

---

# **rocSOLVER Documentation**

*Release 3.13.0*

**Advanced Micro Devices**

**Sep 29, 2021**



# CONTENTS:

<b>1</b>	<b>Legal Disclaimer</b>	<b>1</b>
<b>2</b>	<b>User guide</b>	<b>3</b>
2.1	Introduction	3
2.1.1	Overview	3
2.1.2	Brief description and functionality	3
2.2	Building and installation	6
2.2.1	Prerequisites	6
2.2.2	Installing from pre-built packages	7
2.2.3	Building & installing from source	7
2.2.3.1	Using the install.sh script	7
2.2.3.2	Manual building and installation	8
2.3	Using rocSOLVER	10
2.3.1	QR factorization of a single matrix	10
2.3.2	QR factorization of a batch of matrices	12
2.3.2.1	Strided_batched version	12
2.3.2.2	Batched version	14
2.4	Tuning rocSOLVER Performance	16
2.5	Memory model	16
2.6	Multi-level Logging	16
2.6.1	Logging types	16
2.6.2	Initialization and set-up	16
2.6.3	Multiple host threads	17
2.7	rocSOLVER API	17
2.7.1	Types	17
2.7.1.1	Additional Types	18
2.7.2	Logging Functions	20
2.7.2.1	Logging set-up and tear-down	20
2.7.2.2	Profile log	20
2.7.3	LAPACK Auxiliary Functions	20
2.7.3.1	Vector and Matrix manipulations	20
2.7.3.2	Householder reflexions	22
2.7.3.3	Bidiagonal forms	27
2.7.3.4	Tridiagonal forms	31
2.7.3.5	Orthonormal matrices	34
2.7.3.6	Unitary matrices	50
2.7.4	LAPACK Functions	67
2.7.4.1	Triangular Factorizations	67
2.7.4.2	Orthogonal Factorizations	81
2.7.4.3	Problem and matrix reductions	104

2.7.4.4	Linear-systems solvers . . . . .	148
2.7.4.5	Least-squares solvers . . . . .	158
2.7.4.6	Symmetric Eigensolvers . . . . .	163
2.7.4.7	Singular Value Decomposition . . . . .	180
2.7.5	Lapack-like Functions . . . . .	189
2.7.5.1	Linear-systems solvers . . . . .	189
2.7.5.2	Triangular Factorizations . . . . .	193
2.7.6	Deprecated . . . . .	200
2.7.6.1	Types . . . . .	200
2.7.6.2	Auxiliary Functions . . . . .	202
<b>3</b>	<b>Library Design Documentation</b>	<b>205</b>
3.1	Introduction . . . . .	205
3.2	Batch rocSOLVER . . . . .	205
3.3	Clients . . . . .	205
3.3.1	Testing rocSOLVER . . . . .	205
3.3.2	Benchmarking rocSOLVER . . . . .	206
	<b>Index</b>	<b>207</b>

## LEGAL DISCLAIMER

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



## 2.1 Introduction

### 2.1.1 Overview

rocSOLVER is an implementation of LAPACK routines on top of AMD ROCm. rocSOLVER is implemented in the HIP programming language and optimized for AMD’s latest discrete GPUs.

### 2.1.2 Brief description and functionality

The rocSOLVER library is in the early stages of active development. New features are being continuously added, with new functionality documented at each release of the ROCm platform.

The following table summarizes the LAPACK functionality implemented in rocSOLVER’s latest release.

LAPACK Auxiliary Function	single	double	single complex	double complex
roc solver_lacgv			X	X
roc solver_laswp	X	X	X	X
roc solver_larfg	X	X	X	X
roc solver_larft	X	X	X	X
roc solver_larf	X	X	X	X
roc solver_larfb	X	X	X	X
roc solver_labrd	X	X	X	X
roc solver_latrd	X	X	X	X
roc solver_bdsqr	X	X	X	X
roc solver_org2r	X	X		
roc solver_orgqr	X	X		
roc solver_orgl2	X	X		
roc solver_orglq	X	X		
roc solver_org2l	X	X		
roc solver_orgql	X	X		
roc solver_orgbr	X	X		
roc solver_orgtr	X	X		
roc solver_orm2r	X	X		
roc solver_ormqr	X	X		
roc solver_orml2	X	X		
roc solver_ormlq	X	X		
roc solver_orm2l	X	X		

continues on next page

Table 1 – continued from previous page

LAPACK Auxiliary Function	single	double	single complex	double complex
<b>roc solver_ormql</b>	x	x		
<b>roc solver_ormbr</b>	x	x		
<b>roc solver_ormtr</b>	x	x		
<b>roc solver_ung2r</b>			x	x
<b>roc solver_ungqr</b>			x	x
<b>roc solver_ungl2</b>			x	x
<b>roc solver_unglq</b>			x	x
<b>roc solver_ung2l</b>			x	x
<b>roc solver_ungql</b>			x	x
<b>roc solver_ungbr</b>			x	x
<b>roc solver_ungtr</b>			x	x
<b>roc solver_unm2r</b>			x	x
<b>roc solver_unmqr</b>			x	x
<b>roc solver_unml2</b>			x	x
<b>roc solver_unmlq</b>			x	x
<b>roc solver_unm2l</b>			x	x
<b>roc solver_unmql</b>			x	x
<b>roc solver_unmbr</b>			x	x
<b>roc solver_unmtr</b>			x	x
<b>roc solver_sterf</b>	x	x		
<b>roc solver_steqr</b>	x	x	x	x

LAPACK Function	single	double	single complex	double complex
<b>roc solver_potf2</b>	x	x	x	x
roc solver_potf2_batched	x	x	x	x
roc solver_potf2_strided_batched	x	x	x	x
<b>roc solver_potrf</b>	x	x	x	x
roc solver_potrf_batched	x	x	x	x
roc solver_potrf_strided_batched	x	x	x	x
<b>roc solver_getf2</b>	x	x	x	x
roc solver_getf2_batched	x	x	x	x
roc solver_getf2_strided_batched	x	x	x	x
<b>roc solver_getrf</b>	x	x	x	x
roc solver_getrf_batched	x	x	x	x
roc solver_getrf_strided_batched	x	x	x	x
<b>roc solver_geqr2</b>	x	x	x	x
roc solver_geqr2_batched	x	x	x	x
roc solver_geqr2_strided_batched	x	x	x	x
<b>roc solver_geqrf</b>	x	x	x	x
roc solver_geqrf_batched	x	x	x	x
roc solver_geqrf_strided_batched	x	x	x	x
<b>roc solver_geql2</b>	x	x	x	x
roc solver_geql2_batched	x	x	x	x
roc solver_geql2_strided_batched	x	x	x	x
<b>roc solver_geqlf</b>	x	x	x	x
roc solver_geqlf_batched	x	x	x	x
roc solver_geqlf_strided_batched	x	x	x	x
<b>roc solver_gelq2</b>	x	x	x	x

continues on next page



Table 2 – continued from previous page

LAPACK Function	single	double	single complex	double complex
roc solver_gelq2_batched	x	x	x	x
roc solver_gelq2_strided_batched	x	x	x	x
<b>roc solver_gelqf</b>	x	x	x	x
roc solver_gelqf_batched	x	x	x	x
roc solver_gelqf_strided_batched	x	x	x	x
<b>roc solver_getrs</b>	x	x	x	x
roc solver_getrs_batched	x	x	x	x
roc solver_getrs_strided_batched	x	x	x	x
<b>roc solver_trtri</b>	x	x	x	x
roc solver_trtri_batched	x	x	x	x
roc solver_trtri_strided_batched	x	x	x	x
<b>roc solver_getri</b>	x	x	x	x
roc solver_getri_batched	x	x	x	x
roc solver_getri_strided_batched	x	x	x	x
<b>roc solver_gels</b>	x	x	x	x
roc solver_gels_batched	x	x	x	x
roc solver_gels_strided_batched	x	x	x	x
<b>roc solver_gebd2</b>	x	x	x	x
roc solver_gebd2_batched	x	x	x	x
roc solver_gebd2_strided_batched	x	x	x	x
<b>roc solver_gebrd</b>	x	x	x	x
roc solver_gebrd_batched	x	x	x	x
roc solver_gebrd_strided_batched	x	x	x	x
<b>roc solver_gesvd</b>	x	x	x	x
roc solver_gesvd_batched	x	x	x	x
roc solver_gesvd_strided_batched	x	x	x	x
<b>roc solver_sytd2</b>	x	x		
roc solver_sytd2_batched	x	x		
roc solver_sytd2_strided_batched	x	x		
<b>roc solver_sytrd</b>	x	x		
roc solver_sytrd_batched	x	x		
roc solver_sytrd_strided_batched	x	x		
<b>roc solver_hetd2</b>			x	x
roc solver_hetd2_batched			x	x
roc solver_hetd2_strided_batched			x	x
<b>roc solver_hetrd</b>			x	x
roc solver_hetrd_batched			x	x
roc solver_hetrd_strided_batched			x	x
<b>roc solver_sygs2</b>	x	x		
roc solver_sygs2_batched	x	x		
roc solver_sygs2_strided_batched	x	x		
<b>roc solver_sygst</b>	x	x		
roc solver_sygst_batched	x	x		
roc solver_sygst_strided_batched	x	x		
<b>roc solver_hegs2</b>			x	x
roc solver_hegs2_batched			x	x
roc solver_hegs2_strided_batched			x	x
<b>roc solver_hegst</b>			x	x
roc solver_hegst_batched			x	x

continues on next page

Table 2 – continued from previous page

LAPACK Function	single	double	single complex	double complex
roc solver_hegst_strided_batched			x	x
<b>roc solver_syev</b>	x	x		
roc solver_syev_batched	x	x		
roc solver_syev_strided_batched	x	x		
<b>roc solver_heev</b>			x	x
roc solver_heev_batched			x	x
roc solver_heev_strided_batched			x	x
<b>roc solver_sygv</b>	x	x		
roc solver_sygv_batched	x	x		
roc solver_sygv_strided_batched	x	x		
<b>roc solver_hegv</b>			x	x
roc solver_hegv_batched			x	x
roc solver_hegv_strided_batched			x	x

Lapack-like Function	single	double	single complex	double complex
<b>roc solver_getf2_npvt</b>	x	x	x	x
roc solver_getf2_npvt_batched	x	x	x	x
roc solver_getf2_npvt_strided_batched	x	x	x	x
<b>roc solver_getrf_npvt</b>	x	x	x	x
roc solver_getrf_npvt_batched	x	x	x	x
roc solver_getrf_npvt_strided_batched	x	x	x	x
<b>roc solver_getri_outofplace</b>	x	x	x	x
roc solver_getri_outofplace_batched	x	x	x	x
roc solver_getri_outofplace_strided_batched	x	x	x	x

## 2.2 Building and installation

### 2.2.1 Prerequisites

rocSOLVER requires a ROCm-enabled platform. For more information, see the [ROCm install guide](#).

rocSOLVER also requires a compatible version of rocBLAS installed on the system. For more information, see the [rocBLAS install guide](#).

rocBLAS and rocSOLVER are both still under active development, and it is hard to define minimal compatibility versions. For now, a good rule of thumb is to always use rocSOLVER together with the matching rocBLAS version. For example, if you want to install rocSOLVER from ROCm 3.3 release, then be sure that ROCm 3.3 rocBLAS is also installed; if you are building the rocSOLVER branch tip, then you will need to build and install rocBLAS branch tip as well.

## 2.2.2 Installing from pre-built packages

If you have added the ROCm repositories to your Linux distribution, the latest release version of rocSOLVER can be installed using a package manager. On Ubuntu, for example, use the commands:

```
sudo apt-get update
sudo apt-get install rocsolver
```

## 2.2.3 Building & installing from source

The rocSOLVER source code is hosted on GitHub. Download the code and checkout the desired branch using:

```
git clone -b <desired_branch_name> https://github.com/ROCmSoftwarePlatform/rocSOLVER.git
cd rocSOLVER
```

To build from source, some external dependencies such as CMake and Python are required. Additionally, if the library clients are to be built (by default they are not), then LAPACK and GoogleTest will be also required. (The library clients: rocsolver-test and rocsolver-bench, provide the infrastructure for testing and benchmarking rocSOLVER. For more details on the library clients see the Design Documentation here: *Clients*).

### 2.2.3.1 Using the install.sh script

It is recommended that the provided install.sh script be used to build and install rocSOLVER. The command

```
./install.sh --help
```

gives detailed information on how to use this installation script.

Next, some common use cases are listed:

```
./install.sh
```

This command builds rocSOLVER and puts the generated library files, such as headers and librocsolver.so, in the output directory: rocSOLVER/build/release/rocsolver-install. Other output files from the configuration and building process can also be found at rocSOLVER/build and rocSOLVER/build/release directories. It is assumed that all external library dependencies have been installed. It also assumes that rocBLAS library is located at: /opt/rocm/rocblas.

```
./install.sh -g
```

Use the -g flag to build in debug mode. In this case the generated library files will be located at rocSOLVER/build/debug/rocsolver-install. Other output files from the configuration and building process can also be found at rocSOLVER/build and rocSOLVER/build/debug directories

```
./install.sh --lib_dir /home/user/rocsolverlib --build_dir buildoutput
```

Use --lib\_dir and --build\_dir to change output directories. In this case, for example, the installer will put the headers and library files at /home/user/rocsolverlib, while the outputs of the configure and building process will be at rocSOLVER/buildoutput and rocSOLVER/buildoutput/release. The selected output directories must be local, otherwise the user may require sudo privileges. To install rocSOLVER system-wide, we recommend the use of the -i flag as showed below.

```
./install.sh --roblas_dir /alternative/rocblas/location
```

Use `--rocblas_dir` to change where the rocBLAS library will be looked for. In this case, for example, the installer will look for the rocBLAS library at `/alternative/rocblas/location`.

```
./install.sh -s
```

With the `-s` flag, the installer will generate a static library (`librocsolver.a`) instead.

```
./install.sh -h
```

With the `-h` flag, the installer will build rocSOLVER using the hip-clang compiler.

```
./install.sh -d
```

With the `-d` flag, the installer will first install all the external dependencies required by rocSOLVER library in `/usr/local`. This flag only needs to be used once. For subsequent invocations of `install.sh` it is not necessary to rebuild the dependencies.

```
./install.sh -c
```

With the `-c` flag, the installer will additionally build the library clients `rocsolver-bench` and `rocsolver-test`. The binaries will be located at `rocSOLVER/build/release/clients/staging`. It is assumed that all the client external dependencies have been installed.

```
./install.sh -dc
```

By combining `c` and `d` flags, the installer will also install all the external dependencies required by rocSOLVER clients. The `-d` flag only needs to be used once. For subsequent invocations of `install.sh` it is not necessary to rebuild the dependencies.

```
./install.sh -i
```

With the `-i` flag, the installer will additionally generate a pre-built rocSOLVER package and install it, using a suitable package manager, at the standard location `/opt/rocm/rocsolver`. This is the preferred approach to install rocSOLVER in a system. This way the library could be also safely removed using the package manager.

```
./install.sh -p
```

With the `-p` flag, the installer will also generate the rocSOLVER package, but it will not be installed.

```
./install.sh -i --install_dir /package/install/path
```

When generating a package, use `--install_dir` to change the directory where it will be installed. In this case, for example, rocSOLVER package will be installed at `/package/install/path`

### 2.2.3.2 Manual building and installation

Manual installation of all the external dependencies is not an easy task. Get more information on how to install each dependency at their corresponding documentation sources:

- [CMake](#) (version >3.5 is required).
- [Python](#) (version >2.7 is required. Python is installed by default in some systems like Ubuntu).
- [LAPACK](#) (which internally depends on a Fortran compiler), and
- [GoogleTest](#)

Once all dependencies are installed (including ROCm and rocBLAS), rocSOLVER can be manually built using a combination of CMake and Make commands. Using CMake options could provide more flexibility to tailor the building and installation process. Here we just provide a list of examples of common use cases (see the CMake documentation for more information on CMake options).

```
mkdir -p build/release && cd build/release
CXX=/opt/rocm/bin/hipcc cmake -DCMAKE_INSTALL_PREFIX=rocsolver-install ../../
make install
```

This is equivalent to `./install.sh`.

```
mkdir -p buildoutput/release && cd buildoutput/release
CXX=/opt/rocm/bin/hipcc cmake -DCMAKE_INSTALL_PREFIX=/home/user/rocsolverlib ../../
make install
```

This is equivalent to `./install.sh --lib_dir /home/user/rocsolverlib --build_dir buildoutput`.

```
mkdir -p build/release && cd build/release
CXX=/opt/rocm/bin/hipcc cmake -DCMAKE_INSTALL_PREFIX=rocsolver-install -Drocblas_DIR=/
↳ alternative/rocblas/location ../../
make install
```

This is equivalent to `./install.sh --rocblas_dir /alternative/rocblas/location`.

```
mkdir -p build/debug && cd build/debug
CXX=/opt/rocm/bin/hipcc cmake -DCMAKE_INSTALL_PREFIX=rocsolver-install -DCMAKE_BUILD_
↳ TYPE=Debug ../../
make install
```

This is equivalent to `./install.sh -g`.

```
mkdir -p build/release && cd build/release
CXX=/opt/rocm/bin/hipcc cmake -DCMAKE_INSTALL_PREFIX=rocsolver-install -DBUILD_SHARED_
↳ LIBS=OFF ../../
make install
```

This is equivalent to `./install.sh -s`.

```
mkdir -p build/release && cd build/release
CXX=/opt/rocm/bin/hipcc cmake -DCMAKE_INSTALL_PREFIX=rocsolver-install ../../
make install
```

This is equivalent to `./install.sh -h`.

```
mkdir -p build/release && cd build/release
CXX=/opt/rocm/bin/hipcc cmake -DCMAKE_INSTALL_PREFIX=rocsolver-install -DBUILD_CLIENTS_
↳ TESTS=ON -DBUILD_CLIENTS_BENCHMARKS=ON ../../
make install
```

This is equivalent to `./install.sh -c`.

```
mkdir -p build/release && cd build/release
CXX=/opt/rocm/bin/hipcc cmake -DCMAKE_INSTALL_PREFIX=rocsolver-install -DCPACK_SET_
↳ DESTDIR=OFF -DCPACK_PACKAGING_INSTALL_PREFIX=/opt/rocm ../../
```

(continues on next page)

(continued from previous page)

```
make install
make package
```

This is equivalent to `./install.sh -p`.

```
mkdir -p build/release && cd build/release
CXX=/opt/rocm/bin/hipcc cmake -DCMAKE_INSTALL_PREFIX=rocsolver-install -DCPACK_SET_
↳DESTDIR=OFF -DCPACK_PACKAGING_INSTALL_PREFIX=/package/install/path ../../
make install
make package
sudo dpkg -i rocsolver[-\_]*.deb
```

On an Ubuntu system, for example, this would be equivalent to `./install.sh -i --install_dir /package/install/path`.

## 2.3 Using rocSOLVER

Once installed, rocSOLVER can be used just like any other library with a C API. The header file will need to be included in the user code, and both the rocBLAS and rocSOLVER shared libraries will become link-time and run-time dependencies for the user application.

