```

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] [DEPRECATED] specifies the datatype of each vector x_i .
[hipDataType] 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 $n \leq 0$, $incx \leq 0$.
- **resultType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of the result.
[hipDataType] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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.

With HIPBLAS_V2 define, hipblasNrm2StridedBatchedEx accepts hipDataType for xType, resultType, and executionType rather than hipblasDatatype_t. hipblasNrm2StridedBatchedEx will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasNrm2StridedBatchedEx(hipblasHandle_t handle,
                                                int n,
                                                const void* x,
                                                hipDataType xType,
                                                int incx,
                                                hipblasStride stridex,
                                                int batchCount,
                                                void* result,
                                                hipDataType resultType,
                                                hipDataType executionType)

#else // [DEPRECATED]

    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)

#endif
```

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] [DEPRECATED] specifies the datatype of each vector x_i.
[hipDataType] 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 stride_x $\geq n * incx$.
- **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 $n \leq 0$, $incx \leq 0$.
- **resultType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of the result.
[hipDataType] specifies the datatype of the result.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] specifies the datatype of computation.

5.4.7 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.

With HIPBLAS_V2 define, hipblasRotEx accepts hipDataType for xType, yType, csType, and executionType rather than hipblasDatatype_t. hipblasRotEx will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasRotEx(hipblasHandle_t handle,
                                int                n,
                                void*             x,
                                hipDataType        xType,
                                int                incx,
                                void*             y,
                                hipDataType        yType,
                                int                incy,
                                const void*        c,
                                const void*        s,
                                hipDataType        csType,
                                hipDataType        executionType)

#else // [DEPRECATED]
```

(continues on next page)

(continued from previous page)

```

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)

#endif

```

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] [DEPRECATED] specifies the datatype of vector x.
[hipDataType] 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] [DEPRECATED] specifies the datatype of vector y.
[hipDataType] 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] [DEPRECATED] specifies the datatype of c and s.
[hipDataType] specifies the datatype of c and s.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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.

With `HIPBLAS_V2` define, `hipblasRotBatchedEx` accepts `hipDataType` for `xType`, `yType`, `csType`, and `executionType` rather than `hipblasDatatype_t`. `hipblasRotBatchedEx` will only accept `hipDataType` in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasRotBatchedEx(hipblasHandle_t handle,
                                        int n,
                                        void* x,
                                        hipDataType xType,
                                        int incx,
                                        void* y,
                                        hipDataType yType,
                                        int incy,
                                        const void* c,
                                        const void* s,
                                        hipDataType csType,
                                        int batchCount,
                                        hipDataType executionType)

#else // [DEPRECATED]

    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)

#endif
```

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]` `[DEPRECATED]` specifies the datatype of each vector `x_i`.
`[hipDataType]` 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] [DEPRECATED] specifies the datatype of each vector y_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of c and s .
[hipDataType] 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] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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.

With HIPBLAS_V2 define, `hipblasRotStridedBatchedEx` accepts `hipDataType` for `xType`, `yType`, `csType`, and `executionType` rather than `hipblasDatatype_t`. `hipblasRotStridedBatchedEx` will only accept `hipDataType` in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

hipblasStatus_t hipblasRotStridedBatchedEx(hipblasHandle_t handle,
                                           int           n,
                                           void*        x,
                                           hipDataType    xType,
                                           int           incx,
                                           hipblasStride  stridex,
                                           void*        y,
                                           hipDataType    yType,
                                           int           incy,
                                           hipblasStride  stridey,
                                           const void*    c,
                                           const void*    s,
                                           hipDataType    csType,
```

(continues on next page)

(continued from previous page)

```

                                int          batchCount,
                                hipDataType   executionType)

#else // [DEPRECATED]

    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)

#endif

```

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] [DEPRECATED] specifies the datatype of each vector x_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of each vector y_i .
[hipDataType] 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] [DEPRECATED] specifies the datatype of c and s .
[hipDataType] 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] [DEPRECATED] specifies the datatype of computation.

[hipDataType] specifies the datatype of computation.

5.4.8 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.

With HIPBLAS_V2 define, hipblasScalEx accepts hipDataType for alphaType, xType, and executionType rather than hipblasDatatype_t. hipblasScalEx will only accept hipDataType in a future release.

```
#ifndef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasScalEx(hipblasHandle_t handle, a
                                int n,
                                const void* alpha,
                                hipDataType alphaType,
                                void* x,
                                hipDataType xType,
                                int incx,
                                hipDataType executionType)

#else // [DEPRECATED]

    hipblasStatus_t hipblasScalEx(hipblasHandle_t handle,
                                int n,
                                const void* alpha,
                                hipblasDatatype_t alphaType,
                                void* x,
                                hipblasDatatype_t xType,
                                int incx,
                                hipblasDatatype_t executionType)

#endif
```

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] [DEPRECATED] specifies the datatype of alpha.
[hipDataType] specifies the datatype of alpha.

- **x** – [inout] device pointer storing vector x.
- **xType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of vector x.
[hipDataType] specifies the datatype of vector x.
- **incx** – [in] [int] specifies the increment for the elements of x.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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.

With HIPBLAS_V2 define, hipblasScalBatchedEx accepts hipDataType for alphaType, xType, and executionType rather than hipblasDatatype_t. hipblasScalBatchedEx will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasScalBatchedEx(hipblasHandle_t handle,
                                         int n,
                                         const void* alpha,
                                         hipDataType alphaType,
                                         void* x,
                                         hipDataType xType,
                                         int incx,
                                         int batchCount,
                                         hipDataType executionType)

#else // [DEPRECATED]

    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)

#endif
```

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] [DEPRECATED] specifies the datatype of **alpha**.
[hipDataType] specifies the datatype of **alpha**.
- **x** – [inout] device array of device pointers storing each vector **x_i**.
- **xType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of each vector **x_i**.
[hipDataType] 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] [DEPRECATED] specifies the datatype of computation.
[hipDataType] 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.

$x := \text{alpha} * x$

- Supported types are determined by the backend. See rocBLAS/cuBLAS documentation.

With HIPBLAS_V2 define, hipblasScalStridedBatchedEx accepts hipDataType for alphaType, xType, and executionType rather than hipblasDatatype_t. hipblasScalStridedBatchedEx will only accept hipDataType in a future release.

```
#ifdef HIPBLAS_V2 // available in hipBLAS version 2.0.0 and later with -DHIPBLAS_V2

    hipblasStatus_t hipblasScalStridedBatchedEx(hipblasHandle_t handle,
                                                int n,
                                                const void* alpha,
                                                hipDataType alphaType,
                                                void* x,
                                                hipDataType xType,
                                                int incx,
                                                hipblasStride stridex,
                                                int batchCount,
                                                hipDataType executionType)

#else // [DEPRECATED]

    hipblasStatus_t hipblasScalStridedBatchedEx(hipblasHandle_t handle,
                                                int n,
                                                const void* alpha,
                                                hipblasDatatype_t alphaType,
```

(continues on next page)

(continued from previous page)

```

void*      x,
hipblasDatatype_t xType,
int        incx,
hipblasStride stridex,
int        batchCount,
hipblasDatatype_t executionType)

#endif

```

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] [DEPRECATED] specifies the datatype of **alpha**.
[hipDataType] specifies the datatype of **alpha**.
- **x** – [inout] device pointer to the first vector **x₁**.
- **xType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of each vector **x_i**.
[hipDataType] 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 **stridex** $\geq n * \text{incx}$.
- **batchCount** – [in] [int] number of instances in the batch.
- **executionType** – [in] [hipblasDatatype_t] [DEPRECATED] specifies the datatype of computation.
[hipDataType] specifies the datatype of computation.

5.5 SOLVER API

List of SOLVER APIs

- *hipblasXgetrf + Batched, stridedBatched*
- *hipblasXgetrs + Batched, stridedBatched*
- *hipblasXgetri + Batched, stridedBatched*
- *hipblasXgeqrf + Batched, stridedBatched*
- *hipblasXgels + Batched, StridedBatched*

5.5.1 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 $lda \times n$.

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. $lda \geq n$.

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 >= 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 >= 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. $\text{lda} \geq n$.
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 $\text{strideA} \geq \text{lda} * n$
- **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 $\text{ipiv}_i[j]$. 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 $\text{strideP} \geq n$.
- **info** – [out] pointer to int. Array of batchCount integers on the GPU.
If $\text{info}[i] = 0$, successful exit for factorization of A_i . If $\text{info}[i] = j > 0$, U_i is singular. $U_i[j,j]$ is the first zero pivot.
- **batchCount** – [in] int. $\text{batchCount} \geq 0$.
Number of matrices in the batch.