### 2.3.1 QR factorization of a single matrix

The following code snippet uses rocSOLVER to compute the QR factorization of a general m-by-n real matrix in double precision. For a full description of the used rocSOLVER routine, see the API documentation here: *roc-solver\_<type>geqrf()*.

```
////////////////////////////////////
// example.cpp source code //
////////////////////////////////////

#include <algorithm> // for std::min
#include <stdio.h>   // for size_t, printf
#include <vector>
#include <hip/hip_runtime_api.h> // for hip functions
#include <rocsolver.h> // for all the rocsolver C interfaces and type declarations

// Example: Compute the QR Factorization of a matrix on the GPU

void get_example_matrix(std::vector<double>& hA,
                       rocblas_int& M,
                       rocblas_int& N,
                       rocblas_int& lda) {
    // a *very* small example input; not a very efficient use of the API
    const double A[3][3] = { { 12, -51, 4},
                              { 6, 167, -68},
                              { -4, 24, -41} };

    M = 3;
    N = 3;
    lda = 3;
```

(continues on next page)

(continued from previous page)

```

// note: rocsolver matrices must be stored in column major format,
//       i.e. entry (i,j) should be accessed by hA[i + j*lda]
hA.resize(size_t(lda) * N);
for (size_t i = 0; i < M; ++i) {
    for (size_t j = 0; j < N; ++j) {
        // copy A (2D array) into hA (1D array, column-major)
        hA[i + j*lda] = A[i][j];
    }
}
}

// We use rocsolver_dgeqrf to factor a real M-by-N matrix, A.
// See https://rocsolver.readthedocs.io/en/latest/userguide\_api.html#\_CPPv416rocsolver\_dgeqrf14rocblas\_handleK11rocblas\_intK11rocblas\_intPdK11rocblas\_intPd
// and https://www.netlib.org/lapack/explore-html/df/dc5/group\_\_variants\_g\_ecomputational\_ga3766ea903391b5cf9008132f7440ec7b.html
int main() {
    rocblas_int M;           // rows
    rocblas_int N;           // cols
    rocblas_int lda;        // leading dimension
    std::vector<double> hA; // input matrix on CPU
    get_example_matrix(hA, M, N, lda);

    // let's print the input matrix, just to see it
    printf("A = [\n");
    for (size_t i = 0; i < M; ++i) {
        printf(" ");
        for (size_t j = 0; j < N; ++j) {
            printf("% .3f ", hA[i + j*lda]);
        }
        printf(";\n");
    }
    printf("]\n");

    // initialization
    rocblas_handle handle;
    rocblas_create_handle(&handle);

    // calculate the sizes of our arrays
    size_t size_A = size_t(lda) * N; // count of elements in matrix A
    size_t size_piv = size_t(std::min(M, N)); // count of Householder scalars

    // allocate memory on GPU
    double *dA, *dIpiv;
    hipMalloc(&dA, sizeof(double)*size_A);
    hipMalloc(&dIpiv, sizeof(double)*size_piv);

    // copy data to GPU
    hipMemcpy(dA, hA.data(), sizeof(double)*size_A, hipMemcpyHostToDevice);

    // compute the QR factorization on the GPU
    rocsolver_dgeqrf(handle, M, N, dA, lda, dIpiv);

```

(continues on next page)

(continued from previous page)

```

// copy the results back to CPU
std::vector<double> hIpiv(size_piv); // array for householder scalars on CPU
hipMemcpy(hA.data(), dA, sizeof(double)*size_A, hipMemcpyDeviceToHost);
hipMemcpy(hIpiv.data(), dIpiv, sizeof(double)*size_piv, hipMemcpyDeviceToHost);

// the results are now in hA and hIpiv
// we can print some of the results if we want to see them
printf("R = [\n");
for (size_t i = 0; i < M; ++i) {
    printf(" ");
    for (size_t j = 0; j < N; ++j) {
        printf("%.3f ", (i <= j) ? hA[i + j*lda] : 0);
    }
    printf(";\n");
}
printf("]\n");

// clean up
hipFree(dA);
hipFree(dIpiv);
rocblas_destroy_handle(handle);
}

```

The exact command used to compile the example above may vary depending on the system environment, but here is a typical example:

```

/opt/rocm/bin/hipcc -I/opt/rocm/include -c example.cpp
/opt/rocm/bin/hipcc -o example -L/opt/rocm/lib -lrocblas -lrocblas example.o

```

## 2.3.2 QR factorization of a batch of matrices

One of the advantages of using GPUs is the ability to execute in parallel many operations of the same type but on different data sets. Based on this idea, rocSOLVER and rocBLAS provide a *batch* version of most of their routines. These batch versions allow the user to execute the same operation on a set of different matrices and/or vectors with a single library call. For more details on the approach to batch functionality followed in rocSOLVER, see *Batch rocSOLVER*.

### 2.3.2.1 Strided\_batched version

The following code snippet uses rocSOLVER to compute the QR factorization of a series of general m-by-n real matrices in double precision. The matrices must be stored in contiguous memory locations on the GPU, and are accessed by a pointer to the first matrix and a stride value that gives the separation between one matrix and the next one. For a full description of the used rocSOLVER routine, see the API documentation here: *rocblas\_<type>geqrf\_strided\_batched()*.

```

////////////////////////////////////
// example_strided.cpp source code //
////////////////////////////////////

#include <iostream>

```

(continues on next page)



(continued from previous page)

```

#include <stdlib.h>
#include <vector>
#include <rocsolver.h> // this includes all the rocsolver C interfaces and type
↳ declarations

using namespace std;

int main() {
    rocblas_int M;
    rocblas_int N;
    rocblas_int lda;
    rocblas_int batch_count; // number of matrices to factorize in the batch

    // initialize batch_count, M, N and lda with desired values
    // here====>>

    rocblas_handle handle;
    rocblas_create_handle(&handle); // this creates the rocblas handle

    rocblas_int strideA = lda*N; // separation between two matrices in
↳ memory
    size_t size_A = size_t(strideA)*batch_count; // size of the array that holds the
↳ matrices
    rocblas_int strideP = min(M,N); // separation between two sets of
↳ Householder scalars
    size_t size_piv = size_t(strideP)*batch_count; // size of the array that will have
↳ the Householder scalars

    vector<double> hA(size_A); // creates array for matrices in CPU
    vector<double> hIpiv(size_piv); // creates array for householder scalars in CPU

    double *dA, *dIpiv;
    hipMalloc(&dA, sizeof(double)*size_A); // allocates memory for matrices in GPU
    hipMalloc(&dIpiv, sizeof(double)*size_piv); // allocates memory for scalars in GPU

    // initialize all matrices (array hA) with input data
    // here====>>
    // ( matrices must be stored in column major format, i.e. entry (i,j)
    // of the k-th matrix in the batch should be accessed by hA[i + j*lda + k*strideA]
↳ )

    hipMemcpy(dA, hA.data(), sizeof(double)*size_A, hipMemcpyHostToDevice); //
↳ copy data to GPU

    rocsolver_dgeqrf_strided_batched(handle, M, N, dA,
    lda, strideA, dIpiv, strideP, batch_count); //
↳ compute the QR factorization on the GPU

    hipMemcpy(hA.data(), dA, sizeof(double)*size_A, hipMemcpyDeviceToHost); //
↳ copy the results back to CPU
    hipMemcpy(hIpiv.data(), dIpiv, sizeof(double)*size_piv, hipMemcpyDeviceToHost);

```

(continues on next page)

(continued from previous page)

```

// do something with the results in hA and hIpiv
// here====>

hipFree(dA);           // de-allocate GPU memory
hipFree(dIpiv);
rocblas_destroy_handle(handle); // destroy handle

return 0;
}

```

### 2.3.2.2 Batched version

The following code snippet uses rocSOLVER to compute the QR factorization of a series of general m-by-n real matrices in double precision. The matrices do not need to be in contiguous memory locations on the GPU, and will be accessed by an array of pointers. For a full description of the used rocSOLVER routine, see the API documentation here: `roc solver_<type>geqrf_batched()`.

```

////////////////////////////////////
// example_batched.cpp source code //
////////////////////////////////////

#include <iostream>
#include <stdlib.h>
#include <vector>
#include <roc solver.h> // this includes all the roc solver C interfaces and type_
↳ declarations

using namespace std;

int main() {
    rocblas_int M;
    rocblas_int N;
    rocblas_int lda;
    rocblas_int batch_count; // number of matrices to factorize in the batch

    // initialize batch_count, M, N and lda with desired values
    // here====>

    rocblas_handle handle;
    rocblas_create_handle(&handle); // this creates the rocblas handle

    size_t size_A = size_t(lda)*N; // size of the array that holds one_
↳ matrix
    rocblas_int strideP = min(M,N); // separation between two sets of_
↳ Householder scalars
    size_t size_piv = size_t(strideP)*batch_count; // size of the array that will have_
↳ the Householder scalars

    vector<double> hIpiv(size_piv); // creates array for householder scalars in_
↳ CPU
    vector<double> hA[batch_count]; // creates array on the CPU of pointers to_
↳ the CPU

```

(continues on next page)

(continued from previous page)

```

    for(int b=0; b < batch_count; ++b) {
        hA[b] = vector<double>(size_A);    // each pointer will point to a matrix of
↪the batch on the CPU
    }

    double* A[batch_count];                // creates array on the CPU of pointers
↪to the GPU
    for (int b = 0; b < batch_count; ++b)
        hipMalloc(&A[b], sizeof(double)*size_A); // each pointer will point to a matrix
↪of the batch on the GPU

    double **dA, *dIpivot;
    hipMalloc(&dA, sizeof(double*) * size_A); // array on the GPU of pointers on the
↪GPU
    hipMalloc(&dIpivot, sizeof(double)*size_piv); // allocates memory for scalars in GPU

    // initialize all matrices (arrays hA[k]) with input data
    // here====>>
    // ( matrices must be stored in column major format, i.e. entry (i,j)
    //   of the k-th matrix in the batch should be accessed by hA[k][i + j*lda] )

    for(int b=0; b < batch_count; ++b)
        hipMemcpy(A[b], hA[b].data(), sizeof(double)*size_A, hipMemcpyHostToDevice); //
↪/ copy data to GPU
    hipMemcpy(dA, A, sizeof(double*) * batch_count, hipMemcpyHostToDevice); //
↪/ copy pointers to GPU

    rocsolver_dgeqrf_batched(handle, M, N, dA,
↪                               lda, dIpivot, strideP, batch_count); // compute the QR
↪factorization on the GPU

    for(int b=0;b<batch_count;b++)
        hipMemcpy(hA[b].data(), A[b], sizeof(double) * size_A, hipMemcpyDeviceToHost); //
↪ copy the results back
    hipMemcpy(hIpivot.data(), dIpivot, sizeof(double)*size_piv, hipMemcpyDeviceToHost);

    // do something with the results in hA and hIpivot
    // here====>>

    for(int b=0;b<batch_count;++b)
        hipFree(A[b]);
    hipFree(dA); // de-allocate GPU memory
    hipFree(dIpivot);
    rocblas_destroy_handle(handle); // destroy handle

    return 0;
}

```

## 2.4 Tuning rocSOLVER Performance

More to come later...

## 2.5 Memory model

More to come later...

## 2.6 Multi-level Logging

Similar to rocBLAS, rocSOLVER provides logging facilities that can be used to output information on rocSOLVER function calls. Three types of logging are supported: trace logging, bench logging, and profile logging.

Note that performance will degrade when logging is enabled.

### 2.6.1 Logging types

Trace logging outputs a line each time an internal rocSOLVER or rocBLAS routine is called, outputting the function name and the values of its arguments (excluding stride arguments). The maximum depth of nested function calls that can appear in the log is specified by the user.

Bench logging outputs a line each time a public rocSOLVER routine is called (excluding auxiliary library functions), outputting a line that can be used with the executable `rocsolver-bench` to call the function with the same size arguments.

Profile logging, upon calling `rocsolver_log_write_profile` or `rocsolver_log_flush_profile`, or terminating the logging session using `rocsolver_log_end`, will output statistics on each called internal rocSOLVER and rocBLAS routine. These include the number of times each function was called, the total program runtime occupied by the function, and the total program runtime occupied by its nested function calls. As with trace logging, the maximum depth of nested output is specified by the user. Note that, when profile logging is enabled, the stream will be synchronized after every internal function call.

### 2.6.2 Initialization and set-up

In order to use rocSOLVER's logging facilities, the user must first call `rocsolver_log_begin` in order to allocate the internal data structures used for logging and begin the logging session. The user may then specify a layer mode and max level depth, either programmatically using `rocsolver_log_set_layer_mode`, `rocsolver_log_set_max_levels`, or by setting the corresponding environment variables.

The layer mode specifies which logging type(s) are activated, and can be `rocblas_layer_mode_none`, `rocblas_layer_mode_log_trace`, `rocblas_layer_mode_log_bench`, `rocblas_layer_mode_log_profile`, or a bitwise combination of these. The max level depth specifies the default maximum depth of nested function calls that may appear in the trace and profile logging.

Both the default layer mode and max level depth can be specified using environment variables.

- `ROCSOLVER_LAYER`
- `ROCSOLVER_LEVELS`

If these variables are not set, the layer mode will default to `rocblas_layer_mode_none` and the max level depth will default to 1. These defaults can be restored by calling the function `rocsolver_log_restore_defaults`.

`ROCSOLVER_LAYER` is a bitwise OR of zero or more bit masks as follows:

- If `ROCSOLVER_LAYER` is not set, then there is no logging
- If  $(\text{ROCSOLVER\_LAYER} \ \& \ 1) \neq 0$ , then there is trace logging
- If  $(\text{ROCSOLVER\_LAYER} \ \& \ 2) \neq 0$ , then there is bench logging
- If  $(\text{ROCSOLVER\_LAYER} \ \& \ 4) \neq 0$ , then there is profile logging

Three environment variables can set the full path name for a log file:

- `ROCSOLVER_LOG_TRACE_PATH` sets the full path name for trace logging
- `ROCSOLVER_LOG_BENCH_PATH` sets the full path name for bench logging
- `ROCSOLVER_LOG_PROFILE_PATH` sets the full path name for profile logging

If one of these environment variables is not set, then `ROCSOLVER_LOG_PATH` sets the full path for the corresponding logging, if it is set. If neither the above nor `ROCSOLVER_LOG_PATH` are set, then the corresponding logging output is streamed to standard error.

The results of profile logging, if enabled, can be printed using `rocsolver_log_write_profile` or `rocsolver_log_flush_profile`. Once logging facilities are no longer required (e.g. at program termination), the user must call `rocsolver_log_end` to free the data structures used for logging. If the profile log has not been flushed beforehand, then `rocsolver_log_end` will also output the results of profile logging.

### 2.6.3 Multiple host threads

The logging facilities for rocSOLVER assume that each `rocblas_handle` is associated with at most one host thread. When using rocSOLVER's multi-level logging setup, it is recommended to create a separate `rocblas_handle` for each host thread.

The `rocsolver_log_*` functions are not thread-safe. Calling a log function while any rocSOLVER routine is executing on another host thread will result in undefined behaviour. Once enabled, logging data collection is thread-safe. However, note that trace logging will likely result in garbled trace trees if rocSOLVER routines are called from multiple host threads.

## 2.7 rocSOLVER API

This section provides details of the rocSOLVER library API.

### 2.7.1 Types

rocSOLVER uses types and enumerations defined by the rocBLAS API. For more information, see the [rocBLAS types](#).

### 2.7.1.1 Additional Types

These are types that extend the rocBLAS API.

#### **rocblas\_direct**

enum **rocblas\_direct**

Used to specify the order in which multiple elementary matrices are applied together.

*Values:*

enumerator **rocblas\_forward\_direction**

Elementary matrices applied from the right.

enumerator **rocblas\_backward\_direction**

Elementary matrices applied from the left.

#### **rocblas\_storev**

enum **rocblas\_storev**

Used to specify how householder vectors are stored in a matrix of vectors.

*Values:*

enumerator **rocblas\_column\_wise**

Householder vectors are stored in the columns of a matrix.

enumerator **rocblas\_row\_wise**

Householder vectors are stored in the rows of a matrix.

#### **rocblas\_svect**

enum **rocblas\_svect**

Used to specify how the singular vectors are to be computed and stored.

*Values:*

enumerator **rocblas\_svect\_all**

The entire associated orthogonal/unitary matrix is computed.

enumerator **rocblas\_svect\_singular**

Only the singular vectors are computed and stored in output array.

enumerator **rocblas\_svect\_overwrite**

Only the singular vectors are computed and overwrite the input matrix.

enumerator **rocblas\_svect\_none**

No singular vectors are computed.

## rocblas\_evect

enum **rocblas\_evect**

Used to specify how the eigenvectors are to be computed.

*Values:*

enumerator **rocblas\_evect\_original**

Compute eigenvectors for the original symmetric/Hermitian matrix.

enumerator **rocblas\_evect\_tridiagonal**

Compute eigenvectors for the symmetric tridiagonal matrix.

enumerator **rocblas\_evect\_none**

No eigenvectors are computed.

## rocblas\_workmode

enum **rocblas\_workmode**

Used to enable the use of fast algorithms (with out-of-place computations) in some of the routines.

*Values:*

enumerator **rocblas\_outofplace**

Out-of-place computations are allowed; this requires enough free memory.

enumerator **rocblas\_inplace**

When not enough memory, this forces in-place computations

## rocblas\_iform

enum **rocblas\_iform**

Used to specify the form of the generalized eigenproblem.

*Values:*

enumerator **rocblas\_iform\_ax**

The problem is  $A*x = \lambda*B*x$ .

enumerator **rocblas\_iform\_abx**

The problem is  $A*B*x = \lambda*x$ .

enumerator **rocblas\_iform\_bax**

The problem is  $B*A*x = \lambda*x$ .

## 2.7.2 Logging Functions

These are functions that enable and control rocSOLVER's *Multi-level Logging* capabilities.

### 2.7.2.1 Logging set-up and tear-down

#### **roc solver\_log\_<function>()**

rocblas\_status **roc solver\_log\_begin**(void)

rocblas\_status **roc solver\_log\_end**(void)

rocblas\_status **roc solver\_log\_set\_layer\_mode**(const rocblas\_layer\_mode\_flags layer\_mode)

rocblas\_status **roc solver\_log\_set\_max\_levels**(const rocblas\_int max\_levels)

rocblas\_status **roc solver\_log\_restore\_defaults**(void)

LOG\_RESTORE\_DEFAULTS restores the default values of the rocSOLVER multi-level logging environment.

This function sets the logging mode and maximum trace log depth to their default values (no logging and one level depth).

### 2.7.2.2 Profile log

#### **roc solver\_log\_<write/flush>\_profile()**

rocblas\_status **roc solver\_log\_write\_profile**(void)

rocblas\_status **roc solver\_log\_flush\_profile**(void)

LOG\_FLUSH\_PROFILE prints the profile logging results and clears the profile record.