5.5.2 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 $\text{ldb} \times \text{nrhs}$.
On entry, the right hand side matrix B. On exit, the solution matrix X.
- **ldb** – [in] `int`. $\text{ldb} \geq n$.
The leading dimension of B.
- **info** – [out] pointer to a `int` on the host.
If $\text{info} = 0$, successful exit. If $\text{info} = j < 0$, the argument at position $-j$ 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:

$$\begin{aligned}
 A_i X_i &= B_i && \text{not transposed,} \\
 A_i^T X_i &= B_i && \text{transposed, or} \\
 A_i^H X_i &= B_i && \text{conjugate transposed.}
 \end{aligned}$$

Matrix A_i is defined by its triangular factors as returned by *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. $n \geq 0$.
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 argument at position $-j$ 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:

$$\begin{array}{ll} A_i X_i = B_i & \text{not transposed,} \\ A_i^T X_i = B_i & \text{transposed, or} \\ A_i^H X_i = B_i & \text{conjugate transposed.} \end{array}$$

Matrix A_i is defined by its triangular factors as returned by *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 argument at position $-j$ is invalid.
- **batchCount** – [in] int. $\text{batchCount} \geq 0$.
Number of instances (systems) in the batch.

5.5.3 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 $lda \times n$.
The factors L_i and U_i of the factorization $A_i = P_i * L_i * U_i$ returned by *getrfBatched*.
- **lda** – [in] `int`. $lda \geq n$.
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 $ldc \times n$.
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.

5.5.4 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 argument at position $-j$ 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 batchCount)

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 batchCount)

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 `ipiv_i` of corresponding Householder scalars.

- **info** – [out] pointer to a int on the host.

If `info = 0`, successful exit. If `info = j < 0`, the argument at position `-j` is invalid.

- **batchCount** – [in] int. `batchCount >= 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` = $j < 0$, the argument at position $-j$ is invalid.
- **batchCount** – [in] `int`. `batchCount` ≥ 0 .
Number of matrices in the batch.

5.5.5 hipblasXgels + Batched, StridedBatched

hipblasStatus_t **hipblasSgels**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, float *A, const int lda, float *B, const int ldb, int *info, int *deviceInfo)

hipblasStatus_t **hipblasDgels**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, double *A, const int lda, double *B, const int ldb, int *info, int *deviceInfo)

hipblasStatus_t **hipblasCgels**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, *hipblasComplex* *A, const int lda, *hipblasComplex* *B, const int ldb, int *info, int *deviceInfo)

hipblasStatus_t **hipblasZgels**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, *hipblasDoubleComplex* *A, const int lda, *hipblasDoubleComplex* *B, const int ldb, int *info, int *deviceInfo)

GELS solves an overdetermined (or underdetermined) linear system defined by an m-by-n matrix A, and a corresponding matrix B, using the QR factorization computed by *GEQRF* (or the LQ factorization computed by “GELQF”).

Depending on the value of trans, the problem solved by this function is either of the form

$$\begin{aligned} AX &= B && \text{not transposed, or} \\ A'X &= B && \text{transposed if real, or conjugate transposed if complex} \end{aligned}$$

If $m \geq n$ (or $m < n$ in the case of transpose/conjugate transpose), the system is overdetermined and a least-squares solution approximating X is found by minimizing

$$\|B - AX\| \quad (\text{or } \|B - A'X\|)$$

If $m < n$ (or $m \geq n$ in the case of transpose/conjugate transpose), the system is underdetermined and a unique solution for X is chosen such that $\|X\|$ is minimal.

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : currently unsupported

Parameters

- **handle** – [in] hipblasHandle_t.
- **trans** – [in] hipblasOperation_t.
Specifies the form of the system of equations.
- **m** – [in] int. $m \geq 0$.
The number of rows of matrix A.
- **n** – [in] int. $n \geq 0$.
The number of columns of matrix A.
- **nrhs** – [in] int. $\text{nrhs} \geq 0$.
The number of columns of matrices B and X; i.e., the columns on the right hand side.
- **A** – [inout] pointer to type. Array on the GPU of dimension $\text{lda} \times n$.
On entry, the matrix A. On exit, the QR (or LQ) factorization of A as returned by “GEQRF” (or “GELQF”).
- **lda** – [in] int. $\text{lda} \geq m$.
Specifies the leading dimension of matrix A.
- **B** – [inout] pointer to type. Array on the GPU of dimension $\text{ldb} \times \text{nrhs}$.
On entry, the matrix B. On exit, when $\text{info} = 0$, B is overwritten by the solution vectors (and the residuals in the overdetermined cases) stored as columns.
- **ldb** – [in] int. $\text{ldb} \geq \max(m, n)$.
Specifies the leading dimension of matrix B.

- **info** – [out] pointer to an int on the host.

If info = 0, successful exit. If info = j < 0, the argument at position -j is invalid.

- **deviceInfo** – [out] pointer to int on the GPU.

If info = 0, successful exit. If info = i > 0, the solution could not be computed because input matrix A is rank deficient; the i-th diagonal element of its triangular factor is zero.

hipblasStatus_t **hipblasSgelsBatched**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, float *const A[], const int lda, float *const B[], const int ldb, int *info, int *deviceInfo, const int batchSize)

hipblasStatus_t **hipblasDgelsBatched**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, double *const A[], const int lda, double *const B[], const int ldb, int *info, int *deviceInfo, const int batchSize)

hipblasStatus_t **hipblasCgelsBatched**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, *hipblasComplex* *const A[], const int lda, *hipblasComplex* *const B[], const int ldb, int *info, int *deviceInfo, const int batchSize)

hipblasStatus_t **hipblasZgelsBatched**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, *hipblasDoubleComplex* *const A[], const int lda, *hipblasDoubleComplex* *const B[], const int ldb, int *info, int *deviceInfo, const int batchSize)

gelsBatched solves a batch of overdetermined (or underdetermined) linear systems defined by a set of m-by-n matrices A_j , and corresponding matrices B_j , using the QR factorizations computed by “GEQRF_BATCHED” (or the LQ factorizations computed by “GELQF_BATCHED”).

For each instance in the batch, depending on the value of trans, the problem solved by this function is either of the form

$$\begin{array}{ll} A_j X_j = B_j & \text{not transposed, or} \\ A_j' X_j = B_j & \text{transposed if real, or conjugate transposed if complex} \end{array}$$

If $m \geq n$ (or $m < n$ in the case of transpose/conjugate transpose), the system is overdetermined and a least-squares solution approximating X_j is found by minimizing

$$\|B_j - A_j X_j\| \quad (\text{or } \|B_j - A_j' X_j\|)$$

If $m < n$ (or $m \geq n$ in the case of transpose/conjugate transpose), the system is underdetermined and a unique solution for X_j is chosen such that $\|X_j\|$ is minimal.

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : s,d,c,z Note that cuBLAS backend supports only the non-transpose operation and only solves over-determined systems ($m \geq n$).

Parameters

- **handle** – [in] *hipblasHandle_t*.
- **trans** – [in] *hipblasOperation_t*.

Specifies the form of the system of equations.

- **m** – [in] int. $m \geq 0$.
The number of rows of all matrices A_j in the batch.
- **n** – [in] int. $n \geq 0$.
The number of columns of all matrices A_j in the batch.
- **nrhs** – [in] int. $nrhs \geq 0$.
The number of columns of all matrices B_j and X_j in the batch; i.e., the columns on the right hand side.
- **A** – [inout] array of pointer to type. Each pointer points to an array on the GPU of dimension $lda*n$.
On entry, the matrices A_j . On exit, the QR (or LQ) factorizations of A_j as returned by “GEQRF_BATCHED” (or “GELQF_BATCHED”).
- **lda** – [in] int. $lda \geq m$.
Specifies the leading dimension of matrices A_j .
- **B** – [inout] array of pointer to type. Each pointer points to an array on the GPU of dimension $ldb*nrhs$.
On entry, the matrices B_j . On exit, when $info[j] = 0$, B_j is overwritten by the solution vectors (and the residuals in the overdetermined cases) stored as columns.
- **ldb** – [in] int. $ldb \geq \max(m,n)$.
Specifies the leading dimension of matrices B_j .
- **info** – [out] pointer to an int on the host.
If $info = 0$, successful exit. If $info = j < 0$, the argument at position $-j$ is invalid.
- **deviceInfo** – [out] pointer to int. Array of batchCount integers on the GPU.
If $deviceInfo[j] = 0$, successful exit for solution of A_j . If $deviceInfo[j] = i > 0$, the solution of A_j could not be computed because input matrix A_j is rank deficient; the i -th diagonal element of its triangular factor is zero.
- **batchCount** – [in] int. $batchCount \geq 0$.
Number of matrices in the batch.

hipblasStatus_t **hipblasSgelsStridedBatched**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, float *A, const int lda, const *hipblasStride* strideA, float *B, const int ldb, const *hipblasStride* strideB, int *info, int *deviceInfo, const int batchCount)

hipblasStatus_t **hipblasDgelsStridedBatched**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, double *A, const int lda, const *hipblasStride* strideA, double *B, const int ldb, const *hipblasStride* strideB, int *info, int *deviceInfo, const int batchCount)

hipblasStatus_t **hipblasCgelsStridedBatched**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, *hipblasComplex* *A, const int lda, const *hipblasStride* strideA, *hipblasComplex* *B, const int ldb, const *hipblasStride* strideB, int *info, int *deviceInfo, const int batchCount)

hipblasStatus_t **hipblasZgelsStridedBatched**(*hipblasHandle_t* handle, *hipblasOperation_t* trans, const int m, const int n, const int nrhs, *hipblasDoubleComplex* *A, const int lda, const *hipblasStride* strideA, *hipblasDoubleComplex* *B, const int ldb, const *hipblasStride* strideB, int *info, int *deviceInfo, const int batchCount)

gelsStridedBatched solves a batch of overdetermined (or underdetermined) linear systems defined by a set of m-by-n matrices A_j , and corresponding matrices B_j , using the QR factorizations computed by “GEQRF_STRIDED_BATCHED” (or the LQ factorizations computed by “GELQF_STRIDED_BATCHED”).

For each instance in the batch, depending on the value of trans, the problem solved by this function is either of the form

$$\begin{array}{ll} A_j X_j = B_j & \text{not transposed, or} \\ A_j' X_j = B_j & \text{transposed if real, or conjugate transposed if complex} \end{array}$$

If $m \geq n$ (or $m < n$ in the case of transpose/conjugate transpose), the system is overdetermined and a least-squares solution approximating X_j is found by minimizing

$$\|B_j - A_j X_j\| \quad (\text{or } \|B_j - A_j' X_j\|)$$

If $m < n$ (or $m \geq n$ in the case of transpose/conjugate transpose), the system is underdetermined and a unique solution for X_j is chosen such that $\|X_j\|$ is minimal.

- Supported precisions in rocSOLVER : s,d,c,z
- Supported precisions in cuBLAS : currently unsupported

Parameters

- **handle** – [in] *hipblasHandle_t*.
- **trans** – [in] *hipblasOperation_t*.
Specifies the form of the system of equations.
- **m** – [in] int. $m \geq 0$.
The number of rows of all matrices A_j in the batch.
- **n** – [in] int. $n \geq 0$.
The number of columns of all matrices A_j in the batch.
- **nrhs** – [in] int. $\text{nrhs} \geq 0$.
The number of columns of all matrices B_j and X_j in the batch; i.e., the columns on the right hand side.
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of strideA).
On entry, the matrices A_j . On exit, the QR (or LQ) factorizations of A_j as returned by “GEQRF_STRIDED_BATCHED” (or “GELQF_STRIDED_BATCHED”).
- **lda** – [in] int. $\text{lda} \geq m$.
Specifies the leading dimension of matrices A_j .

- **strideA** – [in] hipblasStride.
Stride from the start of one matrix A_j to the next one A_{j+1} . There is no restriction for the value of strideA. Normal use case is $\text{strideA} \geq \text{lda} * n$
- **B** – [inout] pointer to type. Array on the GPU (the size depends on the value of strideB).
On entry, the matrices B_j . On exit, when $\text{info}[j] = 0$, each B_j is overwritten by the solution vectors (and the residuals in the overdetermined cases) stored as columns.
- **ldb** – [in] int. $\text{ldb} \geq \max(m, n)$.
Specifies the leading dimension of matrices B_j .
- **strideB** – [in] hipblasStride.
Stride from the start of one matrix B_j to the next one B_{j+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 an int on the host.
If $\text{info} = 0$, successful exit. If $\text{info} = j < 0$, the argument at position $-j$ is invalid.
- **deviceInfo** – [out] pointer to int. Array of batchCount integers on the GPU.
If $\text{deviceInfo}[j] = 0$, successful exit for solution of A_j . If $\text{deviceInfo}[j] = i > 0$, the solution of A_j could not be computed because input matrix A_j is rank deficient; the i -th diagonal element of its triangular factor is zero.
- **batchCount** – [in] int. $\text{batchCount} \geq 0$.
Number of matrices in the batch.

5.6 Auxiliary

5.6.1 hipblasCreate

hipblasStatus_t **hipblasCreate**(*hipblasHandle_t* *handle)

Create hipblas handle.

5.6.2 hipblasDestroy

hipblasStatus_t **hipblasDestroy**(*hipblasHandle_t* handle)

Destroys the library context created using *hipblasCreate()*

5.6.3 hipblasSetStream

hipblasStatus_t **hipblasSetStream**(*hipblasHandle_t* handle, *hipStream_t* streamId)

Set stream for handle.

5.6.4 hipblasGetStream

hipblasStatus_t **hipblasGetStream**(*hipblasHandle_t* handle, *hipStream_t* *streamId)

Get stream[0] for handle.

5.6.5 hipblasSetPointerMode

hipblasStatus_t **hipblasSetPointerMode**(*hipblasHandle_t* handle, *hipblasPointerMode_t* mode)

Set hipblas pointer mode.

5.6.6 hipblasGetPointerMode

hipblasStatus_t **hipblasGetPointerMode**(*hipblasHandle_t* handle, *hipblasPointerMode_t* *mode)

Get hipblas pointer mode.

5.6.7 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

5.6.8 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

5.6.9 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

5.6.10 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, ldb >= rows

5.6.11 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

5.6.12 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

5.6.13 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

5.6.14 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

5.6.15 hipblasSetAtomicsMode

hipblasStatus_t **hipblasSetAtomicsMode**(*hipblasHandle_t* handle, *hipblasAtomicsMode_t* atomics_mode)

Set hipblasSetAtomicsMode.

5.6.16 hipblasGetAtomicsMode

hipblasStatus_t **hipblasGetAtomicsMode**(*hipblasHandle_t* handle, *hipblasAtomicsMode_t* *atomics_mode)

Get hipblasSetAtomicsMode.

5.6.17 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

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.

6.1 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:

```
./rocbblas-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.

If multiple arguments or even multiple functions need to be benchmarked there is support for data driven benchmarks via a yaml format specification file.

```
./hipblas-bench --yaml <file>.yaml
```

An example yaml file that is used for a smoke test is `hipblas_smoke.yaml` but other examples can be found in the rocBLAS repository.

6.2 hipblas-test

hipblas-test is used in performing hipBLAS unit tests and it uses Googletest framework.

To run the hipblas tests:

```
./hipblas-test
```

To run a subset of tests a filter may be provided. For example to only run axpy function tests via command line use:

```
./hipblas-test --gtest_filter=*axpy*
```

The pattern for `--gtest_filter` is:

```
--gtest_filter=POSTIVE_PATTERNS[-NEGATIVE_PATTERNS]
```

If specific function arguments or even multiple functions need to be tested there is support for data driven testing via a yaml format test specification file.

```
./hipblas-test --yaml <file>.yaml
```

An example yaml file that is used to define a smoke test is `hipblas_smoke.yaml` but other examples can be found in the rocBLAS repository. Yaml based tests list function parameter values in the test name which can be also used for test filtering via the `gtest_filter` argument. To run the provided smoke test use:

```
./hipblas-test --yaml hipblas_smoke.yaml
```


DEPRECATIONS BY VERSION

7.1 Announced in hipBLAS 0.49

7.1.1 Inplace hipblasXtrmm will be replaced with out-of-place hipblasXtrmm

The hipblasXtrmm API, along with batched versions, will be changing in hipBLAS 1.0 release to allow in-place and out-of-place behavior. This change will introduce an output matrix 'C', matching the rocblas_xtrmm_outofplace API and the cublasXtrmm API.

7.2 Announced in hipBLAS 0.53

7.2.1 packed_int8x4 datatype will be removed

The packed_int8x4 datatype will be removed in hipBLAS 1.0. There are two int8 datatypes:

- int8_t
- packed_int8x4

int8_t is the C99 unsigned 8 bit integer. packed_int8x4 has 4 consecutive int8_t numbers in the k dimension packed into 32 bits. packed_int8x4 is only used in hipblasGemmEx. int8_t will continue to be available in hipblasGemmEx.

7.3 Announced in hipBLAS 1.0

7.3.1 Legacy BLAS in-place trmm functions will be replaced with trmm functions that support both in-place and out-of-place functionality

Use of the deprecated Legacy BLAS in-place trmm functions will give deprecation warnings telling you to compile with -DHIPBLAS_V1 and use the new in-place and out-of-place trmm functions.

Note that there are no deprecation warnings for the hipBLAS Fortran API.

The Legacy BLAS in-place trmm calculates $B \leftarrow \alpha * op(A) * B$. Matrix B is overwritten by triangular matrix A multiplied by matrix B. The prototype in the include file rocblas-functions.h is:

```
hipblasStatus_t hipblasStrmm(hipblasHandle_t    handle,
                             hipblasSideMode_t  side,
                             hipblasFillMode_t  uplo,
                             hipblasOperation_t transA,
```

(continues on next page)

(continued from previous page)