## 2.7.3 LAPACK Auxiliary Functions

These are functions that support more advanced LAPACK routines.

### 2.7.3.1 Vector and Matrix manipulations

#### **roc solver\_<type>lacgv()**

rocblas\_status **roc solver\_zlacgv**(rocblas\_handle handle, const rocblas\_int n, rocblas\_double\_complex \*x, const rocblas\_int incx)

rocblas\_status **roc solver\_clacgv**(rocblas\_handle handle, const rocblas\_int n, rocblas\_float\_complex \*x, const rocblas\_int incx)

LACGV conjugates the complex vector x.

It conjugates the n entries of a complex vector x with increment incx.



**Parameters**

- **handle** – [in] rocblas\_handle
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of entries of the vector  $x$ .
- **x** – [inout] pointer to type. Array on the GPU of size at least  $n$ .  
On input it is the vector  $x$ , on output it is overwritten with vector  $\text{conjg}(x)$ .
- **incx** – [in] rocblas\_int.  $\text{incx} \neq 0$ .  
The increment between consecutive elements of  $x$ . If  $\text{incx}$  is negative, the elements of  $x$  are indexed in reverse order.

**roc solver\_<type>laswp()**

rocblas\_status **roc solver\_zlaswp**(rocblas\_handle handle, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_int k1, const rocblas\_int k2, const rocblas\_int \*ipiv, const rocblas\_int incx)

rocblas\_status **roc solver\_claswp**(rocblas\_handle handle, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_int k1, const rocblas\_int k2, const rocblas\_int \*ipiv, const rocblas\_int incx)

rocblas\_status **roc solver\_dlaswp**(rocblas\_handle handle, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_int k1, const rocblas\_int k2, const rocblas\_int \*ipiv, const rocblas\_int incx)

rocblas\_status **roc solver\_slaswp**(rocblas\_handle handle, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_int k1, const rocblas\_int k2, const rocblas\_int \*ipiv, const rocblas\_int incx)

LASWP performs a series of row interchanges on the matrix  $A$ .

It interchanges row  $I$  with row  $\text{IPIV}[k1 + (I - k1) * \text{abs}(\text{incx})]$ , for each of rows  $K1$  through  $K2$  of  $A$ .  $k1$  and  $k2$  are 1-based indices.

**Parameters**

- **handle** – [in] rocblas\_handle
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix  $A$ .
- **A** – [inout] pointer to type. Array on the GPU of dimension  $\text{lda} * n$ .  
On entry, the matrix of column dimension  $n$  to which the row interchanges will be applied.  
On exit, the permuted matrix.
- **lda** – [in] rocblas\_int.  $\text{lda} > 0$ .  
The leading dimension of the array  $A$ .
- **k1** – [in] rocblas\_int.  $k1 > 0$ .  
The first element of  $\text{IPIV}$  for which a row interchange will be done. This is a 1-based index.

- **k2** – [in] rocblas\_int.  $k2 > k1 > 0$ .

$(K2-K1+1)$  is the number of elements of IPIV for which a row interchange will be done. This is a 1-based index.

- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU of dimension at least  $k1 + (k2 - k1) * \text{abs}(\text{incx})$ .

The vector of pivot indices. Only the elements in positions  $k1$  through  $(k1 + (k2 - k1) * \text{abs}(\text{incx}))$  of IPIV are accessed. Elements of ipiv are considered 1-based.

- **incx** – [in] rocblas\_int.  $\text{incx} \neq 0$ .

The increment between successive values of IPIV. If IPIV is negative, the pivots are applied in reverse order.

### 2.7.3.2 Householder reflexions

#### roc solver\_<type>larfg()

rocblas\_status **roc solver\_zlarfg**(rocblas\_handle handle, const rocblas\_int n, rocblas\_double\_complex \*alpha, rocblas\_double\_complex \*x, const rocblas\_int incx, rocblas\_double\_complex \*tau)

rocblas\_status **roc solver\_clarfg**(rocblas\_handle handle, const rocblas\_int n, rocblas\_float\_complex \*alpha, rocblas\_float\_complex \*x, const rocblas\_int incx, rocblas\_float\_complex \*tau)

rocblas\_status **roc solver\_dlarfg**(rocblas\_handle handle, const rocblas\_int n, double \*alpha, double \*x, const rocblas\_int incx, double \*tau)

rocblas\_status **roc solver\_slarfg**(rocblas\_handle handle, const rocblas\_int n, float \*alpha, float \*x, const rocblas\_int incx, float \*tau)

LARFG generates an orthogonal Householder reflector H of order n.

Householder reflector H is such that

$$H * \begin{bmatrix} \alpha \\ x \end{bmatrix} = \begin{bmatrix} \beta \\ 0 \end{bmatrix}$$

where x is an n-1 vector and alpha and beta are scalars. Matrix H can be generated as

$$H = I - \tau * \begin{bmatrix} 1 \\ v \end{bmatrix} * \begin{bmatrix} 1 & v' \end{bmatrix}$$

with v an n-1 vector and tau a scalar.

#### Parameters

- **handle** – [in] rocblas\_handle

- **n** – [in] rocblas\_int.  $n \geq 0$ .

The order (size) of reflector H.

- **alpha** – [inout] pointer to type. A scalar on the GPU.

On input the scalar alpha, on output it is overwritten with beta.

- **x** – [inout] pointer to type. Array on the GPU of size at least n-1.  
On input it is the vector x, on output it is overwritten with vector v.
- **incx** – [in] rocblas\_int. incx > 0.  
The increment between consecutive elements of x.
- **tau** – [out] pointer to type. A scalar on the GPU.  
The scalar tau.

### roc solver\_<type>larft()

rocblas\_status **roc solver\_zlarft**(rocblas\_handle handle, const *rocblas\_direct* direct, const *rocblas\_storev* storev, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*V, const rocblas\_int ldv, rocblas\_double\_complex \*tau, rocblas\_double\_complex \*T, const rocblas\_int ldt)

rocblas\_status **roc solver\_clarft**(rocblas\_handle handle, const *rocblas\_direct* direct, const *rocblas\_storev* storev, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*V, const rocblas\_int ldv, rocblas\_float\_complex \*tau, rocblas\_float\_complex \*T, const rocblas\_int ldt)

rocblas\_status **roc solver\_dlarft**(rocblas\_handle handle, const *rocblas\_direct* direct, const *rocblas\_storev* storev, const rocblas\_int n, const rocblas\_int k, double \*V, const rocblas\_int ldv, double \*tau, double \*T, const rocblas\_int ldt)

rocblas\_status **roc solver\_slarft**(rocblas\_handle handle, const *rocblas\_direct* direct, const *rocblas\_storev* storev, const rocblas\_int n, const rocblas\_int k, float \*V, const rocblas\_int ldv, float \*tau, float \*T, const rocblas\_int ldt)

LARFT Generates the triangular factor T of a block reflector H of order n.

The block reflector H is defined as the product of k Householder matrices as

$$H = H(1) * H(2) * \dots * H(k) \quad (\text{forward direction}), \text{ or}$$

$$H = H(k) * \dots * H(2) * H(1) \quad (\text{backward direction})$$

depending on the value of direct.

The triangular matrix T is upper triangular in forward direction and lower triangular in backward direction. If storev is column-wise, then

$$H = I - V * T * V'$$

where the i-th column of matrix V contains the Householder vector associated to H(i). If storev is row-wise, then

$$H = I - V' * T * V$$

where the i-th row of matrix V contains the Householder vector associated to H(i).

#### Parameters

- **handle** – [in] rocblas\_handle.
- **direct** – [in] *rocblas\_direct* .  
Specifies the direction in which the Householder matrices are applied.

- **storev** – [in] *rocblas\_storev* .  
Specifies how the Householder vectors are stored in matrix V.
- **n** – [in] *rocblas\_int*.  $n \geq 0$ .  
The order (size) of the block reflector.
- **k** – [in] *rocblas\_int*.  $k \geq 1$ .  
The number of Householder matrices.
- **V** – [in] pointer to type. Array on the GPU of size  $ldv*k$  if column-wise, or  $ldv*n$  if row-wise.  
The matrix of Householder vectors.
- **ldv** – [in] *rocblas\_int*.  $ldv \geq n$  if column-wise, or  $ldv \geq k$  if row-wise.  
Leading dimension of V.
- **tau** – [in] pointer to type. Array of  $k$  scalars on the GPU.  
The vector of all the scalars associated to the Householder matrices.
- **T** – [out] pointer to type. Array on the GPU of dimension  $ldt*k$ .  
The triangular factor. T is upper triangular if forward operation, otherwise it is lower triangular. The rest of the array is not used.
- **ldt** – [in] *rocblas\_int*.  $ldt \geq k$ .  
The leading dimension of T.

### roc solver\_<type>larf()

*rocblas\_status* **roc solver\_zlarf**(*rocblas\_handle* handle, const *rocblas\_side* side, const *rocblas\_int* m, const *rocblas\_int* n, *rocblas\_double\_complex* \*x, const *rocblas\_int* incx, const *rocblas\_double\_complex* \*alpha, *rocblas\_double\_complex* \*A, const *rocblas\_int* lda)

*rocblas\_status* **roc solver\_clarf**(*rocblas\_handle* handle, const *rocblas\_side* side, const *rocblas\_int* m, const *rocblas\_int* n, *rocblas\_float\_complex* \*x, const *rocblas\_int* incx, const *rocblas\_float\_complex* \*alpha, *rocblas\_float\_complex* \*A, const *rocblas\_int* lda)

*rocblas\_status* **roc solver\_dlarf**(*rocblas\_handle* handle, const *rocblas\_side* side, const *rocblas\_int* m, const *rocblas\_int* n, *double* \*x, const *rocblas\_int* incx, const *double* \*alpha, *double* \*A, const *rocblas\_int* lda)

*rocblas\_status* **roc solver\_slarf**(*rocblas\_handle* handle, const *rocblas\_side* side, const *rocblas\_int* m, const *rocblas\_int* n, *float* \*x, const *rocblas\_int* incx, const *float* \*alpha, *float* \*A, const *rocblas\_int* lda)

LARF applies a Householder reflector H to a general matrix A.

The Householder reflector H, of order m (or n), is to be applied to a m-by-n matrix A from the left (or the right). H is given by

$$H = I - \alpha * x * x'$$

where alpha is a scalar and x a Householder vector. H is never actually computed.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
If side = rocblas\_side\_left, then compute  $H*A$  If side = rocblas\_side\_right, then compute  $A*H$
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of A.
- **x** – [in] pointer to type. Array on the GPU of size at least  $(1 + (m-1)*abs(incx))$  if left side, or at least  $(1 + (n-1)*abs(incx))$  if right side.  
The Householder vector x.
- **incx** – [in] rocblas\_int.  $incx \neq 0$ .  
Increment between to consecutive elements of x. If  $incx < 0$ , the elements of x are used in reverse order.
- **alpha** – [in] pointer to type. A scalar on the GPU.  
If  $alpha = 0$ , then  $H = I$  (A will remain the same, x is never used)
- **A** – [inout] pointer to type. Array on the GPU of size  $lda*n$ .  
On input, the matrix A. On output it is overwritten with  $H*A$  (or  $A*H$ ).
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Leading dimension of A.

**roc solver\_<type>larfb()**

rocblas\_status **roc solver\_zlarfb**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const *rocblas\_direct* direct, const *rocblas\_storev* storev, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*V, const rocblas\_int ldv, rocblas\_double\_complex \*T, const rocblas\_int ldt, rocblas\_double\_complex \*A, const rocblas\_int lda)

rocblas\_status **roc solver\_clarfb**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const *rocblas\_direct* direct, const *rocblas\_storev* storev, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*V, const rocblas\_int ldv, rocblas\_float\_complex \*T, const rocblas\_int ldt, rocblas\_float\_complex \*A, const rocblas\_int lda)

rocblas\_status **roc solver\_dlarfb**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const *rocblas\_direct* direct, const *rocblas\_storev* storev, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*V, const rocblas\_int ldv, double \*T, const rocblas\_int ldt, double \*A, const rocblas\_int lda)

rocblas\_status **rocblas\_slarfb**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const *rocblas\_direct* direct, const *rocblas\_storev* storev, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*V, const rocblas\_int ldv, float \*T, const rocblas\_int ldt, float \*A, const rocblas\_int lda)

LARFB applies a block reflector H to a general m-by-n matrix A.

The block reflector H is applied in one of the following forms, depending on the values of side and trans:

H * A	(No transpose <b>from the left</b> )
H' * A	(Transpose or conjugate transpose <b>from the left</b> )
A * H	(No transpose <b>from the right</b> ), <b>and</b>
A * H'	(Transpose or conjugate transpose <b>from the right</b> )

The block reflector H is defined as the product of k Householder matrices as

$H = H(1) * H(2) * \dots * H(k)$	(forward direction), <b>or</b>
$H = H(k) * \dots * H(2) * H(1)$	(backward direction)

depending on the value of direct. H is never stored. It is calculated as

$H = I - V * T * V'$
----------------------

where the i-th column of matrix V contains the Householder vector associated with H(i), if storev is column-wise; or

$H = I - V' * T * V$
----------------------

where the i-th row of matrix V contains the Householder vector associated with H(i), if storev is row-wise. T is the associated triangular factor as computed by LARFT.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply H.
- **trans** – [in] rocblas\_operation.  
Specifies whether the block reflector or its transpose/conjugate transpose is to be applied.
- **direct** – [in] *rocblas\_direct* .  
Specifies the direction in which the Householder matrices were to be applied to generate H.
- **storev** – [in] *rocblas\_storev* .  
Specifies how the Householder vectors are stored in matrix V.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix A.
- **k** – [in] rocblas\_int.  $k \geq 1$ .  
The number of Householder matrices.

- **V** – [in] pointer to type. Array on the GPU of size  $ldv*k$  if column-wise,  $ldv*n$  if row-wise and applying from the right, or  $ldv*m$  if row-wise and applying from the left.

The matrix of Householder vectors.

- **ldv** – [in] rocblas\_int.  $ldv \geq k$  if row-wise,  $ldv \geq m$  if column-wise and applying from the left, or  $ldv \geq n$  if column-wise and applying from the right.

Leading dimension of V.

- **T** – [in] pointer to type. Array on the GPU of dimension  $ldt*k$ .

The triangular factor of the block reflector.

- **ldt** – [in] rocblas\_int.  $ldt \geq k$ .

The leading dimension of T.

- **A** – [inout] pointer to type. Array on the GPU of size  $lda*n$ .

On input, the matrix A. On output it is overwritten with  $H*A$ ,  $A*H$ ,  $H'*A$ , or  $A*H'$ .

- **lda** – [in] rocblas\_int.  $lda \geq m$ .

Leading dimension of A.

### 2.7.3.3 Bidiagonal forms

#### roc solver\_<type>lalbrd()

rocblas\_status **roc solver\_zlalbrd**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, double \*D, double \*E, rocblas\_double\_complex \*tauq, rocblas\_double\_complex \*taup, rocblas\_double\_complex \*X, const rocblas\_int ldx, rocblas\_double\_complex \*Y, const rocblas\_int ldy)

rocblas\_status **roc solver\_clalbrd**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, float \*D, float \*E, rocblas\_float\_complex \*tauq, rocblas\_float\_complex \*taup, rocblas\_float\_complex \*X, const rocblas\_int ldx, rocblas\_float\_complex \*Y, const rocblas\_int ldy)

rocblas\_status **roc solver\_dlalbrd**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*D, double \*E, double \*tauq, double \*taup, double \*X, const rocblas\_int ldx, double \*Y, const rocblas\_int ldy)

rocblas\_status **roc solver\_slalbrd**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*D, float \*E, float \*tauq, float \*taup, float \*X, const rocblas\_int ldx, float \*Y, const rocblas\_int ldy)

LABRD computes the bidiagonal form of the first k rows and columns of a general m-by-n matrix A, as well as the matrices X and Y needed to reduce the remaining part of A.

The bidiagonal form is given by:

$$B = Q' * A * P$$

where B is upper bidiagonal if  $m \geq n$  and lower bidiagonal if  $m < n$ , and Q and P are orthogonal/unitary matrices represented as the product of Householder matrices

$$Q = H(1) * H(2) * \dots * H(k) \text{ and } P = G(1) * G(2) * \dots * G(k-1), \text{ if } m \geq n, \text{ or}$$

$$Q = H(1) * H(2) * \dots * H(k-1) \text{ and } P = G(1) * G(2) * \dots * G(k), \text{ if } m < n$$

Each Householder matrix H(i) and G(i) is given by

$$H(i) = I - \tau_{\text{auq}}[i-1] * v(i) * v(i)', \text{ and}$$

$$G(i) = I - \tau_{\text{aup}}[i-1] * u(i) * u(i)'$$

If  $m \geq n$ , the first  $i-1$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i] = 1$ ; while the first  $i$  elements of the Householder vector  $u(i)$  are zero, and  $u(i)[i+1] = 1$ . If  $m < n$ , the first  $i$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i+1] = 1$ ; while the first  $i-1$  elements of the Householder vector  $u(i)$  are zero, and  $u(i)[i] = 1$ .

The unreduced part of the matrix A can be updated using a block update:

$$A = A - V * Y' - X * U'$$

where V is an  $m$ -by- $k$  matrix and U is an  $n$ -by- $k$  formed using the vectors  $v$  and  $u$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **k** – [in] rocblas\_int.  $\min(m,n) \geq k \geq 0$ .  
The number of leading rows and columns of the matrix A to be reduced.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $\text{lda} * n$ .  
On entry, the  $m$ -by- $n$  matrix to be factored. On exit, the elements on the diagonal and superdiagonal (if  $m \geq n$ ), or subdiagonal (if  $m < n$ ) contain the bidiagonal form B. If  $m \geq n$ , the elements below the diagonal are the  $m - i$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, n$ , and the elements above the superdiagonal are the  $n - i - 1$  elements of vector  $u(i)$  for  $i = 1, 2, \dots, n-1$ . If  $m < n$ , the elements below the subdiagonal are the  $m - i - 1$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, m-1$ , and the elements above the diagonal are the  $n - i$  elements of vector  $u(i)$  for  $i = 1, 2, \dots, m$ .
- **lda** – [in] rocblas\_int.  $\text{lda} \geq m$ .  
specifies the leading dimension of A.
- **D** – [out] pointer to real type. Array on the GPU of dimension  $k$ .  
The diagonal elements of B.
- **E** – [out] pointer to real type. Array on the GPU of dimension  $k$ .  
The off-diagonal elements of B.
- **tauq** – [out] pointer to type. Array on the GPU of dimension  $k$ .  
The scalar factors of the Householder matrices H(i).



- **taup** – [out] pointer to type. Array on the GPU of dimension k.  
The scalar factors of the Householder matrices G(i).
- **X** – [out] pointer to type. Array on the GPU of dimension ldx\*k.  
The m-by-k matrix needed to reduce the unreduced part of A.
- **ldx** – [in] rocblas\_int. ldx >= m.  
specifies the leading dimension of X.
- **Y** – [out] pointer to type. Array on the GPU of dimension ldy\*k.  
The n-by-k matrix needed to reduce the unreduced part of A.
- **ldy** – [in] rocblas\_int. ldy >= n.  
specifies the leading dimension of Y.

### roc solver\_<type>bdsqr()

rocblas\_status **roc solver\_zbdsqr**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, const rocblas\_int nv, const rocblas\_int nu, const rocblas\_int nc, double \*D, double \*E, rocblas\_double\_complex \*V, const rocblas\_int ldv, rocblas\_double\_complex \*U, const rocblas\_int ldu, rocblas\_double\_complex \*C, const rocblas\_int ldc, rocblas\_int \*info)

rocblas\_status **roc solver\_cbdsqr**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, const rocblas\_int nv, const rocblas\_int nu, const rocblas\_int nc, float \*D, float \*E, rocblas\_float\_complex \*V, const rocblas\_int ldv, rocblas\_float\_complex \*U, const rocblas\_int ldu, rocblas\_float\_complex \*C, const rocblas\_int ldc, rocblas\_int \*info)

rocblas\_status **roc solver\_dbdsqr**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, const rocblas\_int nv, const rocblas\_int nu, const rocblas\_int nc, double \*D, double \*E, double \*V, const rocblas\_int ldv, double \*U, const rocblas\_int ldu, double \*C, const rocblas\_int ldc, rocblas\_int \*info)

rocblas\_status **roc solver\_sbdsqr**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, const rocblas\_int nv, const rocblas\_int nu, const rocblas\_int nc, float \*D, float \*E, float \*V, const rocblas\_int ldv, float \*U, const rocblas\_int ldu, float \*C, const rocblas\_int ldc, rocblas\_int \*info)

BDSQR computes the singular value decomposition (SVD) of a n-by-n bidiagonal matrix B.

The SVD of B has the form:

$$B = U_b * S * V_b^T$$

where S is the n-by-n diagonal matrix of singular values of B, the columns of U<sub>b</sub> are the left singular vectors of B, and the columns of V<sub>b</sub> are its right singular vectors.

The computation of the singular vectors is optional; this function accepts input matrices U (of size nu-by-n) and V (of size n-by-nv) that are overwritten with U\*U<sub>b</sub> and V<sub>b</sub>\*V. If nu = 0 no left vectors are computed; if nv = 0 no right vectors are computed.

Optionally, this function can also compute U<sub>b</sub>\*C for a given n-by-nc input matrix C.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether B is upper or lower bidiagonal.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of matrix B.
- **nv** – [in] rocblas\_int.  $nv \geq 0$ .  
The number of columns of matrix V.
- **nu** – [in] rocblas\_int.  $nu \geq 0$ .  
The number of rows of matrix U.
- **nc** – [in] rocblas\_int.  $nc \geq 0$ .  
The number of columns of matrix C.
- **D** – [inout] pointer to real type. Array on the GPU of dimension n.  
On entry, the diagonal elements of B. On exit, if  $info = 0$ , the singular values of B in decreasing order; if  $info > 0$ , the diagonal elements of a bidiagonal matrix orthogonally equivalent to B.
- **E** – [inout] pointer to real type. Array on the GPU of dimension  $n-1$ .  
On entry, the off-diagonal elements of B. On exit, if  $info > 0$ , the off-diagonal elements of a bidiagonal matrix orthogonally equivalent to B (if  $info = 0$  this matrix converges to zero).
- **V** – [inout] pointer to type. Array on the GPU of dimension  $ldv*nv$ .  
On entry, the matrix V. On exit, it is overwritten with  $Vb^*V$ . (Not referenced if  $nv = 0$ ).
- **ldv** – [in] rocblas\_int.  $ldv \geq n$  if  $nv > 0$ , or  $ldv \geq 1$  if  $nv = 0$ .  
Specifies the leading dimension of V.
- **U** – [inout] pointer to type. Array on the GPU of dimension  $ldu*n$ .  
On entry, the matrix U. On exit, it is overwritten with  $U*Ub$ . (Not referenced if  $nu = 0$ ).
- **ldu** – [in] rocblas\_int.  $ldu \geq nu$ .  
Specifies the leading dimension of U.
- **C** – [inout] pointer to type. Array on the GPU of dimension  $ldc*nc$ .  
On entry, the matrix C. On exit, it is overwritten with  $Ub^*C$ . (Not referenced if  $nc = 0$ ).
- **ldc** – [in] rocblas\_int.  $ldc \geq n$  if  $nc > 0$ , or  $ldc \geq 1$  if  $nc = 0$ .  
Specifies the leading dimension of C.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If  $info = 0$ , successful exit. If  $info = i > 0$ ,  $i$  elements of E have not converged to zero.

### 2.7.3.4 Tridiagonal forms

#### roc solver\_<type>latrd()

rocblas\_status **roc solver\_zlatrd**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, double \*E, rocblas\_double\_complex \*tau, rocblas\_double\_complex \*W, const rocblas\_int ldw)

rocblas\_status **roc solver\_clatrd**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, float \*E, rocblas\_float\_complex \*tau, rocblas\_float\_complex \*W, const rocblas\_int ldw)

rocblas\_status **roc solver\_dlatrd**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*E, double \*tau, double \*W, const rocblas\_int ldw)

rocblas\_status **roc solver\_slatrd**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*E, float \*tau, float \*W, const rocblas\_int ldw)

LATRD computes the tridiagonal form of  $k$  rows and columns of a symmetric/hermitian matrix  $A$ , as well as the matrix  $W$  needed to update the remaining part of  $A$ .

The reduced form is given by:

$$T = Q' * A * Q$$

If uplo is lower, the first  $k$  rows and columns of  $T$  form a tridiagonal block, if uplo is upper, then the last  $k$  rows and columns of  $T$  form the tridiagonal block.  $Q$  is an orthogonal/unitary matrix represented as the product of Householder matrices

$$Q = H(1) * H(2) * \dots * H(k) \text{ if uplo indicates lower, or } \\ Q = H(n-1) * H(n-2) * \dots * H(n-k) \text{ if uplo is upper.}$$

Each Householder matrix  $H(i)$  is given by

$$H(i) = I - \tau[i] * v(i) * v(i)'$$

where  $\tau[i]$  is the corresponding Householder scalar. When uplo indicates lower, the first  $i$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i+1] = 1$ . If uplo is upper, the last  $n-i$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i] = 1$ .

The unreduced part of the matrix  $A$  can be updated using a rank update of the form:

$$A = A - V * W' - W * V'$$

where  $V$  is an  $n$ -by- $k$  matrix formed by the vectors  $v(i)$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.

Specifies whether the upper or lower part of the matrix  $A$  is stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.

- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrix A.
- **k** – [in] rocblas\_int.  $0 \leq k \leq n$ .  
The number of rows and columns of the matrix A to be reduced.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the n-by-n matrix to be reduced. On exit, if uplo is lower, the first k columns have been reduced to tridiagonal form (given in the diagonal elements of A and the array E), the elements below the diagonal contain the vectors  $v(i)$  stored as columns. If uplo is upper, the last k columns have been reduced to tridiagonal form (given in the diagonal elements of A and the array E), the elements above the diagonal contain the vectors  $v(i)$  stored as columns.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of A.
- **E** – [out] pointer to real type. Array on the GPU of dimension  $n-1$ .  
If upper (lower), the last (first) k elements of E are the off-diagonal elements of the computed tridiagonal block.
- **tau** – [out] pointer to type. Array on the GPU of dimension  $n-1$ .  
If upper (lower), the last (first) k elements of tau are the scalar factors of the Householder matrices  $H(i)$ .
- **W** – [out] pointer to type. Array on the GPU of dimension  $ldw \times k$ .  
The n-by-k matrix needed to update the unreduced part of A.
- **ldw** – [in] rocblas\_int.  $ldw \geq n$ .  
specifies the leading dimension of W.

### roc solver\_<type>sterf()

rocblas\_status **roc solver\_dsterf**(rocblas\_handle handle, const rocblas\_int n, double \*D, double \*E, rocblas\_int \*info)

rocblas\_status **roc solver\_ssterf**(rocblas\_handle handle, const rocblas\_int n, float \*D, float \*E, rocblas\_int \*info)  
STERF computes the eigenvalues of a symmetric tridiagonal matrix.

The eigenvalues of the symmetric tridiagonal matrix are computed by the Pal-Walker-Kahan variant of the QL/QR algorithm, and returned in increasing order.

The matrix is not represented explicitly, but rather as the array of diagonal elements D and the array of symmetric off-diagonal elements E as returned by, e.g., SYTRD or HETRD.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the tridiagonal matrix.
- **D** – [inout] pointer to real type. Array on the GPU of dimension n.

On entry, the diagonal elements of the matrix. On exit, if `info = 0`, the eigenvalues in increasing order. If `info > 0`, the diagonal elements of a tridiagonal matrix that is similar to the original matrix (i.e. has the same eigenvalues).

- **E** – **[inout]** pointer to real type. Array on the GPU of dimension `n-1`.

On entry, the off-diagonal elements of the matrix. On exit, if `info = 0`, this array converges to zero. If `info > 0`, the off-diagonal elements of a tridiagonal matrix that is similar to the original matrix (i.e. has the same eigenvalues).

- **info** – **[out]** pointer to a `rocblas_int` on the GPU.

If `info = 0`, successful exit. If `info = i > 0`, STERF did not converge. `i` elements of `E` did not converge to zero.

### roc solver\_<type>steqr()

`rocblas_status roc solver_zsteqr(rocblas_handle handle, const rocblas_evect evect, const rocblas_int n, double *D, double *E, rocblas_double_complex *C, const rocblas_int ldc, rocblas_int *info)`

`rocblas_status roc solver_csteqr(rocblas_handle handle, const rocblas_evect evect, const rocblas_int n, float *D, float *E, rocblas_float_complex *C, const rocblas_int ldc, rocblas_int *info)`

`rocblas_status roc solver_dsteqr(rocblas_handle handle, const rocblas_evect evect, const rocblas_int n, double *D, double *E, double *C, const rocblas_int ldc, rocblas_int *info)`

`rocblas_status roc solver_ssteqr(rocblas_handle handle, const rocblas_evect evect, const rocblas_int n, float *D, float *E, float *C, const rocblas_int ldc, rocblas_int *info)`

STEQR computes the eigenvalues and (optionally) eigenvectors of a symmetric tridiagonal matrix.

The eigenvalues of the symmetric tridiagonal matrix are computed by the implicit QL/QR algorithm, and returned in increasing order.

The matrix is not represented explicitly, but rather as the array of diagonal elements `D` and the array of symmetric off-diagonal elements `E` as returned by, e.g., SYTRD or HETRD. If the tridiagonal matrix is the reduced form of a full symmetric/Hermitian matrix as returned by, e.g., SYTRD or HETRD, then the eigenvectors of the original matrix can also be computed, depending on the value of `evect`.

#### Parameters

- **handle** – **[in]** `rocblas_handle`.
- **evect** – **[in]** `rocblas_evect`.  
Specifies how the eigenvectors are computed.
- **n** – **[in]** `rocblas_int`. `n >= 0`.  
The number of rows and columns of the tridiagonal matrix.
- **D** – **[inout]** pointer to real type. Array on the GPU of dimension `n`.

On entry, the diagonal elements of the matrix. On exit, if `info = 0`, the eigenvalues in increasing order. If `info > 0`, the diagonal elements of a tridiagonal matrix that is similar to the original matrix (i.e. has the same eigenvalues).

- **E** – **[inout]** pointer to real type. Array on the GPU of dimension  $n-1$ .  
On entry, the off-diagonal elements of the matrix. On exit, if `info = 0`, this array converges to zero. If `info > 0`, the off-diagonal elements of a tridiagonal matrix that is similar to the original matrix (i.e. has the same eigenvalues).
- **C** – **[inout]** pointer to type. Array on the GPU of dimension  $ldc*n$ .  
On entry, if `evect` is original, the orthogonal/unitary matrix used for the reduction to tridiagonal form as returned by, e.g., `ORGTR` or `UNGTR`. On exit, it is overwritten with the eigenvectors of the original symmetric/Hermitian matrix (if `evect` is original), or the eigenvectors of the tridiagonal matrix (if `evect` is tridiagonal). (Not referenced if `evect` is none).
- **ldc** – **[in]** `rocblas_int`.  $ldc \geq n$  if `evect` is original or tridiagonal.  
Specifies the leading dimension of C. (Not referenced if `evect` is none).
- **info** – **[out]** pointer to a `rocblas_int` on the GPU.  
If `info = 0`, successful exit. If `info = i > 0`, STEQR did not converge.  $i$  elements of E did not converge to zero.

### 2.7.3.5 Orthonormal matrices

#### `roc solver_<type>org2r()`

`rocblas_status roc solver_dorg2r`(`rocblas_handle` handle, `const rocblas_int` m, `const rocblas_int` n, `const rocblas_int` k, `double *A`, `const rocblas_int` lda, `double *ipiv`)

`rocblas_status roc solver_sorg2r`(`rocblas_handle` handle, `const rocblas_int` m, `const rocblas_int` n, `const rocblas_int` k, `float *A`, `const rocblas_int` lda, `float *ipiv`)

ORG2R generates a  $m$ -by- $n$  Matrix Q with orthonormal columns.

(This is the unblocked version of the algorithm).

The matrix Q is defined as the first  $n$  columns of the product of  $k$  Householder reflectors of order  $m$

$$Q = H(1) * H(2) * \dots * H(k)$$

Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vector  $v(i)$  and scalar  $ipiv_i$  as returned by GEQRF.

#### Parameters

- **handle** – **[in]** `rocblas_handle`.
- **m** – **[in]** `rocblas_int`.  $m \geq 0$ .  
The number of rows of the matrix Q.
- **n** – **[in]** `rocblas_int`.  $0 \leq n \leq m$ .  
The number of columns of the matrix Q.
- **k** – **[in]** `rocblas_int`.  $0 \leq k \leq n$ .  
The number of Householder reflectors.
- **A** – **[inout]** pointer to type. Array on the GPU of dimension  $lda*n$ .  
On entry, the  $i$ -th column has Householder vector  $v(i)$ , for  $i = 1, 2, \dots, k$  as returned in the first  $k$  columns of matrix A of GEQRF. On exit, the computed matrix Q.

- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQRF.

### roc solver\_<type>orgqr()

rocblas\_status **roc solver\_dorgqr**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc solver\_sorgqr**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv)

ORGQR generates a m-by-n Matrix Q with orthonormal columns.

(This is the blocked version of the algorithm).

The matrix Q is defined as the first n columns of the product of k Householder reflectors of order m

$$Q = H(1) * H(2) * \dots * H(k)$$

Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vector  $v(i)$  and scalar  $ipiv_i$  as returned by GEQRF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix Q.
- **n** – [in] rocblas\_int.  $0 \leq n \leq m$ .  
The number of columns of the matrix Q.
- **k** – [in] rocblas\_int.  $0 \leq k \leq n$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda*n$ .  
On entry, the i-th column has Householder vector  $v(i)$ , for  $i = 1, 2, \dots, k$  as returned in the first k columns of matrix A of GEQRF. On exit, the computed matrix Q.
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQRF.

**roc solver\_<type>orgl2()**

rocblas\_status **roc solver\_dorgl2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc solver\_sorgl2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv)

ORGL2 generates a m-by-n Matrix Q with orthonormal rows.

(This is the unblocked version of the algorithm).

The matrix Q is defined as the first m rows of the product of k Householder reflectors of order n

$$Q = H(k) * H(k-1) * \dots * H(1)$$

Householder matrices H(i) are never stored, they are computed from its corresponding Householder vector v(i) and scalar ipiv\_i as returned by GELQF.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $0 \leq m \leq n$ .  
The number of rows of the matrix Q.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix Q.
- **k** – [in] rocblas\_int.  $0 \leq k \leq m$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension lda\*n.  
On entry, the i-th row has Householder vector v(i), for  $i = 1, 2, \dots, k$  as returned in the first k rows of matrix A of GELQF. On exit, the computed matrix Q.
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices H(i) as returned by GELQF.

**roc solver\_<type>orglq()**

rocblas\_status **roc solver\_dorglq**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc solver\_sorglq**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv)

ORGLQ generates a m-by-n Matrix Q with orthonormal rows.

(This is the blocked version of the algorithm).

The matrix Q is defined as the first m rows of the product of k Householder reflectors of order n



$$Q = H(k) * H(k-1) * \dots * H(1)$$

Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vector  $v(i)$  and scalar  $ipiv\_i$  as returned by GELQF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $0 \leq m \leq n$ .  
The number of rows of the matrix Q.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix Q.
- **k** – [in] rocblas\_int.  $0 \leq k \leq m$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda * n$ .  
On entry, the  $i$ -th row has Householder vector  $v(i)$ , for  $i = 1, 2, \dots, k$  as returned in the first  $k$  rows of matrix A of GELQF. On exit, the computed matrix Q.
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GELQF.

### roc solver\_<type>org2l()

rocblas\_status **roc solver\_dorg2l**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc solver\_sorg2l**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv)

ORG2L generates a  $m$ -by- $n$  Matrix Q with orthonormal columns.

(This is the unblocked version of the algorithm).

The matrix Q is defined as the last  $n$  columns of the product of  $k$  Householder reflectors of order  $m$

$$Q = H(k) * H(k-1) * \dots * H(1)$$

Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vector  $v(i)$  and scalar  $ipiv\_i$  as returned by GEQLF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix Q.
- **n** – [in] rocblas\_int.  $0 \leq n \leq m$ .  
The number of columns of the matrix Q.

- **k** – [in] rocblas\_int.  $0 \leq k \leq n$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the  $(n-k+i)$ -th column has Householder vector  $v(i)$ , for  $i = 1, 2, \dots, k$  as returned in the last  $k$  columns of matrix  $A$  of GEQLF. On exit, the computed matrix  $Q$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQLF.

### roc solver\_<type>orgql()

rocblas\_status **roc solver\_dorgql**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc solver\_sorgql**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv)

ORGQL generates a  $m$ -by- $n$  Matrix  $Q$  with orthonormal columns.

(This is the blocked version of the algorithm).

The matrix  $Q$  is defined as the last  $n$  column of the product of  $k$  Householder reflectors of order  $m$

$$Q = H(k) * H(k-1) * \dots * H(1)$$

Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vector  $v(i)$  and scalar  $ipiv_i$  as returned by GEQLF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix  $Q$ .
- **n** – [in] rocblas\_int.  $0 \leq n \leq m$ .  
The number of columns of the matrix  $Q$ .
- **k** – [in] rocblas\_int.  $0 \leq k \leq n$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the  $(n-k+i)$ -th column has Householder vector  $v(i)$ , for  $i = 1, 2, \dots, k$  as returned in the last  $k$  columns of matrix  $A$  of GEQLF. On exit, the computed matrix  $Q$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQLF.

**rocblas\_status roc solver\_<type>orgbr()**

rocblas\_status **roc solver\_dorgbr**(rocblas\_handle handle, const *rocblas\_storev* storev, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc solver\_sorgbr**(rocblas\_handle handle, const *rocblas\_storev* storev, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv)

ORGBR generates a m-by-n Matrix Q with orthonormal rows or columns.