```
hipblasDiagType_t diag,
int               m,
int               n,
const float*      alpha,
const float*      AP,
int               lda,
float*            BP,
int               ldb);
```

The above is replaced by an in-place and out-of-place trmm that calculates $C \leftarrow \alpha * \text{op}(A) * B$. The prototype is:

```
hipblasStatus_t hipblasStrmmOutofplace(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,
const float*      BP,
int               ldb,
float*            CP,
int               ldc);
```

The new API provides the legacy BLAS in-place functionality if you set pointer C equal to pointer B and ldc equal to ldb.

There are similar deprecations for the `_batched` and `_strided_batched` versions of trmm.

7.4 Removed in hipBLAS 1.0

7.4.1 HIPBLAS_INT8_DATATYPE_PACK_INT8x4 hipblasGemmEx support removed

Packed int8x4 is removed as support for arbitrary dimensioned int8_t data is a superset of this functionality:

- enum `hipblasInt8Datatype_t` is removed
- function `hipblasSetInt8Datatype` is removed
- function `hipblasGetInt8Datatype` is removed

7.5 Announced in hipBLAS 2.0

7.5.1 hipblasDatatype_t will be replaced with hipDataType

Use of hipblasDatatype_t will give deprecation warnings telling you to compile with -DHIPBLAS_V2 and to use hipDataType instead. All functions which currently use hipblasDatatype_t are therefore deprecated as well, and will be replaced with functions which use hipDataType in the place of hipblasDatatype_t. These functions include: hipblasTrsmEx, hipblasGemmEx, hipblasGemmExWithFlags, hipblasAxyEx, hipblasDot(c)Ex, hipblasNrm2Ex, hipblasRotEx, hipblasScalEx, and the batched and strided-batched variants of these. Please see the documentation for each function for more information.

Note that there are no deprecation warnings for the hipBLAS Fortran API.

hipblasDatatype_t will be removed in a future release, and the use of this type in the API will be replaced with hipDataType.

7.5.2 hipblasComplex and hipblasDoubleComplex will be replaced by hipComplex and hipDoubleComplex

Use of these datatypes will give deprecation warnings telling you to compile with -DHIPBLAS_V2 and to use HIP complex types instead. All functions which currently use hipblasComplex and hipblasDoubleComplex are therefore deprecated as well, and will be replaced with functions which use hipComplex and hipDoubleComplex in their place.

Note that there are no deprecation warnings for the hipBLAS Fortran API.

hipComplex and hipDoubleComplex will be removed in a future release, and the use of this type in the API will be replaced by hipComplex and hipDoubleComplex.

ROCM_MATHLIBS_API_USE_HIP_COMPLEX is also deprecated as the behavior provided by defining it will be the default in the future.

7.6 Removed in hipBLAS 2.0

7.6.1 Legacy BLAS in-place trmm is removed

The legacy BLAS in-place hipblasXtrmm that calculates $B \leftarrow \alpha * \text{op}(A) * B$ is removed and replaced with the out-of-place hipblasXtrmm that calculates $C \leftarrow \alpha * \text{op}(A) * B$.

The prototype for the removed legacy BLAS in-place functionality was

```
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*       A,
                             int                lda,
                             float*            B,
                             int                ldb);
```

The prototype for the replacement in-place and out-of-place functionality is

```
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*       A,
                             int                lda,
                             const float*       B,
                             int                ldb,
                             float*             C,
                             int                ldc);
```

The legacy BLAS in-place functionality can be obtained with the new function if you set pointer C equal to pointer B and ldc equal to ldb.

The out-of-place functionality is from setting pointer B distinct from pointer C.

CONTRIBUTING

8.1 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.

8.2 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.

8.2.1 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
```

8.3 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.

LICENSE

MIT License

Copyright (C) 2017-2024 Advanced Micro Devices, Inc. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

This product includes software from copyright holders as shown below, and distributed under their license terms as specified.

Copyright (c) 1992-2022 The University of Tennessee and The University of Tennessee Research Foundation. All rights reserved.

Copyright (c) 2000-2022 The University of California Berkeley. All rights reserved.

Copyright (c) 2006-2022 The University of Colorado Denver. All rights reserved.

\$COPYRIGHT\$

Additional copyrights may follow

\$HEADERS\$

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

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

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

H

- hipblasAtomicsMode_t (C++ enum), 19
- hipblasAtomicsMode_t::HIPBLAS_ATOMICS_ALLOWED (C++ enumerator), 19
- hipblasAtomicsMode_t::HIPBLAS_ATOMICS_NOT_ALLOWED (C++ enumerator), 19
- hipblasAxyBatchedEx (C++ function), 194
- hipblasAxyEx (C++ function), 193
- hipblasAxyStridedBatchedEx (C++ function), 196
- hipblasBfdot (C++ function), 32
- hipblasBfdotBatched (C++ function), 33
- hipblasBfdotStridedBatched (C++ function), 34
- hipblasBfloat16 (C++ struct), 13
- hipblasCaxpy (C++ function), 27
- hipblasCaxpyBatched (C++ function), 28
- hipblasCaxpyStridedBatched (C++ function), 29
- hipblasCcopy (C++ function), 29
- hipblasCcopyBatched (C++ function), 30
- hipblasCcopyStridedBatched (C++ function), 31
- hipblasCdgmm (C++ function), 172
- hipblasCdgmmBatched (C++ function), 173
- hipblasCdgmmStridedBatched (C++ function), 174
- hipblasCdotc (C++ function), 32
- hipblasCdotcBatched (C++ function), 33
- hipblasCdotcStridedBatched (C++ function), 34
- hipblasCdotu (C++ function), 32
- hipblasCdotuBatched (C++ function), 33
- hipblasCdotuStridedBatched (C++ function), 34
- hipblasCgbmv (C++ function), 52
- hipblasCgbmvBatched (C++ function), 53
- hipblasCgbmvStridedBatched (C++ function), 54
- hipblasCgeam (C++ function), 153
- hipblasCgeamBatched (C++ function), 154
- hipblasCgeamStridedBatched (C++ function), 156
- hipblasCgels (C++ function), 232
- hipblasCgelsBatched (C++ function), 234
- hipblasCgelsStridedBatched (C++ function), 235
- hipblasCgemm (C++ function), 124
- hipblasCgemmBatched (C++ function), 125
- hipblasCgemmStridedBatched (C++ function), 127
- hipblasCgemv (C++ function), 56
- hipblasCgemvBatched (C++ function), 57
- hipblasCgemvStridedBatched (C++ function), 58
- hipblasCgeqrf (C++ function), 228
- hipblasCgeqrfBatched (C++ function), 229
- hipblasCgeqrfStridedBatched (C++ function), 231
- hipblasCgerc (C++ function), 59
- hipblasCgercBatched (C++ function), 60
- hipblasCgercStridedBatched (C++ function), 61
- hipblasCgeru (C++ function), 59
- hipblasCgeruBatched (C++ function), 60
- hipblasCgeruStridedBatched (C++ function), 61
- hipblasCgetrf (C++ function), 220
- hipblasCgetrfBatched (C++ function), 221
- hipblasCgetrfStridedBatched (C++ function), 222
- hipblasCgetriBatched (C++ function), 227
- hipblasCgetrs (C++ function), 223
- hipblasCgetrsBatched (C++ function), 224
- hipblasCgetrsStridedBatched (C++ function), 226
- hipblasChbmv (C++ function), 63
- hipblasChbmvBatched (C++ function), 64
- hipblasChbmvStridedBatched (C++ function), 65
- hipblasChemmm (C++ function), 157
- hipblasChemmmBatched (C++ function), 158
- hipblasChemmmStridedBatched (C++ function), 159
- hipblasChemv (C++ function), 66
- hipblasChemvBatched (C++ function), 67
- hipblasChemvStridedBatched (C++ function), 68
- hipblasCher (C++ function), 69
- hipblasCher2 (C++ function), 72
- hipblasCher2Batched (C++ function), 72
- hipblasCher2k (C++ function), 134
- hipblasCher2kBatched (C++ function), 135
- hipblasCher2kStridedBatched (C++ function), 136
- hipblasCher2StridedBatched (C++ function), 73
- hipblasCherBatched (C++ function), 70
- hipblasCherk (C++ function), 128
- hipblasCherkBatched (C++ function), 129
- hipblasCherkStridedBatched (C++ function), 130
- hipblasCherkx (C++ function), 131
- hipblasCherkxBatched (C++ function), 132
- hipblasCherkxStridedBatched (C++ function), 133
- hipblasCherStridedBatched (C++ function), 71
- hipblasChpmv (C++ function), 75