If storev is column-wise, then the matrix Q has orthonormal columns. If  $m \geq k$ , Q is defined as the first n columns of the product of k Householder reflectors of order m

$$Q = H(1) * H(2) * \dots * H(k)$$

If  $m < k$ , Q is defined as the product of Householder reflectors of order m

$$Q = H(1) * H(2) * \dots * H(m-1)$$

On the other hand, if storev is row-wise, then the matrix Q has orthonormal rows. If  $n > k$ , Q is defined as the first m rows of the product of k Householder reflectors of order n

$$Q = H(k) * H(k-1) * \dots * H(1)$$

If  $n \leq k$ , Q is defined as the product of Householder reflectors of order n

$$Q = H(n-1) * H(n-2) * \dots * H(1)$$

The Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vectors  $v(i)$  and scalars  $ipiv_i$  as returned by GEBRD in its arguments A and tauq or taup.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **storev** – [in] *rocblas\_storev* .  
Specifies whether to work column-wise or row-wise.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix Q. If row-wise, then  $\min(n,k) \leq m \leq n$ .
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix Q. If column-wise, then  $\min(m,k) \leq n \leq m$ .
- **k** – [in] rocblas\_int.  $k \geq 0$ .  
The number of columns (if storev is colum-wise) or rows (if row-wise) of the original matrix reduced by GEBRD.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda*n$ .  
On entry, the i-th column (or row) has the Householder vector  $v(i)$  as returned by GEBRD.  
On exit, the computed matrix Q.
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.

- **ipiv** – [in] pointer to type. Array on the GPU of dimension  $\min(m,k)$  if column-wise, or  $\min(n,k)$  if row-wise.

The scalar factors of the Householder matrices  $H(i)$  as returned by GEBRD.

### rocblas\_status roc\_solver\_<type>orgtr()

rocblas\_status **roc\_solver\_dorgtr**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc\_solver\_sorgtr**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*ipiv)

ORGTR generates a  $n$ -by- $n$  orthogonal Matrix  $Q$ .

$Q$  is defined as the product of  $n-1$  Householder reflectors of order  $n$ . If uplo indicates upper, then  $Q$  has the form

$$Q = H(n-1) * H(n-2) * \dots * H(1)$$

On the other hand, if uplo indicates lower, then  $Q$  has the form

$$Q = H(1) * H(2) * \dots * H(n-1)$$

The Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vectors  $v(i)$  and scalars  $ipiv_i$  as returned by SYTRD in its arguments  $A$  and  $\tau$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the SYTRD factorization was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrix  $Q$ .
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda*n$ .  
On entry, the  $(i+1)$ -th column (if uplo indicates upper) or  $i$ -th column (if uplo indicates lower) has the Householder vector  $v(i)$  as returned by SYTRD. On exit, the computed matrix  $Q$ .
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension  $n-1$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by SYTRD.

**roc solver\_<type>orm2r()**

rocblas\_status **roc solver\_dorm2r**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv, double \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_sorm2r**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv, float \*C, const rocblas\_int ldc)

ORM2R applies a matrix Q with orthonormal columns to a general m-by-n matrix C.

(This is the unblocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q	*	C	(No transpose from the left)
Q'	*	C	(Transpose from the left)
C	*	Q	(No transpose from the right), and
C	*	Q'	(Transpose from the right)

Q is an orthogonal matrix defined as the product of k Householder reflectors as

$Q = H(1) * H(2) * \dots * H(k)$
----------------------------------

of order m if applying from the left, or n if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the QR factorization GEQRF.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix C.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix C.
- **k** – [in] rocblas\_int.  $k \geq 0$ ;  $k \leq m$  if side is left,  $k \leq n$  if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size  $lda * k$ .  
The i-th column has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GEQRF in the first k columns of its argument A.
- **lda** – [in] rocblas\_int.  $lda \geq m$  if side is left, or  $lda \geq n$  if side is right.  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQRF.

- **C** – [inout] pointer to type. Array on the GPU of size ldc\*n.  
On input, the matrix C. On output it is overwritten with Q\*C, C\*Q, Q'\*C, or C\*Q'.
- **ldc** – [in] rocblas\_int. ldc >= m.  
Leading dimension of C.

### roc solver\_<type>ormqr()

rocblas\_status **roc solver\_dormqr**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv, double \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_sormqr**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv, float \*C, const rocblas\_int ldc)

ORMQR applies a matrix Q with orthonormal columns to a general m-by-n matrix C.

(This is the blocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q	*	C	(No transpose from the left)
Q'	*	C	(Transpose from the left)
C	*	Q	(No transpose from the right), and
C	*	Q'	(Transpose from the right)

Q is an orthogonal matrix defined as the product of k Householder reflectors as

$Q = H(1) * H(2) * \dots * H(k)$
----------------------------------

of order m if applying from the left, or n if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the QR factorization GEQRF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its transpose is to be applied.
- **m** – [in] rocblas\_int. m >= 0.  
Number of rows of matrix C.
- **n** – [in] rocblas\_int. n >= 0.  
Number of columns of matrix C.
- **k** – [in] rocblas\_int. k >= 0; k <= m if side is left, k <= n if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size lda\*k.

The  $i$ -th column has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GEQRF in the first  $k$  columns of its argument  $A$ .

- **lda** – [in] rocblas\_int.  $lda \geq m$  if side is left, or  $lda \geq n$  if side is right.  
Leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQRF.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc*n$ .  
On input, the matrix  $C$ . On output it is overwritten with  $Q*C$ ,  $C*Q$ ,  $Q'*C$ , or  $C*Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of  $C$ .

### roc solver\_<type>orml2()

rocblas\_status **roc solver\_dorml2**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv, double \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_sorml2**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv, float \*C, const rocblas\_int ldc)

ORML2 applies a matrix  $Q$  with orthonormal rows to a general  $m$ -by- $n$  matrix  $C$ .

(This is the unblocked version of the algorithm).

The matrix  $Q$  is applied in one of the following forms, depending on the values of side and trans:

$Q * C$	(No transpose <b>from the left</b> )
$Q' * C$	(Transpose <b>from the left</b> )
$C * Q$	(No transpose <b>from the right</b> ), <b>and</b>
$C * Q'$	(Transpose <b>from the right</b> )

$Q$  is an orthogonal matrix defined as the product of  $k$  Householder reflectors as

$$Q = H(k) * H(k-1) * \dots * H(1)$$

of order  $m$  if applying from the left, or  $n$  if applying from the right.  $Q$  is never stored, it is calculated from the Householder vectors and scalars returned by the LQ factorization GELQF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply  $Q$ .
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix  $Q$  or its transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix  $C$ .

- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix C.
- **k** – [in] rocblas\_int.  $k \geq 0$ ;  $k \leq m$  if side is left,  $k \leq n$  if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size  $lda \times m$  if side is left, or  $lda \times n$  if side is right.  
The  $i$ -th row has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GELQF in the first  $k$  rows of its argument A.
- **lda** – [in] rocblas\_int.  $lda \geq k$ .  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GELQF.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc \times n$ .  
On input, the matrix C. On output it is overwritten with  $Q \times C$ ,  $C \times Q$ ,  $Q' \times C$ , or  $C \times Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of C.

### roc solver\_<type>ormlq()

rocblas\_status **roc solver\_dormlq**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv, double \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_sormlq**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv, float \*C, const rocblas\_int ldc)

ORMLQ applies a matrix Q with orthonormal rows to a general m-by-n matrix C.

(This is the blocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q * C	(No transpose from the left)
Q' * C	(Transpose from the left)
C * Q	(No transpose from the right), and
C * Q'	(Transpose from the right)

Q is an orthogonal matrix defined as the product of  $k$  Householder reflectors as

$$Q = H(k) * H(k-1) * \dots * H(1)$$

of order  $m$  if applying from the left, or  $n$  if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the LQ factorization GELQF.

#### Parameters

- **handle** – [in] rocblas\_handle.



- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix C.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix C.
- **k** – [in] rocblas\_int.  $k \geq 0$ ;  $k \leq m$  if side is left,  $k \leq n$  if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size  $lda \cdot m$  if side is left, or  $lda \cdot n$  if side is right.  
The  $i$ -th row has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GELQF in the first  $k$  rows of its argument A.
- **lda** – [in] rocblas\_int.  $lda \geq k$ .  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GELQF.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc \cdot n$ .  
On input, the matrix C. On output it is overwritten with  $Q \cdot C$ ,  $C \cdot Q$ ,  $Q' \cdot C$ , or  $C \cdot Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of C.

### roc solver\_<type>orm2l()

rocblas\_status **roc solver\_dorm2l**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv, double \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_sorm2l**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv, float \*C, const rocblas\_int ldc)

ORM2L applies a matrix Q with orthonormal columns to a general m-by-n matrix C.

(This is the unblocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q	*	C	(No transpose from the left)
Q'	*	C	(Transpose from the left)
C	*	Q	(No transpose from the right), and
C	*	Q'	(Transpose from the right)

Q is an orthogonal matrix defined as the product of  $k$  Householder reflectors as

$$Q = H(k) * H(k-1) * \dots * H(1)$$

of order  $m$  if applying from the left, or  $n$  if applying from the right.  $Q$  is never stored, it is calculated from the Householder vectors and scalars returned by the QL factorization GEQLF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply  $Q$ .
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix  $Q$  or its transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix  $C$ .
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix  $C$ .
- **k** – [in] rocblas\_int.  $k \geq 0$ ;  $k \leq m$  if side is left,  $k \leq n$  if side is right.  
The number of Householder reflectors that form  $Q$ .
- **A** – [in] pointer to type. Array on the GPU of size  $lda * k$ .  
The  $i$ -th column has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GEQLF in the last  $k$  columns of its argument  $A$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$  if side is left,  $lda \geq n$  if side is right.  
Leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQLF.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc * n$ .  
On input, the matrix  $C$ . On output it is overwritten with  $Q * C$ ,  $C * Q$ ,  $Q' * C$ , or  $C * Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of  $C$ .

#### roc solver\_<type>ormql()

rocblas\_status **roc solver\_dormql**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv, double \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_sormql**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv, float \*C, const rocblas\_int ldc)

ORMQL applies a matrix  $Q$  with orthonormal columns to a general  $m$ -by- $n$  matrix  $C$ .

(This is the blocked version of the algorithm).

The matrix  $Q$  is applied in one of the following forms, depending on the values of side and trans:

```

Q * C (No transpose from the left)
Q' * C (Transpose from the left)
C * Q (No transpose from the right), and
C * Q' (Transpose from the right)

```

Q is an orthogonal matrix defined as the product of k Householder reflectors as

$$Q = H(k) * H(k-1) * \dots * H(1)$$

of order m if applying from the left, or n if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the QL factorization GEQLF.

### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix C.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix C.
- **k** – [in] rocblas\_int.  $k \geq 0$ ;  $k \leq m$  if side is left,  $k \leq n$  if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size  $lda * k$ .  
The i-th column has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GEQLF in the last k columns of its argument A.
- **lda** – [in] rocblas\_int.  $lda \geq m$  if side is left,  $lda \geq n$  if side is right.  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQLF.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc * n$ .  
On input, the matrix C. On output it is overwritten with  $Q * C$ ,  $C * Q$ ,  $Q' * C$ , or  $C * Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of C.

**roc solver\_<type>ormbr()**

rocblas\_status **roc solver\_dormbr**(rocblas\_handle handle, const *rocblas\_storev* storev, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, double \*A, const rocblas\_int lda, double \*ipiv, double \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_sormbr**(rocblas\_handle handle, const *rocblas\_storev* storev, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, float \*A, const rocblas\_int lda, float \*ipiv, float \*C, const rocblas\_int ldc)

ORMBR applies a matrix Q with orthonormal rows or columns to a general m-by-n matrix C.

If storev is column-wise, then the matrix Q has orthonormal columns. If storev is row-wise, then the matrix Q has orthonormal rows. The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q	*	C	(No transpose <b>from the left</b> )
Q'	*	C	(Transpose <b>from the left</b> )
C	*	Q	(No transpose <b>from the right</b> ), <b>and</b>
C	*	Q'	(Transpose <b>from the right</b> )

The order nq of orthogonal matrix Q is nq = m if applying from the left, or nq = n if applying from the right.

When storev is column-wise, if nq >= k, then Q is defined as the product of k Householder reflectors of order nq

$$Q = H(1) * H(2) * \dots * H(k),$$

and if nq < k, then Q is defined as the product

$$Q = H(1) * H(2) * \dots * H(nq-1).$$

When storev is row-wise, if nq > k, then Q is defined as the product of k Householder reflectors of order nq

$$Q = H(1) * H(2) * \dots * H(k),$$

and if n <= k, Q is defined as the product

$$Q = H(1) * H(2) * \dots * H(nq-1)$$

The Householder matrices H(i) are never stored, they are computed from its corresponding Householder vectors v(i) and scalars ipiv\_i as returned by GEBRD in its arguments A and tauq or taup.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **storev** – [in] *rocblas\_storev* .  
Specifies whether to work column-wise or row-wise.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its transpose is to be applied.

- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix C.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix C.
- **k** – [in] rocblas\_int.  $k \geq 0$ .  
The number of columns (if storev is colum-wise) or rows (if row-wise) of the original matrix reduced by GEBRD.
- **A** – [in] pointer to type. Array on the GPU of size  $lda \cdot \min(nq, k)$  if column-wise, or  $lda \cdot nq$  if row-wise.  
The i-th column (or row) has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GEBRD.
- **lda** – [in] rocblas\_int.  $lda \geq nq$  if column-wise, or  $lda \geq \min(nq, k)$  if row-wise.  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $\min(nq, k)$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEBRD.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc \cdot n$ .  
On input, the matrix C. On output it is overwritten with  $Q \cdot C$ ,  $C \cdot Q$ ,  $Q' \cdot C$ , or  $C \cdot Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of C.

### roc solver\_<type>ormtr()

rocblas\_status **roc solver\_dormtr**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_fill uplo, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*ipiv, double \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_sormtr**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_fill uplo, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*ipiv, float \*C, const rocblas\_int ldc)

ORMTR applies an orthogonal matrix Q to a general m-by-n matrix C.

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q	*	C	(No transpose <b>from the left</b> )
Q'	*	C	(Transpose <b>from the left</b> )
C	*	Q	(No transpose <b>from the right</b> ), <b>and</b>
C	*	Q'	(Transpose <b>from the right</b> )

The order nq of orthogonal matrix Q is  $nq = m$  if applying from the left, or  $nq = n$  if applying from the right.

Q is defined as the product of  $nq-1$  Householder reflectors of order nq. If uplo indicates upper, then Q has the form

$Q = H(nq-1) * H(nq-2) * \dots * H(1).$
---

On the other hand, if uplo indicates lower, then Q has the form

$$Q = H(1) * H(2) * \dots * H(nq-1)$$

The Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vectors  $v(i)$  and scalars  $ipiv\_i$  as returned by SYTRD in its arguments  $A$  and  $\tau$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply  $Q$ .
- **uplo** – [in] rocblas\_fill.  
Specifies whether the SYTRD factorization was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix  $Q$  or its transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix  $C$ .
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix  $C$ .
- **A** – [in] pointer to type. Array on the GPU of size  $lda * nq$ .  
On entry, the  $(i+1)$ -th column (if uplo indicates upper) or  $i$ -th column (if uplo indicates lower) has the Householder vector  $v(i)$  as returned by SYTRD.
- **lda** – [in] rocblas\_int.  $lda \geq nq$ .  
Leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $nq-1$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by SYTRD.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc * n$ .  
On input, the matrix  $C$ . On output it is overwritten with  $Q * C$ ,  $C * Q$ ,  $Q' * C$ , or  $C * Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of  $C$ .

### 2.7.3.6 Unitary matrices

#### roc solver\_<type>ung2r()

rocblas\_status **roc solver\_zung2r**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cung2r**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

UNG2R generates a  $m$ -by- $n$  complex Matrix  $Q$  with orthonormal columns.

(This is the unblocked version of the algorithm).

The matrix Q is defined as the first n columns of the product of k Householder reflectors of order m

$$Q = H(1) * H(2) * \dots * H(k)$$

Householder matrices H(i) are never stored, they are computed from its corresponding Householder vector v(i) and scalar ipiv\_i as returned by GEQRF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix Q.
- **n** – [in] rocblas\_int.  $0 \leq n \leq m$ .  
The number of columns of the matrix Q.
- **k** – [in] rocblas\_int.  $0 \leq k \leq n$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension lda\*n.  
On entry, the i-th column has Householder vector v(i), for  $i = 1, 2, \dots, k$  as returned in the first k columns of matrix A of GEQRF. On exit, the computed matrix Q.
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices H(i) as returned by GEQRF.

### roc solver\_<type>ungqr()

rocblas\_status **roc solver\_zungqr**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cungqr**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

UNGQR generates a m-by-n complex Matrix Q with orthonormal columns.

(This is the blocked version of the algorithm).

The matrix Q is defined as the first n columns of the product of k Householder reflectors of order m

$$Q = H(1) * H(2) * \dots * H(k)$$

Householder matrices H(i) are never stored, they are computed from its corresponding Householder vector v(i) and scalar ipiv\_i as returned by GEQRF.

#### Parameters

- **handle** – [in] rocblas\_handle.

- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix Q.
- **n** – [in] rocblas\_int.  $0 \leq n \leq m$ .  
The number of columns of the matrix Q.
- **k** – [in] rocblas\_int.  $0 \leq k \leq n$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the  $i$ -th column has Householder vector  $v(i)$ , for  $i = 1, 2, \dots, k$  as returned in the first  $k$  columns of matrix A of GEQRF. On exit, the computed matrix Q.
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQRF.

### roc solver\_<type>ungl2()

rocblas\_status **roc solver\_zungl2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cungl2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

UNGL2 generates a  $m$ -by- $n$  complex Matrix Q with orthonormal rows.

(This is the unblocked version of the algorithm).

The matrix Q is defined as the first  $m$  rows of the product of  $k$  Householder reflectors of order  $n$

$$Q = H(k) ** H * H(k-1) ** H * \dots * H(1) ** H$$

Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vector  $v(i)$  and scalar  $ipiv_i$  as returned by GELQF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $0 \leq m \leq n$ .  
The number of rows of the matrix Q.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix Q.
- **k** – [in] rocblas\_int.  $0 \leq k \leq m$ .  
The number of Householder reflectors.



- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the  $i$ -th row has Householder vector  $v(i)$ , for  $i = 1, 2, \dots, k$  as returned in the first  $k$  rows of matrix  $A$  of GELQF. On exit, the computed matrix  $Q$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GELQF.

### roc solver\_<type>unglq()

rocblas\_status **roc solver\_zunglq**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cunglq**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

UNGLQ generates a  $m$ -by- $n$  complex Matrix  $Q$  with orthonormal rows.

(This is the blocked version of the algorithm).