`hipblasChpmvBatched` (C++ function), 75
`hipblasChpmvStridedBatched` (C++ function), 77
`hipblasChpr` (C++ function), 78
`hipblasChpr2` (C++ function), 81
`hipblasChpr2Batched` (C++ function), 82
`hipblasChpr2StridedBatched` (C++ function), 83
`hipblasChprBatched` (C++ function), 79
`hipblasChprStridedBatched` (C++ function), 80
`hipblasComplex` (C++ struct), 14
`hipblasComputeType_t` (C++ enum), 18
`hipblasComputeType_t::HIPBLAS_COMPUTE_16F` (C++ enumerator), 18
`hipblasComputeType_t::HIPBLAS_COMPUTE_16F_PEDANTIC` (C++ enumerator), 18
`hipblasComputeType_t::HIPBLAS_COMPUTE_32F` (C++ enumerator), 18
`hipblasComputeType_t::HIPBLAS_COMPUTE_32F_FAST_16F` (C++ enumerator), 18
`hipblasComputeType_t::HIPBLAS_COMPUTE_32F_FAST_16F_PEDANTIC` (C++ enumerator), 18
`hipblasComputeType_t::HIPBLAS_COMPUTE_32F_FAST_16F_PEDANTIC_2` (C++ enumerator), 18
`hipblasComputeType_t::HIPBLAS_COMPUTE_32F_PEDANTIC` (C++ enumerator), 18
`hipblasComputeType_t::HIPBLAS_COMPUTE_32I` (C++ enumerator), 19
`hipblasComputeType_t::HIPBLAS_COMPUTE_32I_PEDANTIC` (C++ enumerator), 19
`hipblasComputeType_t::HIPBLAS_COMPUTE_64F` (C++ enumerator), 18
`hipblasComputeType_t::HIPBLAS_COMPUTE_64F_PEDANTIC` (C++ enumerator), 18
`hipblasCreate` (C++ function), 237
`hipblasCrot` (C++ function), 37
`hipblasCrotBatched` (C++ function), 38
`hipblasCrotg` (C++ function), 40
`hipblasCrotgBatched` (C++ function), 40
`hipblasCrotgStridedBatched` (C++ function), 41
`hipblasCrotStridedBatched` (C++ function), 39
`hipblasCscal` (C++ function), 47
`hipblasCscalBatched` (C++ function), 47
`hipblasCscalStridedBatched` (C++ function), 48
`hipblasCspr` (C++ function), 89
`hipblasCsprBatched` (C++ function), 90
`hipblasCsprStridedBatched` (C++ function), 91
`hipblasCsrot` (C++ function), 37
`hipblasCsrotBatched` (C++ function), 38
`hipblasCsrotStridedBatched` (C++ function), 39
`hipblasCsscal` (C++ function), 47
`hipblasCsscalBatched` (C++ function), 47
`hipblasCsscalStridedBatched` (C++ function), 48
`hipblasCswap` (C++ function), 49
`hipblasCswapBatched` (C++ function), 50
`hipblasCswapStridedBatched` (C++ function), 50
`hipblasCsymm` (C++ function), 138
`hipblasCsymmBatched` (C++ function), 139
`hipblasCsymmStridedBatched` (C++ function), 140
`hipblasCsymv` (C++ function), 96
`hipblasCsymvBatched` (C++ function), 97
`hipblasCsymvStridedBatched` (C++ function), 98
`hipblasCsyr` (C++ function), 99
`hipblasCsyr2` (C++ function), 102
`hipblasCsyr2Batched` (C++ function), 102
`hipblasCsyr2k` (C++ function), 145
`hipblasCsyr2kBatched` (C++ function), 146
`hipblasCsyr2kStridedBatched` (C++ function), 148
`hipblasCsyr2StridedBatched` (C++ function), 103
`hipblasCsyrBatched` (C++ function), 100
`hipblasCsyrk` (C++ function), 142
`hipblasCsyrkBatched` (C++ function), 143
`hipblasCsyrkStridedBatched` (C++ function), 144
`hipblasCsyrkx` (C++ function), 149
`hipblasCsyrkxBatched` (C++ function), 150
`hipblasCsyrkxStridedBatched` (C++ function), 152
`hipblasCtbs` (C++ function), 101
`hipblasCtbsv` (C++ function), 105
`hipblasCtbsvBatched` (C++ function), 106
`hipblasCtbsvStridedBatched` (C++ function), 107
`hipblasCtbsvBatched` (C++ function), 110
`hipblasCtbsvStridedBatched` (C++ function), 111
`hipblasCtpmv` (C++ function), 112
`hipblasCtpmvBatched` (C++ function), 113
`hipblasCtpmvStridedBatched` (C++ function), 114
`hipblasCtpsv` (C++ function), 115
`hipblasCtpsvBatched` (C++ function), 116
`hipblasCtpsvStridedBatched` (C++ function), 117
`hipblasCtrmm` (C++ function), 160
`hipblasCtrmmBatched` (C++ function), 162
`hipblasCtrmmStridedBatched` (C++ function), 163
`hipblasCtrmv` (C++ function), 118
`hipblasCtrmvBatched` (C++ function), 118
`hipblasCtrmvStridedBatched` (C++ function), 119
`hipblasCtrsm` (C++ function), 165
`hipblasCtrsmBatched` (C++ function), 166
`hipblasCtrsmStridedBatched` (C++ function), 168
`hipblasCtrsv` (C++ function), 120
`hipblasCtrsvBatched` (C++ function), 121
`hipblasCtrsvStridedBatched` (C++ function), 122
`hipblasCtrtri` (C++ function), 169
`hipblasCtrtriBatched` (C++ function), 170
`hipblasCtrtriStridedBatched` (C++ function), 171
`hipblasDasum` (C++ function), 25
`hipblasDasumBatched` (C++ function), 26
`hipblasDasumStridedBatched` (C++ function), 26
`hipblasDatatype_t` (C++ enum), 17
`hipblasDatatype_t::HIPBLAS_C_16B` (C++ enumerator), 18

hipblasDatatype_t::HIPBLAS_C_16F (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_C_32F (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_C_32I (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_C_32U (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_C_64F (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_C_8I (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_C_8U (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_DATATYPE_INVALID (C++ *enumerator*), 18
 hipblasDatatype_t::HIPBLAS_R_16B (C++ *enumerator*), 18
 hipblasDatatype_t::HIPBLAS_R_16F (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_R_32F (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_R_32I (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_R_32U (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_R_64F (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_R_8I (C++ *enumerator*), 17
 hipblasDatatype_t::HIPBLAS_R_8U (C++ *enumerator*), 17
 hipblasDaxpy (C++ *function*), 27
 hipblasDaxpyBatched (C++ *function*), 28
 hipblasDaxpyStridedBatched (C++ *function*), 28
 hipblasDcopy (C++ *function*), 29
 hipblasDcopyBatched (C++ *function*), 30
 hipblasDcopyStridedBatched (C++ *function*), 31
 hipblasDdgemm (C++ *function*), 172
 hipblasDdgemmBatched (C++ *function*), 173
 hipblasDdgemmStridedBatched (C++ *function*), 174
 hipblasDdot (C++ *function*), 32
 hipblasDdotBatched (C++ *function*), 33
 hipblasDdotStridedBatched (C++ *function*), 34
 hipblasDestroy (C++ *function*), 237
 hipblasDgbmv (C++ *function*), 52
 hipblasDgbmvBatched (C++ *function*), 53
 hipblasDgbmvStridedBatched (C++ *function*), 54
 hipblasDgeam (C++ *function*), 153
 hipblasDgeamBatched (C++ *function*), 154
 hipblasDgeamStridedBatched (C++ *function*), 155
 hipblasDgels (C++ *function*), 232
 hipblasDgelsBatched (C++ *function*), 234
 hipblasDgelsStridedBatched (C++ *function*), 235
 hipblasDgemm (C++ *function*), 124
 hipblasDgemmBatched (C++ *function*), 125
 hipblasDgemmStridedBatched (C++ *function*), 126
 hipblasDgemv (C++ *function*), 56
 hipblasDgemvBatched (C++ *function*), 57
 hipblasDgemvStridedBatched (C++ *function*), 58
 hipblasDgeqrf (C++ *function*), 228
 hipblasDgeqrfBatched (C++ *function*), 229
 hipblasDgeqrfStridedBatched (C++ *function*), 231
 hipblasDger (C++ *function*), 59
 hipblasDgerBatched (C++ *function*), 60
 hipblasDgerStridedBatched (C++ *function*), 61
 hipblasDgetrf (C++ *function*), 220
 hipblasDgetrfBatched (C++ *function*), 221
 hipblasDgetrfStridedBatched (C++ *function*), 222
 hipblasDgetriBatched (C++ *function*), 227
 hipblasDgetrs (C++ *function*), 223
 hipblasDgetrsBatched (C++ *function*), 224
 hipblasDgetrsStridedBatched (C++ *function*), 225
 hipblasDiagType_t (C++ *enum*), 16
 hipblasDiagType_t::HIPBLAS_DIAG_NON_UNIT (C++ *enumerator*), 16
 hipblasDiagType_t::HIPBLAS_DIAG_UNIT (C++ *enumerator*), 16
 hipblasDnrm2 (C++ *function*), 35
 hipblasDnrm2Batched (C++ *function*), 36
 hipblasDnrm2StridedBatched (C++ *function*), 36
 hipblasDotBatchedEx (C++ *function*), 199
 hipblasDotcBatchedEx (C++ *function*), 204
 hipblasDotcEx (C++ *function*), 202
 hipblasDotcStridedBatchedEx (C++ *function*), 205
 hipblasDotEx (C++ *function*), 198
 hipblasDotStridedBatchedEx (C++ *function*), 200
 hipblasDoubleComplex (C++ *struct*), 14
 hipblasDrot (C++ *function*), 37
 hipblasDrotBatched (C++ *function*), 38
 hipblasDrotg (C++ *function*), 40
 hipblasDrotgBatched (C++ *function*), 40
 hipblasDrotgStridedBatched (C++ *function*), 41
 hipblasDrotm (C++ *function*), 42
 hipblasDrotmBatched (C++ *function*), 43
 hipblasDrotmg (C++ *function*), 44
 hipblasDrotmgBatched (C++ *function*), 45
 hipblasDrotmgStridedBatched (C++ *function*), 46
 hipblasDrotmStridedBatched (C++ *function*), 43
 hipblasDrotStridedBatched (C++ *function*), 39
 hipblasDsbbmv (C++ *function*), 84
 hipblasDsbbmvBatched (C++ *function*), 85
 hipblasDsbbmvStridedBatched (C++ *function*), 86
 hipblasDscal (C++ *function*), 47
 hipblasDscalBatched (C++ *function*), 47
 hipblasDscalStridedBatched (C++ *function*), 48
 hipblasDspmv (C++ *function*), 87
 hipblasDspmvBatched (C++ *function*), 87