The matrix  $Q$  is defined as the first  $m$  rows of the product of  $k$  Householder reflectors of order  $n$

$$Q = H(k) ** H * H(k-1) ** H * \dots * H(1) ** H$$

Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vector  $v(i)$  and scalar  $ipiv_i$  as returned by GELQF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $0 \leq m \leq n$ .  
The number of rows of the matrix  $Q$ .
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix  $Q$ .
- **k** – [in] rocblas\_int.  $0 \leq k \leq m$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the  $i$ -th row has Householder vector  $v(i)$ , for  $i = 1, 2, \dots, k$  as returned in the first  $k$  rows of matrix  $A$  of GELQF. On exit, the computed matrix  $Q$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GELQF.

**roc solver\_<type>ung2l()**

rocblas\_status **roc solver\_zung2l**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cung2l**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

UNG2L generates a m-by-n complex Matrix Q with orthonormal columns.

(This is the unblocked version of the algorithm).

The matrix Q is defined as the last n columns of the product of k Householder reflectors of order m

$$Q = H(k) * H(k-1) * \dots * H(1)$$

Householder matrices H(i) are never stored, they are computed from its corresponding Householder vector v(i) and scalar ipiv\_i as returned by GEQLF.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix Q.
- **n** – [in] rocblas\_int.  $0 \leq n \leq m$ .  
The number of columns of the matrix Q.
- **k** – [in] rocblas\_int.  $0 \leq k \leq n$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda * n$ .  
On entry, the (n-k+i)-th column has Householder vector v(i), for  $i = 1, 2, \dots, k$  as returned in the last k columns of matrix A of GEQLF. On exit, the computed matrix Q.
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices H(i) as returned by GEQLF.

**roc solver\_<type>ungql()**

rocblas\_status **roc solver\_zungql**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cungql**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

UNGQL generates a m-by-n complex Matrix Q with orthonormal columns.

(This is the blocked version of the algorithm).

The matrix Q is defined as the last n columns of the product of k Householder reflectors of order m

$$Q = H(k) * H(k-1) * \dots * H(1)$$

Householder matrices H(i) are never stored, they are computed from its corresponding Householder vector v(i) and scalar ipiv\_i as returned by GEQLF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix Q.
- **n** – [in] rocblas\_int.  $0 \leq n \leq m$ .  
The number of columns of the matrix Q.
- **k** – [in] rocblas\_int.  $0 \leq k \leq n$ .  
The number of Householder reflectors.
- **A** – [inout] pointer to type. Array on the GPU of dimension lda\*n.  
On entry, the (n-k+i)-th column has Householder vector v(i), for  $i = 1, 2, \dots, k$  as returned in the last k columns of matrix A of GEQLF. On exit, the computed matrix Q.
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices H(i) as returned by GEQLF.

### roc solver\_<type>ungbr()

rocblas\_status **roc solver\_zungbr**(rocblas\_handle handle, const *rocblas\_storev* storev, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cungbr**(rocblas\_handle handle, const *rocblas\_storev* storev, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

UNGBR generates a m-by-n complex Matrix Q with orthonormal rows or columns.

If storev is column-wise, then the matrix Q has orthonormal columns. If  $m \geq k$ , Q is defined as the first n columns of the product of k Householder reflectors of order m

$$Q = H(1) * H(2) * \dots * H(k)$$

If  $m < k$ , Q is defined as the product of Householder reflectors of order m

$$Q = H(1) * H(2) * \dots * H(m-1)$$

On the other hand, if storev is row-wise, then the matrix Q has orthonormal rows. If  $n > k$ , Q is defined as the first m rows of the product of k Householder reflectors of order n

$$Q = H(k) * H(k-1) * \dots * H(1)$$

If  $n \leq k$ ,  $Q$  is defined as the product of Householder reflectors of order  $n$

$$Q = H(n-1) * H(n-2) * \dots * H(1)$$

The Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vectors  $v(i)$  and scalars  $ipiv_i$  as returned by GEBRD in its arguments  $A$  and  $tauq$  or  $taup$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **storev** – [in] *rocblas\_storev* .  
Specifies whether to work column-wise or row-wise.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix  $Q$ . If row-wise, then  $\min(n,k) \leq m \leq n$ .
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix  $Q$ . If column-wise, then  $\min(m,k) \leq n \leq m$ .
- **k** – [in] rocblas\_int.  $k \geq 0$ .  
The number of columns (if storev is column-wise) or rows (if row-wise) of the original matrix reduced by GEBRD.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda*n$ .  
On entry, the  $i$ -th column (or row) has the Householder vector  $v(i)$  as returned by GEBRD.  
On exit, the computed matrix  $Q$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension  $\min(m,k)$  if column-wise, or  $\min(n,k)$  if row-wise.  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEBRD.

#### roc solver\_<type>ungtr()

rocblas\_status **roc solver\_zungtr**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cungtr**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

UNGTR generates a  $n$ -by- $n$  unitary Matrix  $Q$ .

$Q$  is defined as the product of  $n-1$  Householder reflectors of order  $n$ . If uplo indicates upper, then  $Q$  has the form

$$Q = H(n-1) * H(n-2) * \dots * H(1)$$

On the other hand, if uplo indicates lower, then  $Q$  has the form

$$Q = H(1) * H(2) * \dots * H(n-1)$$

The Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vectors  $v(i)$  and scalars  $ipiv\_i$  as returned by HETRD in its arguments  $A$  and  $\tau$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the HETRD factorization was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrix  $Q$ .
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda * n$ .  
On entry, the  $(i+1)$ -th column (if uplo indicates upper) or  $i$ -th column (if uplo indicates lower) has the Householder vector  $v(i)$  as returned by HETRD. On exit, the computed matrix  $Q$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of  $A$ .
- **ipiv** – [in] pointer to type. Array on the GPU of dimension  $n-1$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by HETRD.

### roc solver\_<type>unm2r()

rocblas\_status **roc solver\_zunm2r**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv, rocblas\_double\_complex \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_cunm2r**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv, rocblas\_float\_complex \*C, const rocblas\_int ldc)

UNM2R applies a complex matrix  $Q$  with orthonormal columns to a general  $m$ -by- $n$  matrix  $C$ .

(This is the unblocked version of the algorithm).

The matrix  $Q$  is applied in one of the following forms, depending on the values of  $side$  and  $trans$ :

$Q * C$	(No transpose <b>from the left</b> )
$Q' * C$	(Conjugate transpose <b>from the left</b> )
$C * Q$	(No transpose <b>from the right</b> ), <b>and</b>
$C * Q'$	(Conjugate transpose <b>from the right</b> )

$Q$  is a unitary matrix defined as the product of  $k$  Householder reflectors as

$$Q = H(1) * H(2) * \dots * H(k)$$

of order  $m$  if applying from the left, or  $n$  if applying from the right.  $Q$  is never stored, it is calculated from the Householder vectors and scalars returned by the QR factorization GEQRF.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its conjugate transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix C.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix C.
- **k** – [in] rocblas\_int.  $k \geq 0$ ;  $k \leq m$  if side is left,  $k \leq n$  if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size  $lda \times k$ .  
The i-th column has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GEQRF in the first k columns of its argument A.
- **lda** – [in] rocblas\_int.  $lda \geq m$  if side is left, or  $lda \geq n$  if side is right.  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQRF.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc \times n$ .  
On input, the matrix C. On output it is overwritten with  $Q \times C$ ,  $C \times Q$ ,  $Q' \times C$ , or  $C \times Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of C.

**roc solver\_<type>unmqr()**

rocblas\_status **roc solver\_zunmqr**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv, rocblas\_double\_complex \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_cunmqr**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv, rocblas\_float\_complex \*C, const rocblas\_int ldc)

UNMQR applies a complex matrix Q with orthonormal columns to a general m-by-n matrix C.

(This is the blocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

```

Q * C (No transpose from the left)
Q' * C (Conjugate transpose from the left)
C * Q (No transpose from the right), and
C * Q' (Conjugate transpose from the right)

```

Q is a unitary matrix defined as the product of k Householder reflectors as

$$Q = H(1) * H(2) * \dots * H(k)$$

of order m if applying from the left, or n if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the QR factorization GEQRF.

### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its conjugate transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix C.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix C.
- **k** – [in] rocblas\_int.  $k \geq 0$ ;  $k \leq m$  if side is left,  $k \leq n$  if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size  $lda * k$ .  
The i-th column has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GEQRF in the first k columns of its argument A.
- **lda** – [in] rocblas\_int.  $lda \geq m$  if side is left, or  $lda \geq n$  if side is right.  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQRF.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc * n$ .  
On input, the matrix C. On output it is overwritten with  $Q * C$ ,  $C * Q$ ,  $Q' * C$ , or  $C * Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of C.

**roc solver\_<type>unml2()**

rocblas\_status **roc solver\_zunml2**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv, rocblas\_double\_complex \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_cunml2**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv, rocblas\_float\_complex \*C, const rocblas\_int ldc)

UNML2 applies a complex matrix Q with orthonormal rows to a general m-by-n matrix C.

(This is the unblocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q	*	C	(No transpose <b>from the left</b> )
Q'	*	C	(Conjugate transpose <b>from the left</b> )
C	*	Q	(No transpose <b>from the right</b> ), <b>and</b>
C	*	Q'	(Conjugate transpose <b>from the right</b> )

Q is a unitary matrix defined as the product of k Householder reflectors as

$$Q = H(k)**H * H(k-1)**H * \dots * H(1)**H$$

of order m if applying from the left, or n if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the LQ factorization GELQF.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its conjugate transpose is to be applied.
- **m** – [in] rocblas\_int. m >= 0.  
Number of rows of matrix C.
- **n** – [in] rocblas\_int. n >= 0.  
Number of columns of matrix C.
- **k** – [in] rocblas\_int. k >= 0; k <= m if side is left, k <= n if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size lda\*m if side is left, or lda\*n if side is right.  
The i-th row has the Householder vector v(i) associated with H(i) as returned by GELQF in the first k rows of its argument A.
- **lda** – [in] rocblas\_int. lda >= k.  
Leading dimension of A.



- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices H(i) as returned by GELQF.
- **C** – [inout] pointer to type. Array on the GPU of size ldc\*n.  
On input, the matrix C. On output it is overwritten with Q\*C, C\*Q, Q'\*C, or C\*Q'.
- **ldc** – [in] rocblas\_int. ldc >= m.  
Leading dimension of C.

### roc solver\_<type>unmlq()

rocblas\_status **roc solver\_zunmlq**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv, rocblas\_double\_complex \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_cunmlq**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv, rocblas\_float\_complex \*C, const rocblas\_int ldc)

UNMLQ applies a complex matrix Q with orthonormal rows to a general m-by-n matrix C.

(This is the blocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q	*	C	(No transpose <b>from the left</b> )
Q'	*	C	(Conjugate transpose <b>from the left</b> )
C	*	Q	(No transpose <b>from the right</b> ), <b>and</b>
C	*	Q'	(Conjugate transpose <b>from the right</b> )

Q is a unitary matrix defined as the product of k Householder reflectors as

$$Q = H(k)**H * H(k-1)**H * \dots * H(1)**H$$

of order m if applying from the left, or n if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the LQ factorization GELQF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its conjugate transpose is to be applied.
- **m** – [in] rocblas\_int. m >= 0.  
Number of rows of matrix C.
- **n** – [in] rocblas\_int. n >= 0.  
Number of columns of matrix C.

- **k** – [in] rocblas\_int.  $k \geq 0$ ;  $k \leq m$  if side is left,  $k \leq n$  if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size  $lda*m$  if side is left, or  $lda*n$  if side is right.  
The  $i$ -th row has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GELQF in the first  $k$  rows of its argument A.
- **lda** – [in] rocblas\_int.  $lda \geq k$ .  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GELQF.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc*n$ .  
On input, the matrix C. On output it is overwritten with  $Q*C$ ,  $C*Q$ ,  $Q'*C$ , or  $C*Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of C.

### roc solver\_<type>unm2l()

rocblas\_status **roc solver\_zunm2l**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv, rocblas\_double\_complex \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_cunm2l**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv, rocblas\_float\_complex \*C, const rocblas\_int ldc)

UNM2L applies a complex matrix Q with orthonormal columns to a general  $m$ -by- $n$  matrix C.

(This is the unblocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q * C	(No transpose <b>from the left</b> )
Q' * C	(Conjugate transpose <b>from the left</b> )
C * Q	(No transpose <b>from the right</b> ), <b>and</b>
C * Q'	(Conjugate transpose <b>from the right</b> )

Q is a unitary matrix defined as the product of  $k$  Householder reflectors as

$$Q = H(k) * H(k-1) * \dots * H(1)$$

of order  $m$  if applying from the left, or  $n$  if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the QL factorization GEQLF.

#### Parameters

- **handle** – [in] rocblas\_handle.

- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its conjugate transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix C.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix C.
- **k** – [in] rocblas\_int.  $k \geq 0$ ;  $k \leq m$  if side is left,  $k \leq n$  if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size  $lda \times k$ .  
The  $i$ -th column has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GEQLF in the last  $k$  columns of its argument A.
- **lda** – [in] rocblas\_int.  $lda \geq m$  if side is left,  $lda \geq n$  if side is right.  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $k$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by GEQLF.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc \times n$ .  
On input, the matrix C. On output it is overwritten with  $Q \times C$ ,  $C \times Q$ ,  $Q' \times C$ , or  $C \times Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of C.

### roc solver\_<type>unmql()

rocblas\_status **roc solver\_zunmql**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv, rocblas\_double\_complex \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_cunmql**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv, rocblas\_float\_complex \*C, const rocblas\_int ldc)

UNMQL applies a complex matrix Q with orthonormal columns to a general  $m$ -by- $n$  matrix C.

(This is the blocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q * C	(No transpose from the left)
Q' * C	(Conjugate transpose from the left)
C * Q	(No transpose from the right), and
C * Q'	(Conjugate transpose from the right)

Q is a unitary matrix defined as the product of k Householder reflectors as

$$Q = H(k) * H(k-1) * \dots * H(1)$$

of order m if applying from the left, or n if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the QL factorization GEQLF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its conjugate transpose is to be applied.
- **m** – [in] rocblas\_int. m >= 0.  
Number of rows of matrix C.
- **n** – [in] rocblas\_int. n >= 0.  
Number of columns of matrix C.
- **k** – [in] rocblas\_int. k >= 0; k <= m if side is left, k <= n if side is right.  
The number of Householder reflectors that form Q.
- **A** – [in] pointer to type. Array on the GPU of size lda\*k.  
The i-th column has the Householder vector v(i) associated with H(i) as returned by GEQLF in the last k columns of its argument A.
- **lda** – [in] rocblas\_int. lda >= m if side is left, lda >= n if side is right.  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least k.  
The scalar factors of the Householder matrices H(i) as returned by GEQLF.
- **C** – [inout] pointer to type. Array on the GPU of size ldc\*n.  
On input, the matrix C. On output it is overwritten with Q\*C, C\*Q, Q'\*C, or C\*Q'.
- **ldc** – [in] rocblas\_int. ldc >= m.  
Leading dimension of C.

#### roc solver\_<type>unmbr()

rocblas\_status **roc solver\_zunmbr**(rocblas\_handle handle, const *rocblas\_storev* storev, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv, rocblas\_double\_complex \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_cunmbr**(rocblas\_handle handle, const *rocblas\_storev* storev, const rocblas\_side side, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int k, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv, rocblas\_float\_complex \*C, const rocblas\_int ldc)

UNMBR applies a complex matrix Q with orthonormal rows or columns to a general m-by-n matrix C.

If storev is column-wise, then the matrix Q has orthonormal columns. If storev is row-wise, then the matrix Q has orthonormal rows. The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q	*	C	(No transpose <b>from the left</b> )
Q'	*	C	(Conjugate transpose <b>from the left</b> )
C	*	Q	(No transpose <b>from the right</b> ), <b>and</b>
C	*	Q'	(Conjugate transpose <b>from the right</b> )

The order nq of unitary matrix Q is nq = m if applying from the left, or nq = n if applying from the right.

When storev is column-wise, if nq >= k, then Q is defined as the product of k Householder reflectors of order nq

$$Q = H(1) * H(2) * \dots * H(k),$$

and if nq < k, then Q is defined as the product

$$Q = H(1) * H(2) * \dots * H(nq-1).$$

When storev is row-wise, if nq > k, then Q is defined as the product of k Householder reflectors of order nq

$$Q = H(1) * H(2) * \dots * H(k),$$

and if n <= k, Q is defined as the product

$$Q = H(1) * H(2) * \dots * H(nq-1)$$

The Householder matrices H(i) are never stored, they are computed from its corresponding Householder vectors v(i) and scalars ipiv\_i as returned by GEBRD in its arguments A and tauq or taup.

### Parameters

- **handle** – [in] rocblas\_handle.
- **storev** – [in] *rocblas\_storev*.  
Specifies whether to work column-wise or row-wise.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its conjugate transpose is to be applied.
- **m** – [in] rocblas\_int. m >= 0.  
Number of rows of matrix C.
- **n** – [in] rocblas\_int. n >= 0.  
Number of columns of matrix C.
- **k** – [in] rocblas\_int. k >= 0.

The number of columns (if storev is column-wise) or rows (if row-wise) of the original matrix reduced by GEBRD.

- **A** – **[in]** pointer to type. Array on the GPU of size  $lda * \min(nq, k)$  if column-wise, or  $lda * nq$  if row-wise.

The  $i$ -th column (or row) has the Householder vector  $v(i)$  associated with  $H(i)$  as returned by GEBRD.

- **lda** – **[in]** rocblas\_int.  $lda \geq nq$  if column-wise, or  $lda \geq \min(nq, k)$  if row-wise.

Leading dimension of A.

- **ipiv** – **[in]** pointer to type. Array on the GPU of dimension at least  $\min(nq, k)$ .

The scalar factors of the Householder matrices  $H(i)$  as returned by GEBRD.

- **C** – **[inout]** pointer to type. Array on the GPU of size  $ldc * n$ .

On input, the matrix C. On output it is overwritten with  $Q * C$ ,  $C * Q$ ,  $Q' * C$ , or  $C * Q'$ .

- **ldc** – **[in]** rocblas\_int.  $ldc \geq m$ .

Leading dimension of C.

### roc solver\_<type>unmtr()

rocblas\_status **roc solver\_zunmtr**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_fill uplo, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv, rocblas\_double\_complex \*C, const rocblas\_int ldc)

rocblas\_status **roc solver\_cunmtr**(rocblas\_handle handle, const rocblas\_side side, const rocblas\_fill uplo, const rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv, rocblas\_float\_complex \*C, const rocblas\_int ldc)

UNMTR applies a unitary matrix Q to a general m-by-n matrix C.

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

Q * C	(No transpose <b>from the left</b> )
Q' * C	(Conjugate transpose <b>from the left</b> )
C * Q	(No transpose <b>from the right</b> ), <b>and</b>
C * Q'	(Conjugate transpose <b>from the right</b> )

The order  $nq$  of unitary matrix Q is  $nq = m$  if applying from the left, or  $nq = n$  if applying from the right.

Q is defined as the product of  $nq-1$  Householder reflectors of order  $nq$ . If uplo indicates upper, then Q has the form

$$Q = H(nq-1) * H(nq-2) * \dots * H(1).$$

On the other hand, if uplo indicates lower, then Q has the form

$$Q = H(1) * H(2) * \dots * H(nq-1)$$

The Householder matrices  $H(i)$  are never stored, they are computed from its corresponding Householder vectors  $v(i)$  and scalars  $ipiv_i$  as returned by HETRD in its arguments A and tau.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **side** – [in] rocblas\_side.  
Specifies from which side to apply Q.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the SYTRD factorization was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **trans** – [in] rocblas\_operation.  
Specifies whether the matrix Q or its conjugate transpose is to be applied.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
Number of rows of matrix C.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
Number of columns of matrix C.
- **A** – [in] pointer to type. Array on the GPU of size  $lda \times nq$ .  
On entry, the  $(i+1)$ -th column (if uplo indicates upper) or  $i$ -th column (if uplo indicates lower) has the Householder vector  $v(i)$  as returned by HETRD.
- **lda** – [in] rocblas\_int.  $lda \geq nq$ .  
Leading dimension of A.
- **ipiv** – [in] pointer to type. Array on the GPU of dimension at least  $nq-1$ .  
The scalar factors of the Householder matrices  $H(i)$  as returned by HETRD.
- **C** – [inout] pointer to type. Array on the GPU of size  $ldc \times n$ .  
On input, the matrix C. On output it is overwritten with  $Q \times C$ ,  $C \times Q$ ,  $Q' \times C$ , or  $C \times Q'$ .
- **ldc** – [in] rocblas\_int.  $ldc \geq m$ .  
Leading dimension of C.

## 2.7.4 LAPACK Functions

LAPACK routines solve complex Numerical Linear Algebra problems.

### 2.7.4.1 Triangular Factorizations

#### roc solver\_<type>potf2()

rocblas\_status **roc solver\_zpotf2**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_cpotf2**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_dpotf2**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_spotf2**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, rocblas\_int \*info)  
 POTF2 computes the Cholesky factorization of a real symmetric/complex Hermitian positive definite matrix A.  
 (This is the unblocked version of the algorithm).

The factorization has the form:

$$\begin{aligned} A &= U' * U, \text{ or} \\ A &= L * L' \end{aligned}$$

depending on the value of uplo. U is an upper triangular matrix and L is lower triangular.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
 Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **n** – [in] rocblas\_int. n >= 0.  
 The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU of dimension lda\*n.  
 On entry, the matrix A to be factored. On exit, the lower or upper triangular factor.
- **lda** – [in] rocblas\_int. lda >= n.  
 specifies the leading dimension of A.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
 If info = 0, successful factorization of matrix A. If info = i > 0, the leading minor of order i of A is not positive definite. The factorization stopped at this point.

#### roc solver\_<type>potf2\_batched()

rocblas\_status **roc solver\_zpotf2\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cpotf2\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dpotf2\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_spotf2\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

POTF2\_BATCHED computes the Cholesky factorization of a batch of real symmetric/complex Hermitian positive definite matrices.



(This is the unblocked version of the algorithm).

The factorization of matrix  $A_i$  in the batch has the form:

$$A_i = U_i' * U_i, \text{ or}$$

$$A_i = L_i * L_i'$$

depending on the value of `uplo`.  $U_i$  is an upper triangular matrix and  $L_i$  is lower triangular.

### Parameters

- **handle** – [in] `rocblas_handle`.
- **uplo** – [in] `rocblas_fill`.  
Specifies whether the factorization is upper or lower triangular. If `uplo` indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] `rocblas_int`.  $n \geq 0$ .  
The dimension of matrix  $A_i$ .
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda * n$ .  
On entry, the matrices  $A_i$  to be factored. On exit, the upper or lower triangular factors.
- **lda** – [in] `rocblas_int`.  $lda \geq n$ .  
specifies the leading dimension of  $A_i$ .
- **info** – [out] pointer to `rocblas_int`. Array of `batch_count` integers on the GPU.  
If `info_i = 0`, successful factorization of matrix  $A_i$ . If `info_i = j > 0`, the leading minor of order  $j$  of  $A_i$  is not positive definite. The  $i$ -th factorization stopped at this point.
- **batch\_count** – [in] `rocblas_int`. `batch_count`  $\geq 0$ .  
Number of matrices in the batch.

### `roc solver_<type>potf2_strided_batched()`

```
rocblas_status roc solver_zpotf2_strided_batched(rocblas_handle handle, const rocblas_fill uplo, const
rocblas_int n, rocblas_double_complex *A, const
rocblas_int lda, const rocblas_stride strideA, rocblas_int
*info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_cpotf2_strided_batched(rocblas_handle handle, const rocblas_fill uplo, const
rocblas_int n, rocblas_float_complex *A, const rocblas_int
lda, const rocblas_stride strideA, rocblas_int *info, const
rocblas_int batch_count)
```

```
rocblas_status roc solver_dpotf2_strided_batched(rocblas_handle handle, const rocblas_fill uplo, const
rocblas_int n, double *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_int *info, const rocblas_int
batch_count)
```

```
rocblas_status roc solver_spotf2_strided_batched(rocblas_handle handle, const rocblas_fill uplo, const
                                                    rocblas_int n, float *A, const rocblas_int lda, const
                                                    rocblas_stride strideA, rocblas_int *info, const rocblas_int
                                                    batch_count)
```

POTF2\_STRIDED\_BATCHED computes the Cholesky factorization of a batch of real symmetric/complex Hermitian positive definite matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix  $A_i$  in the batch has the form:

$$A_i = U_i' * U_i, \text{ or}$$

$$A_i = L_i * L_i'$$

depending on the value of uplo.  $U_i$  is an upper triangular matrix and  $L_i$  is lower triangular.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The dimension of matrix  $A_i$ .
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of strideA).  
On entry, the matrices  $A_i$  to be factored. On exit, the upper or lower triangular factors.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_i$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_i$  and the next one  $A_{(i+1)}$ . There is no restriction for the value of strideA. Normal use case is  $strideA \geq lda * n$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_i = 0$ , successful factorization of matrix  $A_i$ . If  $info_i = j > 0$ , the leading minor of order j of  $A_i$  is not positive definite. The i-th factorization stopped at this point.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

#### roc solver\_<type>potrf()

```
rocblas_status roc solver_zpotrf(rocblas_handle handle, const rocblas_fill uplo, const rocblas_int n,
                                   rocblas_double_complex *A, const rocblas_int lda, rocblas_int *info)
```

```
rocblas_status roc solver_cpotrf(rocblas_handle handle, const rocblas_fill uplo, const rocblas_int n,
                                   rocblas_float_complex *A, const rocblas_int lda, rocblas_int *info)
```

rocblas\_status **roc solver\_dpotrf**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_spotrf**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, rocblas\_int \*info)

POTRF computes the Cholesky factorization of a real symmetric/complex Hermitian positive definite matrix A. (This is the blocked version of the algorithm).

The factorization has the form:

$$\begin{aligned} A &= U' * U, \text{ or} \\ A &= L * L' \end{aligned}$$

depending on the value of uplo. U is an upper triangular matrix and L is lower triangular.

### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the matrix A to be factored. On exit, the lower or upper triangular factor.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of A.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If  $info = 0$ , successful factorization of matrix A. If  $info = i > 0$ , the leading minor of order i of A is not positive definite. The factorization stopped at this point.

### roc solver\_<type>potrf\_batched()

rocblas\_status **roc solver\_zpotrf\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cpotrf\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dpotrf\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

```
rocblas_status roc solver_spotrf_batched(rocblas_handle handle, const rocblas_fill uplo, const rocblas_int n,  
                                         float *const A[], const rocblas_int lda, rocblas_int *info, const  
                                         rocblas_int batch_count)
```

POTRF\_BATCHED computes the Cholesky factorization of a batch of real symmetric/complex Hermitian positive definite matrices.

(This is the blocked version of the algorithm).

The factorization of matrix  $A_i$  in the batch has the form:

$$A_i = U_i' * U_i, \text{ or}$$
$$A_i = L_i * L_i'$$

depending on the value of uplo.  $U_i$  is an upper triangular matrix and  $L_i$  is lower triangular.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The dimension of matrix  $A_i$ .
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda * n$ .  
On entry, the matrices  $A_i$  to be factored. On exit, the upper or lower triangular factors.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_i$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_i = 0$ , successful factorization of matrix  $A_i$ . If  $info_i = j > 0$ , the leading minor of order j of  $A_i$  is not positive definite. The i-th factorization stopped at this point.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

#### roc solver\_<type>potrf\_strided\_batched()

```
rocblas_status roc solver_zpotrf_strided_batched(rocblas_handle handle, const rocblas_fill uplo, const  
                                                  rocblas_int n, rocblas_double_complex *A, const  
                                                  rocblas_int lda, const rocblas_stride strideA, rocblas_int  
                                                  *info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_cpotrf_strided_batched(rocblas_handle handle, const rocblas_fill uplo, const  
                                                  rocblas_int n, rocblas_float_complex *A, const rocblas_int  
                                                  lda, const rocblas_stride strideA, rocblas_int *info, const  
                                                  rocblas_int batch_count)
```

rocblas\_status **roc solver\_dpotrff\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_spotrff\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

POTRF\_STRIDED\_BATCHED computes the Cholesky factorization of a batch of real symmetric/complex Hermitian positive definite matrices.

(This is the blocked version of the algorithm).

The factorization of matrix  $A_i$  in the batch has the form:

$$A_i = U_i' * U_i, \text{ or}$$

$$A_i = L_i * L_i'$$

depending on the value of uplo.  $U_i$  is an upper triangular matrix and  $L_i$  is lower triangular.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The dimension of matrix  $A_i$ .
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of strideA).  
On entry, the matrices  $A_i$  to be factored. On exit, the upper or lower triangular factors.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_i$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_i$  and the next one  $A_{(i+1)}$ . There is no restriction for the value of strideA. Normal use case is  $strideA \geq lda * n$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_i = 0$ , successful factorization of matrix  $A_i$ . If  $info_i = j > 0$ , the leading minor of order j of  $A_i$  is not positive definite. The i-th factorization stopped at this point.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

**roc solver\_<type>getf2()**

rocblas\_status **roc solver\_zgetf2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

rocblas\_status **roc solver\_cgetf2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

rocblas\_status **roc solver\_dgetf2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

rocblas\_status **roc solver\_sgetf2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

GETF2 computes the LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges.

(This is the unblocked Level-2-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with small and mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda * n$ .  
On entry, the m-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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [out] pointer to rocblas\_int. Array on the GPU of dimension  $\min(m,n)$ .  
The vector of pivot indices. Elements of ipiv are 1-based indices. For  $1 \leq i \leq \min(m,n)$ , the row i of the matrix was interchanged with row  $ipiv[i]$ . Matrix P of the factorization can be derived from ipiv.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If  $info = 0$ , successful exit. If  $info = i > 0$ , U is singular.  $U(i,i)$  is the first zero pivot.

**roc solver\_<type>getf2\_batched()**

rocblas\_status **roc solver\_zgetf2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgetf2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgetf2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgetf2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

GETF2\_BATCHED computes the LU factorization of a batch of general m-by-n matrices using partial pivoting with row interchanges.

(This is the unblocked Level-2-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with small and mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

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 (lower trapezoidal if  $m > n$ ), and  $U_i$  is upper triangular (upper trapezoidal if  $m < n$ ).

**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns 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  $lda * n$ .  
On entry, the m-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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **ipiv** – [out] pointer to rocblas\_int. Array on the GPU (the size depends on the value of strideP).

Contains the vectors of pivot indices `ipiv_i` (corresponding to `A_i`). Dimension of `ipiv_i` is  $\min(m,n)$ . Elements of `ipiv_i` are 1-based indices. For each instance `A_i` in the batch and for  $1 \leq j \leq \min(m,n)$ , 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`.

- **strideP** – [in] `rocblas_stride`.

Stride from the start of one vector `ipiv_i` to the next one `ipiv_(i+1)`. There is no restriction for the value of `strideP`. Normal use case is `strideP >= \min(m,n)`.

- **info** – [out] pointer to `rocblas_int`. Array of `batch_count` 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.

- **batch\_count** – [in] `rocblas_int`. `batch_count >= 0`.

Number of matrices in the batch.

### `roc solver_<type>getf2_strided_batched()`

```
rocblas_status roc solver_zgetf2_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, rocblas_double_complex *A, const
rocblas_int lda, const rocblas_stride strideA, rocblas_int
*ipiv, const rocblas_stride strideP, rocblas_int *info, const
rocblas_int batch_count)
```

```
rocblas_status roc solver_cgetf2_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, rocblas_float_complex *A, const rocblas_int
lda, const rocblas_stride strideA, rocblas_int *ipiv, const
rocblas_stride strideP, rocblas_int *info, const rocblas_int
batch_count)
```

```
rocblas_status roc solver_dgetf2_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, double *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_int *ipiv, const
rocblas_stride strideP, rocblas_int *info, const rocblas_int
batch_count)
```

```
rocblas_status roc solver_sgetf2_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, float *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_int *ipiv, const
rocblas_stride strideP, rocblas_int *info, const rocblas_int
batch_count)
```

GETF2\_STRIDED\_BATCHED computes the LU factorization of a batch of general `m`-by-`n` matrices using partial pivoting with row interchanges.

(This is the unblocked Level-2-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with small and mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

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 (lower trapezoidal if  $m > n$ ), and  $U_i$  is upper triangular (upper trapezoidal if  $m < n$ ).

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns 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 m-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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_i$  and 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 rocblas\_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  $\min(m, n)$ . Elements of  $ipiv_i$  are 1-based indices. For each instance  $A_i$  in the batch and for  $1 \leq j \leq \min(m, n)$ , 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  $ipiv_i$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_i$  to the next one  $ipiv_{(i+1)}$ . There is no restriction for the value of strideP. Normal use case is  $strideP \geq \min(m, n)$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count 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.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>getrf()

```
rocblas_status roc solver_zgetrf(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                rocblas_double_complex *A, const rocblas_int lda, rocblas_int *ipiv,
                                rocblas_int *info)
```

```
rocblas_status roc solver_cgetrf(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                rocblas_float_complex *A, const rocblas_int lda, rocblas_int *ipiv, rocblas_int
                                *info)
```

rocblas\_status **roc solver\_dgetrf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

rocblas\_status **roc solver\_sgetrf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

GETRF computes the LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges.

(This is the blocked Level-3-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda * n$ .  
On entry, the m-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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [out] pointer to rocblas\_int. Array on the GPU of dimension  $\min(m,n)$ .  
The vector of pivot indices. Elements of ipiv are 1-based indices. For  $1 \leq i \leq \min(m,n)$ , the row i of the matrix was interchanged with row ipiv[i]. Matrix P of the factorization can be derived from ipiv.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If info = 0, successful exit. If info =  $i > 0$ , U is singular. U(i,i) is the first zero pivot.

#### roc solver\_<type>getrf\_batched()

rocblas\_status **roc solver\_zgetrf\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgetrf\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgetrf\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgetrf\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

GETRF\_BATCHED computes the LU factorization of a batch of general m-by-n matrices using partial pivoting with row interchanges.

(This is the blocked Level-3-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

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 (lower trapezoidal if  $m > n$ ), and  $U_i$  is upper triangular (upper trapezoidal if  $m < n$ ).

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns 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  $lda * n$ .  
On entry, the m-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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **ipiv** – [out] pointer to rocblas\_int. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors of pivot indices  $ipiv_i$  (corresponding to  $A_i$ ). Dimension of  $ipiv_i$  is  $\min(m, n)$ . Elements of  $ipiv_i$  are 1-based indices. For each instance  $A_i$  in the batch and for  $1 \leq j \leq \min(m, n)$ , 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  $ipiv_i$ .
- **strideP** – [in] rocblas\_stride.

Stride from the start of one vector `ipiv_i` to the next one `ipiv_(i+1)`. There is no restriction for the value of `strideP`. Normal use case is `strideP >= min(m,n)`.

- **info** – [out] pointer to `rocblas_int`. Array of `batch_count` 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.

- **batch\_count** – [in] `rocblas_int`. `batch_count >= 0`.

Number of matrices in the batch.

### `roc solver_<type>getrf_strided_batched()`

```
rocblas_status roc solver_zgetrf_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, rocblas_double_complex *A, const
rocblas_int lda, const rocblas_stride strideA, rocblas_int
*ipiv, const rocblas_stride strideP, rocblas_int *info, const
rocblas_int batch_count)
```

```
rocblas_status roc solver_cgetrf_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, rocblas_float_complex *A, const rocblas_int
lda, const rocblas_stride strideA, rocblas_int *ipiv, const
rocblas_stride strideP, rocblas_int *info, const rocblas_int
batch_count)
```

```
rocblas_status roc solver_dgetrf_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, double *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_int *ipiv, const
rocblas_stride strideP, rocblas_int *info, const rocblas_int
batch_count)
```

```
rocblas_status roc solver_sgetrf_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, float *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_int *ipiv, const
rocblas_stride strideP, rocblas_int *info, const rocblas_int
batch_count)
```

GETRF\_STRIDED\_BATCHED computes the LU factorization of a batch of general  $m$ -by- $n$  matrices using partial pivoting with row interchanges.

(This is the blocked Level-3-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

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 (lower trapezoidal if  $m > n$ ), and  $U_i$  is upper triangular (upper trapezoidal if  $m < n$ ).

#### Parameters

- **handle** – [in] `rocblas_handle`.

- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns 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 m-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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_i$  and 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 rocblas\_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  $\min(m, n)$ . Elements of  $ipiv_i$  are 1-based indices. For each instance  $A_i$  in the batch and for  $1 \leq j \leq \min(m, n)$ , 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  $ipiv_i$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_i$  to the next one  $ipiv_{(i+1)}$ . There is no restriction for the value of strideP. Normal use case is  $strideP \geq \min(m, n)$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count 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.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### 2.7.4.2 Orthogonal Factorizations

#### roc solver\_<type>geqr2()

```
rocblas_status roc solver_zgeqr2(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                rocblas_double_complex *A, const rocblas_int lda, rocblas_double_complex
                                *ipiv)
```

```
rocblas_status roc solver_cgeqr2(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                rocblas_float_complex *A, const rocblas_int lda, rocblas_float_complex *ipiv)
```

```
rocblas_status roc solver_dgeqr2(rocblas_handle handle, const rocblas_int m, const rocblas_int n, double *A,
                                const rocblas_int lda, double *ipiv)
```

rocblas\_status **roc solver\_sgeqr2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*ipiv)

GEQR2 computes a QR factorization of a general m-by-n matrix A.

(This is the unblocked version of the algorithm).

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

$$Q = H(1) * H(2) * \dots * H(k), \text{ with } k = \min(m, n)$$

Each Householder matrix H(i), for  $i = 1, 2, \dots, k$ , is given by

$$H(i) = I - ipiv[i-1] * v(i) * v(i)'$$

where the first i-1 elements of the Householder vector v(i) are zero, and  $v(i)[i] = 1$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda * n$ .  
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  $m - i$  elements of vector v(i) for  $i = 1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [out] pointer to type. Array on the GPU of dimension  $\min(m, n)$ .  
The scalar factors of the Householder matrices H(i).