hipblasDspmvStridedBatched (C++ function), 88
hipblasDspr (C++ function), 89
hipblasDspr2 (C++ function), 93
hipblasDspr2Batched (C++ function), 93
hipblasDspr2StridedBatched (C++ function), 94
hipblasDsprBatched (C++ function), 90
hipblasDsprStridedBatched (C++ function), 91
hipblasDswap (C++ function), 49
hipblasDswapBatched (C++ function), 50
hipblasDswapStridedBatched (C++ function), 50
hipblasDsymm (C++ function), 138
hipblasDsymmBatched (C++ function), 139
hipblasDsymmStridedBatched (C++ function), 140
hipblasDsymv (C++ function), 96
hipblasDsymvBatched (C++ function), 97
hipblasDsymvStridedBatched (C++ function), 98
hipblasDsyr (C++ function), 99
hipblasDsyr2 (C++ function), 102
hipblasDsyr2Batched (C++ function), 102
hipblasDsyr2k (C++ function), 145
hipblasDsyr2kBatched (C++ function), 146
hipblasDsyr2kStridedBatched (C++ function), 148
hipblasDsyr2StridedBatched (C++ function), 103
hipblasDsyrBatched (C++ function), 100
hipblasDsyrk (C++ function), 142
hipblasDsyrkBatched (C++ function), 143
hipblasDsyrkStridedBatched (C++ function), 144
hipblasDsyrkx (C++ function), 149
hipblasDsyrkxBatched (C++ function), 150
hipblasDsyrkxStridedBatched (C++ function), 152
hipblasDsyrStridedBatched (C++ function), 101
hipblasDtbmv (C++ function), 105
hipblasDtbmvBatched (C++ function), 106
hipblasDtbmvStridedBatched (C++ function), 107
hipblasDtbsv (C++ function), 109
hipblasDtbsvBatched (C++ function), 109
hipblasDtbsvStridedBatched (C++ function), 111
hipblasDtpmv (C++ function), 112
hipblasDtpmvBatched (C++ function), 113
hipblasDtpmvStridedBatched (C++ function), 114
hipblasDtpsv (C++ function), 115
hipblasDtpsvBatched (C++ function), 115
hipblasDtpsvStridedBatched (C++ function), 116
hipblasDtrmm (C++ function), 160
hipblasDtrmmBatched (C++ function), 161
hipblasDtrmmStridedBatched (C++ function), 163
hipblasDtrmv (C++ function), 118
hipblasDtrmvBatched (C++ function), 118
hipblasDtrmvStridedBatched (C++ function), 119
hipblasDtrsm (C++ function), 165
hipblasDtrsmBatched (C++ function), 166
hipblasDtrsmStridedBatched (C++ function), 168
hipblasDtrsv (C++ function), 120
hipblasDtrsvBatched (C++ function), 121
hipblasDtrsvStridedBatched (C++ function), 122
hipblasDtrtri (C++ function), 169
hipblasDtrtriBatched (C++ function), 170
hipblasDtrtriStridedBatched (C++ function), 171
hipblasDzasum (C++ function), 25
hipblasDzasumBatched (C++ function), 26
hipblasDzasumStridedBatched (C++ function), 26
hipblasDznrm2 (C++ function), 35
hipblasDznrm2Batched (C++ function), 36
hipblasDznrm2StridedBatched (C++ function), 36
hipblasFillMode_t (C++ enum), 16
hipblasFillMode_t::HIPBLAS_FILL_MODE_FULL
(C++ enumerator), 16
hipblasFillMode_t::HIPBLAS_FILL_MODE_LOWER
(C++ enumerator), 16
hipblasFillMode_t::HIPBLAS_FILL_MODE_UPPER
(C++ enumerator), 16
hipblasGemmAlgo_t (C++ enum), 19
hipblasGemmAlgo_t::HIPBLAS_GEMM_DEFAULT
(C++ enumerator), 19
hipblasGemmBatchedEx (C++ function), 178
hipblasGemmEx (C++ function), 175
hipblasGemmStridedBatchedEx (C++ function), 182
hipblasGetAtomsMode (C++ function), 241
hipblasGetMatrix (C++ function), 239
hipblasGetMatrixAsync (C++ function), 241
hipblasGetPointerMode (C++ function), 238
hipblasGetStream (C++ function), 238
hipblasGetVector (C++ function), 238
hipblasGetVectorAsync (C++ function), 240
hipblasHalf (C++ type), 13
hipblasHandle_t (C++ type), 13
hipblasHaxpy (C++ function), 27
hipblasHaxpyBatched (C++ function), 28
hipblasHaxpyStridedBatched (C++ function), 28
hipblasHdot (C++ function), 32
hipblasHdotBatched (C++ function), 32
hipblasHdotStridedBatched (C++ function), 34
hipblasHgemm (C++ function), 124
hipblasHgemmBatched (C++ function), 125
hipblasHgemmStridedBatched (C++ function), 126
hipblasIcamax (C++ function), 21
hipblasIcamaxBatched (C++ function), 22
hipblasIcamaxStridedBatched (C++ function), 23
hipblasIcamin (C++ function), 23
hipblasIcaminBatched (C++ function), 24
hipblasIcaminStridedBatched (C++ function), 24
hipblasIdamax (C++ function), 21
hipblasIdamaxBatched (C++ function), 22
hipblasIdamaxStridedBatched (C++ function), 22
hipblasIdamin (C++ function), 23
hipblasIdaminBatched (C++ function), 24
hipblasIdaminStridedBatched (C++ function), 24
hipblasInt8 (C++ type), 13

hipblasIsamax (C++ function), 21
 hipblasIsamaxBatched (C++ function), 22
 hipblasIsamaxStridedBatched (C++ function), 22
 hipblasIsamin (C++ function), 23
 hipblasIsaminBatched (C++ function), 24
 hipblasIsaminStridedBatched (C++ function), 24
 hipblasIzamax (C++ function), 21
 hipblasIzamaxBatched (C++ function), 22
 hipblasIzamaxStridedBatched (C++ function), 23
 hipblasIzamin (C++ function), 23
 hipblasIzaminBatched (C++ function), 24
 hipblasIzaminStridedBatched (C++ function), 24
 hipblasNrm2BatchedEx (C++ function), 208
 hipblasNrm2Ex (C++ function), 207
 hipblasNrm2StridedBatchedEx (C++ function), 209
 hipblasOperation_t (C++ enum), 15
 hipblasOperation_t::HIPBLAS_OP_C (C++ enumerator), 15
 hipblasOperation_t::HIPBLAS_OP_N (C++ enumerator), 15
 hipblasOperation_t::HIPBLAS_OP_T (C++ enumerator), 15
 hipblasPointerMode_t (C++ enum), 15
 hipblasPointerMode_t::HIPBLAS_POINTER_MODE_DEVICE (C++ enumerator), 15
 hipblasPointerMode_t::HIPBLAS_POINTER_MODE_HOST (C++ enumerator), 15
 hipblasRotBatchedEx (C++ function), 212
 hipblasRotEx (C++ function), 211
 hipblasRotStridedBatchedEx (C++ function), 214
 hipblasSasum (C++ function), 25
 hipblasSasumBatched (C++ function), 26
 hipblasSasumStridedBatched (C++ function), 26
 hipblasSaxpy (C++ function), 27
 hipblasSaxpyBatched (C++ function), 28
 hipblasSaxpyStridedBatched (C++ function), 28
 hipblasScalBatchedEx (C++ function), 217
 hipblasScalEx (C++ function), 216
 hipblasScalStridedBatchedEx (C++ function), 218
 hipblasScasum (C++ function), 25
 hipblasScasumBatched (C++ function), 26
 hipblasScasumStridedBatched (C++ function), 26
 hipblasScnrm2 (C++ function), 35
 hipblasScnrm2Batched (C++ function), 36
 hipblasScnrm2StridedBatched (C++ function), 36
 hipblasScopy (C++ function), 29
 hipblasScopyBatched (C++ function), 30
 hipblasScopyStridedBatched (C++ function), 31
 hipblasSdgm (C++ function), 172
 hipblasSdgmBatched (C++ function), 173
 hipblasSdgmStridedBatched (C++ function), 174
 hipblasSdot (C++ function), 32
 hipblasSdotBatched (C++ function), 33
 hipblasSdotStridedBatched (C++ function), 34
 hipblasSetAtomicMode (C++ function), 241
 hipblasSetMatrix (C++ function), 239
 hipblasSetMatrixAsync (C++ function), 240
 hipblasSetPointerMode (C++ function), 238
 hipblasSetStream (C++ function), 237
 hipblasSetVector (C++ function), 238
 hipblasSetVectorAsync (C++ function), 239
 hipblasSgbmv (C++ function), 52
 hipblasSgbmvBatched (C++ function), 53
 hipblasSgbmvStridedBatched (C++ function), 54
 hipblasSgeam (C++ function), 153
 hipblasSgeamBatched (C++ function), 154
 hipblasSgeamStridedBatched (C++ function), 155
 hipblasSgels (C++ function), 232
 hipblasSgelsBatched (C++ function), 234
 hipblasSgelsStridedBatched (C++ function), 235
 hipblasSgemm (C++ function), 124
 hipblasSgemmBatched (C++ function), 125
 hipblasSgemmStridedBatched (C++ function), 126
 hipblasSgemv (C++ function), 56
 hipblasSgemvBatched (C++ function), 57
 hipblasSgemvStridedBatched (C++ function), 58
 hipblasSgeqrf (C++ function), 228
 hipblasSgeqrfBatched (C++ function), 229
 hipblasSgeqrfStridedBatched (C++ function), 231
 hipblasSger (C++ function), 59
 hipblasSgerBatched (C++ function), 60
 hipblasSgerStridedBatched (C++ function), 61
 hipblasSgetrf (C++ function), 220
 hipblasSgetrfBatched (C++ function), 220
 hipblasSgetrfStridedBatched (C++ function), 222
 hipblasSgetriBatched (C++ function), 227
 hipblasSgetrs (C++ function), 223
 hipblasSgetrsBatched (C++ function), 224
 hipblasSgetrsStridedBatched (C++ function), 225
 hipblasSideMode_t (C++ enum), 16
 hipblasSideMode_t::HIPBLAS_SIDE_BOTH (C++ enumerator), 16
 hipblasSideMode_t::HIPBLAS_SIDE_LEFT (C++ enumerator), 16
 hipblasSideMode_t::HIPBLAS_SIDE_RIGHT (C++ enumerator), 16
 hipblasSnrm2 (C++ function), 35
 hipblasSnrm2Batched (C++ function), 35
 hipblasSnrm2StridedBatched (C++ function), 36
 hipblasSrot (C++ function), 37
 hipblasSrotBatched (C++ function), 38
 hipblasSrotg (C++ function), 40
 hipblasSrotgBatched (C++ function), 40
 hipblasSrotgStridedBatched (C++ function), 41
 hipblasSrotm (C++ function), 42
 hipblasSrotmBatched (C++ function), 43
 hipblasSrotmg (C++ function), 44
 hipblasSrotmgBatched (C++ function), 45

`hipblasSrotmgStridedBatched` (C++ function), 46
`hipblasSrotmStridedBatched` (C++ function), 43
`hipblasSrotStridedBatched` (C++ function), 39
`hipblasSsbmv` (C++ function), 84
`hipblasSsbmvBatched` (C++ function), 85
`hipblasSsbmvStridedBatched` (C++ function), 85
`hipblasSscal` (C++ function), 47
`hipblasSscalBatched` (C++ function), 47
`hipblasSscalStridedBatched` (C++ function), 48
`hipblasSspmv` (C++ function), 87
`hipblasSspmvBatched` (C++ function), 87
`hipblasSspmvStridedBatched` (C++ function), 88
`hipblasSspr` (C++ function), 89
`hipblasSspr2` (C++ function), 93
`hipblasSspr2Batched` (C++ function), 93
`hipblasSspr2StridedBatched` (C++ function), 94
`hipblasSsprBatched` (C++ function), 90
`hipblasSsprStridedBatched` (C++ function), 91
`hipblasSswap` (C++ function), 49
`hipblasSswapBatched` (C++ function), 50
`hipblasSswapStridedBatched` (C++ function), 50
`hipblasSsymm` (C++ function), 138
`hipblasSsymmBatched` (C++ function), 139
`hipblasSsymmStridedBatched` (C++ function), 140
`hipblasSsymv` (C++ function), 96
`hipblasSsymvBatched` (C++ function), 96
`hipblasSsymvStridedBatched` (C++ function), 97
`hipblasSsyrr` (C++ function), 99
`hipblasSsyrr2` (C++ function), 102
`hipblasSsyrr2Batched` (C++ function), 102
`hipblasSsyrr2k` (C++ function), 145
`hipblasSsyrr2kBatched` (C++ function), 146
`hipblasSsyrr2kStridedBatched` (C++ function), 147
`hipblasSsyrr2StridedBatched` (C++ function), 103
`hipblasSsyrrBatched` (C++ function), 100
`hipblasSsyrrk` (C++ function), 142
`hipblasSsyrrkBatched` (C++ function), 143
`hipblasSsyrrkStridedBatched` (C++ function), 144
`hipblasSsyrrkx` (C++ function), 149
`hipblasSsyrrkxBatched` (C++ function), 150
`hipblasSsyrrkxStridedBatched` (C++ function), 152
`hipblasSsyrrStridedBatched` (C++ function), 100
`hipblasStatus_t` (C++ enum), 14
`hipblasStatus_t::HIPBLAS_STATUS_ALLOC_FAILED` (C++ enumerator), 14
`hipblasStatus_t::HIPBLAS_STATUS_ARCH_MISMATCH` (C++ enumerator), 14
`hipblasStatus_t::HIPBLAS_STATUS_EXECUTION_FAILED` (C++ enumerator), 14
`hipblasStatus_t::HIPBLAS_STATUS_HANDLE_IS_NULL` (C++ enumerator), 15
`hipblasStatus_t::HIPBLAS_STATUS_INTERNAL_ERROR` (C++ enumerator), 14
`hipblasStatus_t::HIPBLAS_STATUS_INVALID_ENUM` (C++ enumerator), 15
`hipblasStatus_t::HIPBLAS_STATUS_INVALID_VALUE` (C++ enumerator), 14
`hipblasStatus_t::HIPBLAS_STATUS_MAPPING_ERROR` (C++ enumerator), 14
`hipblasStatus_t::HIPBLAS_STATUS_NOT_INITIALIZED` (C++ enumerator), 14
`hipblasStatus_t::HIPBLAS_STATUS_NOT_SUPPORTED` (C++ enumerator), 14
`hipblasStatus_t::HIPBLAS_STATUS_SUCCESS` (C++ enumerator), 14
`hipblasStatus_t::HIPBLAS_STATUS_UNKNOWN` (C++ enumerator), 15
`hipblasStatusToString` (C++ function), 241
`hipblasStbmv` (C++ function), 105
`hipblasStbmvBatched` (C++ function), 106
`hipblasStbmvStridedBatched` (C++ function), 107
`hipblasStbsv` (C++ function), 109
`hipblasStbsvBatched` (C++ function), 109
`hipblasStbsvStridedBatched` (C++ function), 110
`hipblasStpmv` (C++ function), 112
`hipblasStpmvBatched` (C++ function), 113
`hipblasStpmvStridedBatched` (C++ function), 113
`hipblasStpsv` (C++ function), 115
`hipblasStpsvBatched` (C++ function), 115
`hipblasStpsvStridedBatched` (C++ function), 116
`hipblasStride` (C++ type), 13
`hipblasStrmm` (C++ function), 160
`hipblasStrmmBatched` (C++ function), 161
`hipblasStrmmStridedBatched` (C++ function), 163
`hipblasStrmv` (C++ function), 118
`hipblasStrmvBatched` (C++ function), 118
`hipblasStrmvStridedBatched` (C++ function), 119
`hipblasStrsm` (C++ function), 165
`hipblasStrsmBatched` (C++ function), 166
`hipblasStrsmStridedBatched` (C++ function), 168
`hipblasStrsv` (C++ function), 120
`hipblasStrsvBatched` (C++ function), 121
`hipblasStrsvStridedBatched` (C++ function), 122
`hipblasStrtri` (C++ function), 169
`hipblasStrtriBatched` (C++ function), 170
`hipblasStrtriStridedBatched` (C++ function), 171
`hipblasTrsmBatchedEx` (C++ function), 188
`hipblasTrsmEx` (C++ function), 186
`hipblasTrsmStridedBatchedEx` (C++ function), 190
`hipblasZaxpy` (C++ function), 27
`hipblasZaxpyBatched` (C++ function), 28
`hipblasZaxpyStridedBatched` (C++ function), 29
`hipblasZcopy` (C++ function), 29
`hipblasZcopyBatched` (C++ function), 30
`hipblasZcopyStridedBatched` (C++ function), 31
`hipblasZdgm` (C++ function), 172
`hipblasZdgmBatched` (C++ function), 173

- hipblasZdggmmStridedBatched (C++ function), 174
- hipblasZdotc (C++ function), 32
- hipblasZdotcBatched (C++ function), 33
- hipblasZdotcStridedBatched (C++ function), 34
- hipblasZdotu (C++ function), 32
- hipblasZdotuBatched (C++ function), 33
- hipblasZdotuStridedBatched (C++ function), 34
- hipblasZdrot (C++ function), 37
- hipblasZdrotBatched (C++ function), 38
- hipblasZdrotStridedBatched (C++ function), 39
- hipblasZdscal (C++ function), 47
- hipblasZdscalBatched (C++ function), 48
- hipblasZdscalStridedBatched (C++ function), 48
- hipblasZgbmv (C++ function), 52
- hipblasZgbmvBatched (C++ function), 53
- hipblasZgbmvStridedBatched (C++ function), 55
- hipblasZgeam (C++ function), 153
- hipblasZgeamBatched (C++ function), 154
- hipblasZgeamStridedBatched (C++ function), 156
- hipblasZgels (C++ function), 232
- hipblasZgelsBatched (C++ function), 234
- hipblasZgelsStridedBatched (C++ function), 235
- hipblasZgemm (C++ function), 124
- hipblasZgemmBatched (C++ function), 125
- hipblasZgemmStridedBatched (C++ function), 127
- hipblasZgemv (C++ function), 56
- hipblasZgemvBatched (C++ function), 57
- hipblasZgemvStridedBatched (C++ function), 58
- hipblasZgeqrf (C++ function), 228
- hipblasZgeqrfBatched (C++ function), 230
- hipblasZgeqrfStridedBatched (C++ function), 231
- hipblasZgerc (C++ function), 59
- hipblasZgercBatched (C++ function), 60
- hipblasZgercStridedBatched (C++ function), 62
- hipblasZgeru (C++ function), 59
- hipblasZgeruBatched (C++ function), 60
- hipblasZgeruStridedBatched (C++ function), 62
- hipblasZgetrf (C++ function), 220
- hipblasZgetrfBatched (C++ function), 221
- hipblasZgetrfStridedBatched (C++ function), 222
- hipblasZgetriBatched (C++ function), 227
- hipblasZgetrs (C++ function), 223
- hipblasZgetrsBatched (C++ function), 224
- hipblasZgetrsStridedBatched (C++ function), 226
- hipblasZhbmV (C++ function), 63
- hipblasZhbmVBatched (C++ function), 64
- hipblasZhbmVStridedBatched (C++ function), 65
- hipblasZhemm (C++ function), 157
- hipblasZhemmBatched (C++ function), 158
- hipblasZhemmStridedBatched (C++ function), 159
- hipblasZhemv (C++ function), 66
- hipblasZhemvBatched (C++ function), 67
- hipblasZhemvStridedBatched (C++ function), 68
- hipblasZher (C++ function), 69
- hipblasZher2 (C++ function), 72
- hipblasZher2Batched (C++ function), 73
- hipblasZher2k (C++ function), 134
- hipblasZher2kBatched (C++ function), 135
- hipblasZher2kStridedBatched (C++ function), 136
- hipblasZher2StridedBatched (C++ function), 74
- hipblasZherBatched (C++ function), 70
- hipblasZherk (C++ function), 128
- hipblasZherkBatched (C++ function), 129
- hipblasZherkStridedBatched (C++ function), 130
- hipblasZherkx (C++ function), 131
- hipblasZherkxBatched (C++ function), 132
- hipblasZherkxStridedBatched (C++ function), 133
- hipblasZherStridedBatched (C++ function), 71
- hipblasZhpmv (C++ function), 75
- hipblasZhpmvBatched (C++ function), 76
- hipblasZhpmvStridedBatched (C++ function), 77
- hipblasZhpr (C++ function), 78
- hipblasZhpr2 (C++ function), 81
- hipblasZhpr2Batched (C++ function), 82
- hipblasZhpr2StridedBatched (C++ function), 83
- hipblasZhprBatched (C++ function), 79
- hipblasZhprStridedBatched (C++ function), 80
- hipblasZrot (C++ function), 37
- hipblasZrotBatched (C++ function), 38
- hipblasZrotg (C++ function), 40
- hipblasZrotgBatched (C++ function), 40
- hipblasZrotgStridedBatched (C++ function), 41
- hipblasZrotStridedBatched (C++ function), 39
- hipblasZscal (C++ function), 47
- hipblasZscalBatched (C++ function), 47
- hipblasZscalStridedBatched (C++ function), 48
- hipblasZspr (C++ function), 89
- hipblasZsprBatched (C++ function), 90
- hipblasZsprStridedBatched (C++ function), 91
- hipblasZswap (C++ function), 49
- hipblasZswapBatched (C++ function), 50
- hipblasZswapStridedBatched (C++ function), 50
- hipblasZsymm (C++ function), 138
- hipblasZsymmBatched (C++ function), 139
- hipblasZsymmStridedBatched (C++ function), 140
- hipblasZsymv (C++ function), 96
- hipblasZsymvBatched (C++ function), 97
- hipblasZsymvStridedBatched (C++ function), 98
- hipblasZsyr (C++ function), 99
- hipblasZsyr2 (C++ function), 102
- hipblasZsyr2Batched (C++ function), 103
- hipblasZsyr2k (C++ function), 145
- hipblasZsyr2kBatched (C++ function), 146
- hipblasZsyr2kStridedBatched (C++ function), 148
- hipblasZsyr2StridedBatched (C++ function), 104
- hipblasZsyrBatched (C++ function), 100
- hipblasZsyrk (C++ function), 142
- hipblasZsyrkBatched (C++ function), 143

`hipblasZsyrkStridedBatched` (C++ *function*), 144
`hipblasZsyrkx` (C++ *function*), 149
`hipblasZsyrkxBatched` (C++ *function*), 151
`hipblasZsyrkxStridedBatched` (C++ *function*), 152
`hipblasZsyrStridedBatched` (C++ *function*), 101
`hipblasZtbmv` (C++ *function*), 105
`hipblasZtbmvBatched` (C++ *function*), 106
`hipblasZtbmvStridedBatched` (C++ *function*), 107
`hipblasZtbsv` (C++ *function*), 109
`hipblasZtbsvBatched` (C++ *function*), 110
`hipblasZtbsvStridedBatched` (C++ *function*), 111
`hipblasZtpmv` (C++ *function*), 112
`hipblasZtpmvBatched` (C++ *function*), 113
`hipblasZtpmvStridedBatched` (C++ *function*), 114
`hipblasZtpsv` (C++ *function*), 115
`hipblasZtpsvBatched` (C++ *function*), 116
`hipblasZtpsvStridedBatched` (C++ *function*), 117
`hipblasZtrmm` (C++ *function*), 160
`hipblasZtrmmBatched` (C++ *function*), 162
`hipblasZtrmmStridedBatched` (C++ *function*), 163
`hipblasZtrmv` (C++ *function*), 118
`hipblasZtrmvBatched` (C++ *function*), 119
`hipblasZtrmvStridedBatched` (C++ *function*), 120
`hipblasZtrsm` (C++ *function*), 165
`hipblasZtrsmBatched` (C++ *function*), 166
`hipblasZtrsmStridedBatched` (C++ *function*), 168
`hipblasZtrsv` (C++ *function*), 121
`hipblasZtrsvBatched` (C++ *function*), 121
`hipblasZtrsvStridedBatched` (C++ *function*), 122
`hipblasZtrtri` (C++ *function*), 169
`hipblasZtrtriBatched` (C++ *function*), 170
`hipblasZtrtriStridedBatched` (C++ *function*), 171