#### roc solver\_<type>geqr2\_batched()

rocblas\_status **roc solver\_zgeqr2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_double\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgeqr2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_float\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgeqr2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgeqr2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEQR2\_BATCHED computes the QR factorization of a batch of general m-by-n matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = Q_j * \begin{bmatrix} R_j \\ 0 \end{bmatrix}$$

where  $R_j$  is upper triangular (upper trapezoidal if  $m < n$ ), and  $Q_j$  is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(k), \text{ with } k = \min(m, n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - ipiv_j[i-1] * v_j(i) * v_j(i)'$$

where the first  $i-1$  elements of Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  in the batch.
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda * n$ .  
On entry, the m-by-n matrices  $A_j$  to be factored. On exit, the elements on and above the diagonal contain the factor  $R_j$ . The elements below the diagonal are the  $m - i$  elements of vector  $v_j(i)$  for  $i=1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $ipiv_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_j$  to the next one  $ipiv_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq \min(m, n)$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

**roc solver\_<type>geqr2\_strided\_batched()**

rocblas\_status **roc solver\_zgeqr2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_double\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgeqr2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_float\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgeqr2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgeqr2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEQR2\_STRIDED\_BATCHED computes the QR factorization of a batch of general m-by-n matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = Q_j * \begin{bmatrix} R_j \\ \mathbf{0} \end{bmatrix}$$

where  $R_j$  is upper triangular (upper trapezoidal if  $m < n$ ), and  $Q_j$  is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(k), \text{ with } k = \min(m, n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i) * v_j(i)'$$

where the first  $i-1$  elements of Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  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  $A_j$  to be factored. On exit, the elements on and above the diagonal contain the factor  $R_j$ . The elements below the diagonal are the  $m - i$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .

- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . 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  $ipiv_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_j$  to the next one  $ipiv_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq \min(m, n)$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>geqrf()

rocblas\_status **roc solver\_zgeqrf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cgeqrf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

rocblas\_status **roc solver\_dgeqrf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc solver\_sgeqrf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*ipiv)

GEQRF computes a QR factorization of a general m-by-n matrix A.

(This is the blocked version of the algorithm).

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

$$Q = H(1) * H(2) * \dots * H(k), \text{ with } k = \min(m, n)$$

Each Householder matrix  $H(i)$ , for  $i = 1, 2, \dots, k$ , is given by

$$H(i) = I - ipiv[i-1] * v(i) * v(i)'$$

where the first  $i-1$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
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  $m - i$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [out] pointer to type. Array on the GPU of dimension  $\min(m, n)$ .  
The scalar factors of the Householder matrices  $H(i)$ .

### roc solver\_<type>geqrf\_batched()

```
rocblas_status roc solver_zgeqrf_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                         rocblas_double_complex *const A[], const rocblas_int lda,
                                         rocblas_double_complex *ipiv, const rocblas_stride strideP, const
                                         rocblas_int batch_count)
```

```
rocblas_status roc solver_cgeqrf_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                         rocblas_float_complex *const A[], const rocblas_int lda,
                                         rocblas_float_complex *ipiv, const rocblas_stride strideP, const
                                         rocblas_int batch_count)
```

```
rocblas_status roc solver_dgeqrf_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                         double *const A[], const rocblas_int lda, double *ipiv, const
                                         rocblas_stride strideP, const rocblas_int batch_count)
```

```
rocblas_status roc solver_sgeqrf_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n, float
                                         *const A[], const rocblas_int lda, float *ipiv, const rocblas_stride
                                         strideP, const rocblas_int batch_count)
```

GEQRF\_BATCHED computes the QR factorization of a batch of general  $m$ -by- $n$  matrices.

(This is the blocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = Q_j * \begin{bmatrix} R_j \\ 0 \end{bmatrix}$$

where  $R_j$  is upper triangular (upper trapezoidal if  $m < n$ ), and  $Q_j$  is a  $m$ -by- $m$  orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(k), \text{ with } k = \min(m,n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i) * v_j(i)'$$

where the first  $i-1$  elements of vector Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  in the batch.
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $\text{lda} * n$ .  
On entry, the  $m$ -by- $n$  matrices  $A_j$  to be factored. On exit, the elements on and above the diagonal contain the factor  $R_j$ . The elements below the diagonal are the  $m - i$  elements of vector  $v_j(i)$  for  $i=1, 2, \dots, \min(m,n)$ .
- **lda** – [in] rocblas\_int.  $\text{lda} \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideP`).  
Contains the vectors  $\text{ipiv}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{ipiv}_j$  to the next one  $\text{ipiv}_{j+1}$ . There is no restriction for the value of `strideP`. Normal use is  $\text{strideP} \geq \min(m,n)$ .
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>geqrf\_strided\_batched()

```
rocblas_status roc solver_zgeqrf_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, rocblas_double_complex *A, const rocblas_int lda, const rocblas_stride strideA,
rocblas_double_complex *ipiv, const rocblas_stride
strideP, const rocblas_int batch_count)
```

```
rocblas_status roc solver_cgeqrf_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, rocblas_float_complex *A, const rocblas_int
lda, const rocblas_stride strideA, rocblas_float_complex
*ipiv, const rocblas_stride strideP, const rocblas_int
batch_count)
```

rocblas\_status **roc solver\_dgeqrf\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgeqrf\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEQRF\_STRIDED\_BATCHED computes the QR factorization of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = Q_j * \begin{bmatrix} R_j \\ 0 \end{bmatrix}$$

where  $R_j$  is upper triangular (upper trapezoidal if  $m < n$ ), and  $Q_j$  is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(k), \text{ with } k = \min(m, n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i) * v_j(i)'$$

where the first  $i-1$  elements of vector Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  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  $A_j$  to be factored. On exit, the elements on and above the diagonal contain the factor  $R_j$ . The elements below the diagonal are the  $m - i$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $\text{lda} \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $\text{strideA} \geq \text{lda} * n$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\text{ipiv}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.

Stride from the start of one vector `ipiv_j` to the next one `ipiv_(j+1)`. There is no restriction for the value of `strideP`. Normal use is `strideP >= min(m,n)`.

- **batch\_count** – [in] `rocblas_int`. `batch_count >= 0`.

Number of matrices in the batch.

### roc solver\_<type>geql2()

`rocblas_status roc solver_zgeql2(rocblas_handle handle, const rocblas_int m, const rocblas_int n, rocblas_double_complex *A, const rocblas_int lda, rocblas_double_complex *ipiv)`

`rocblas_status roc solver_cgeql2(rocblas_handle handle, const rocblas_int m, const rocblas_int n, rocblas_float_complex *A, const rocblas_int lda, rocblas_float_complex *ipiv)`

`rocblas_status roc solver_dgeql2(rocblas_handle handle, const rocblas_int m, const rocblas_int n, double *A, const rocblas_int lda, double *ipiv)`

`rocblas_status roc solver_sgeql2(rocblas_handle handle, const rocblas_int m, const rocblas_int n, float *A, const rocblas_int lda, float *ipiv)`

GEQL2 computes a QL factorization of a general m-by-n matrix A.

(This is the unblocked version of the algorithm).

The factorization has the form

$$A = Q * \begin{bmatrix} \mathbb{0} \\ L \end{bmatrix}$$

where L is lower triangular (lower trapezoidal if  $m < n$ ), and Q is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q = H(k) * \dots * H(2) * H(1), \text{ with } k = \min(m, n)$$

Each Householder matrix H(i), for  $i = 1, 2, \dots, k$ , is given by

$$H(i) = I - ipiv[i-1] * v(i) * v(i)'$$

where the last m-i elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i] = 1$ .

#### Parameters

- **handle** – [in] `rocblas_handle`.
- **m** – [in] `rocblas_int`.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] `rocblas_int`.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension `lda*n`.

On entry, the m-by-n matrix to be factored. On exit, the elements on and below the (m-n)th subdiagonal (when  $m \geq n$ ) or the (n-m)th superdiagonal (when  $n > m$ ) contain the factor

L; the elements above the sub/superdiagonal are the  $i - 1$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .

- **lda** – [in] rocblas\_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 scalar factors of the Householder matrices  $H(i)$ .

### roc solver\_<type>geql2\_batched()

rocblas\_status **roc solver\_zgeql2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_double\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgeql2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_float\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgeql2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgeql2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEQL2\_BATCHED computes the QL factorization of a batch of general m-by-n matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = Q_j * \begin{bmatrix} I & \\ & L_j \end{bmatrix}$$

where  $L_j$  is lower triangular (lower trapezoidal if  $m < n$ ), and  $Q_j$  is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(k) * \dots * H_j(2) * H_j(1), \text{ with } k = \min(m, n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i) * v_j(i)'$$

where the last  $m-i$  elements of Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.

- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  in the batch.
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda \cdot n$ .  
On entry, the  $m$ -by- $n$  matrices  $A_j$  to be factored. On exit, the elements on and below the  $(m-n)$ th subdiagonal (when  $m \geq n$ ) or the  $(n-m)$ th superdiagonal (when  $n > m$ ) contain the factor  $L_j$ ; the elements above the sub/superdiagonal are the  $i - 1$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $ipiv_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_j$  to the next one  $ipiv_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq \min(m, n)$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>geql2\_strided\_batched()

rocblas\_status **roc solver\_zgeql2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_double\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgeql2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_float\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgeql2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgeql2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEQL2\_STRIDED\_BATCHED computes the QL factorization of a batch of general  $m$ -by- $n$  matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = Q_j * \begin{bmatrix} 0 \\ L_j \end{bmatrix}$$

where  $L_j$  is lower triangular (lower trapezoidal if  $m < n$ ), and  $Q_j$  is a  $m$ -by- $m$  orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(k) * \dots * H_j(2) * H_j(1), \text{ with } k = \min(m, n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i) * v_j(i)'$$

where the last  $m-i$  elements of Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  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  $A_j$  to be factored. On exit, the elements on and below the  $(m-n)$ th subdiagonal (when  $m \geq n$ ) or the  $(n-m)$ th superdiagonal (when  $n > m$ ) contain the factor  $L_j$ ; the elements above the sub/superdiagonal are the  $i - 1$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $\text{strideA} \geq lda * n$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\text{ipiv}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{ipiv}_j$  to the next one  $\text{ipiv}_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $\text{strideP} \geq \min(m, n)$ .
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.



**roc solver\_<type>geqlf()**

rocblas\_status **roc solver\_zgeqlf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cgeqlf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

rocblas\_status **roc solver\_dgeqlf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc solver\_sgeqlf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*ipiv)

GEQLF computes a QL factorization of a general m-by-n matrix A.

(This is the blocked version of the algorithm).

The factorization has the form

$$A = Q * \begin{bmatrix} \mathbb{O} \\ L \end{bmatrix}$$

where L is lower triangular (lower trapezoidal if  $m < n$ ), and Q is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q = H(k) * \dots * H(2) * H(1), \text{ with } k = \min(m, n)$$

Each Householder matrix H(i), for  $i = 1, 2, \dots, k$ , is given by

$$H(i) = I - ipiv[i-1] * v(i) * v(i)'$$

where the last m-i elements of the Householder vector v(i) are zero, and  $v(i)[i] = 1$ .

**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension lda\*n.  
On entry, the m-by-n matrix to be factored. On exit, the elements on and below the (m-n)th subdiagonal (when  $m \geq n$ ) or the (n-m)th superdiagonal (when  $n > m$ ) contain the factor L; the elements above the sub/superdiagonal are the  $i - 1$  elements of vector v(i) for  $i = 1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [out] pointer to type. Array on the GPU of dimension  $\min(m, n)$ .  
The scalar factors of the Householder matrices H(i).

**roc solver\_<type>geqlf\_batched()**

rocblas\_status **roc solver\_zgeqlf\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_double\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgeqlf\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_float\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgeqlf\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgeqlf\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEQLF\_BATCHED computes the QL factorization of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = Q_j * \begin{bmatrix} 0 \\ L_j \end{bmatrix}$$

where  $L_j$  is lower triangular (lower trapezoidal if  $m < n$ ), and  $Q_j$  is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(k) * \dots * H_j(2) * H_j(1), \text{ with } k = \min(m, n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i) * v_j(i)'$$

where the last  $m-i$  elements of vector Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  in the batch.
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $\text{lda} * n$ .

On entry, the m-by-n matrices  $A_j$  to be factored. On exit, the elements on and below the (m-n)th subdiagonal (when  $m \geq n$ ) or the (n-m)th superdiagonal (when  $n > m$ ) contain the factor  $L_j$ ; the elements above the sub/superdiagonal are the  $i - 1$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .

- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $ipiv_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_j$  to the next one  $ipiv_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq \min(m,n)$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>geqlf\_strided\_batched()

rocblas\_status **roc solver\_zgeqlf\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_double\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgeqlf\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_float\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgeqlf\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgeqlf\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEQLF\_STRIDED\_BATCHED computes the QL factorization of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = Q_j * \begin{bmatrix} \mathbb{0} \\ L_j \end{bmatrix}$$

where  $L_j$  is lower triangular (lower trapezoidal if  $m < n$ ), and  $Q_j$  is a m-by-m orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(k) * \dots * H_j(2) * H_j(1), \text{ with } k = \min(m,n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, batch\_count$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i) * v_j(i)'$$

where the last  $m-i$  elements of vector Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  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  $A_j$  to be factored. On exit, the elements on and below the  $(m-n)$ th subdiagonal (when  $m \geq n$ ) or the  $(n-m)$ th superdiagonal (when  $n > m$ ) contain the factor  $L_j$ ; the elements above the sub/superdiagonal are the  $i - 1$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $\text{strideA} \geq lda * n$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\text{ipiv}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{ipiv}_j$  to the next one  $\text{ipiv}_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $\text{strideP} \geq \min(m, n)$ .
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>gelq2()

```
rocblas_status roc solver_zgelq2(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                rocblas_double_complex *A, const rocblas_int lda, rocblas_double_complex
                                *ipiv)
```

```
rocblas_status roc solver_cgelq2(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                rocblas_float_complex *A, const rocblas_int lda, rocblas_float_complex *ipiv)
```

```
rocblas_status roc solver_dgelq2(rocblas_handle handle, const rocblas_int m, const rocblas_int n, double *A,
                                const rocblas_int lda, double *ipiv)
```

```
rocblas_status roc solver_sgelq2(rocblas_handle handle, const rocblas_int m, const rocblas_int n, float *A, const
                                rocblas_int lda, float *ipiv)
```

GELQ2 computes a LQ factorization of a general  $m$ -by- $n$  matrix  $A$ .

(This is the unblocked version of the algorithm).

The factorization has the form

$$A = [ L \ 0 ] * Q$$

where L is lower triangular (lower trapezoidal if  $m > n$ ), and Q is a n-by-n orthogonal/unitary matrix represented as the product of Householder matrices

$$Q = H(k) * H(k-1) * \dots * H(1), \text{ with } k = \min(m,n)$$

Each Householder matrix H(i), for  $i = 1, 2, \dots, k$ , is given by

$$H(i) = I - \text{ipiv}[i-1] * v(i)' * v(i)$$

where the first  $i-1$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $\text{lda} * n$ .  
On entry, the m-by-n matrix to be factored. On exit, the elements on and below the diagonal contain the factor L; the elements above the diagonal are the  $n - i$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, \min(m,n)$ .
- **lda** – [in] rocblas\_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 scalar factors of the Householder matrices H(i).

### roc solver\_<type>gelq2\_batched()

```
rocblas_status roc solver_zgelq2_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                         rocblas_double_complex *const A[], const rocblas_int lda,
                                         rocblas_double_complex *ipiv, const rocblas_stride strideP, const
                                         rocblas_int batch_count)
```

```
rocblas_status roc solver_cgelq2_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                         rocblas_float_complex *const A[], const rocblas_int lda,
                                         rocblas_float_complex *ipiv, const rocblas_stride strideP, const
                                         rocblas_int batch_count)
```

```
rocblas_status roc solver_dgelq2_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                         double *const A[], const rocblas_int lda, double *ipiv, const
                                         rocblas_stride strideP, const rocblas_int batch_count)
```

rocblas\_status **roc solver\_sgelq2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GELQ2\_BATCHED computes the LQ factorization of a batch of general m-by-n matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = [ L_j \ 0 ] * Q_j$$

where  $L_j$  is lower triangular (lower trapezoidal if  $m > n$ ), and  $Q_j$  is a n-by-n orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(k) * H_j(k-1) * \dots * H_j(1), \text{ with } k = \min(m,n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i)' * v_j(i)$$

where the first  $i-1$  elements of Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  in the batch.
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $\text{lda} * n$ .  
On entry, the m-by-n matrices  $A_j$  to be factored. On exit, the elements on and below the diagonal contain the factor  $L_j$ . The elements above the diagonal are the  $n - i$  elements of vector  $v_j(i)$  for  $i=1, 2, \dots, \min(m,n)$ .
- **lda** – [in] rocblas\_int.  $\text{lda} \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\text{ipiv}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{ipiv}_j$  to the next one  $\text{ipiv}_{j+1}$ . There is no restriction for the value of strideP. Normal use is  $\text{strideP} \geq \min(m,n)$ .
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

**roc solver\_<type>gelq2\_strided\_batched()**

rocblas\_status **roc solver\_zgelq2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_double\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgelq2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_float\_complex \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgelq2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgelq2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GELQ2\_STRIDED\_BATCHED computes the LQ factorization of a batch of general m-by-n matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = [ L_j \ 0 ] * Q_j$$

where  $L_j$  is lower triangular (lower trapezoidal if  $m > n$ ), and  $Q_j$  is a n-by-n orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(k) * H_j(k-1) * \dots * H_j(1), \text{ with } k = \min(m, n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i)' * v_j(i)$$

where the first  $i-1$  elements of vector Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  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  $A_j$  to be factored. On exit, the elements on and below the diagonal contain the factor  $L_j$ . The elements above the diagonal are the  $n - i$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .

- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . 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  $ipiv_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_j$  to the next one  $ipiv_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq \min(m, n)$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>gelqf()

rocblas\_status **roc solver\_zgelqf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*ipiv)

rocblas\_status **roc solver\_cg gelqf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*ipiv)

rocblas\_status **roc solver\_d gelqf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*ipiv)

rocblas\_status **roc solver\_s gelqf**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*ipiv)

GELQF computes a LQ factorization of a general m-by-n matrix A.

(This is the blocked version of the algorithm).

The factorization has the form

$$A = \begin{bmatrix} L & 0 \end{bmatrix} * Q$$

where L is lower triangular (lower trapezoidal if  $m > n$ ), and Q is a n-by-n orthogonal/unitary matrix represented as the product of Householder matrices

$$Q = H(k) * H(k-1) * \dots * H(1), \text{ with } k = \min(m, n)$$

Each Householder matrix  $H(i)$ , for  $i = 1, 2, \dots, k$ , is given by

$$H(i) = I - ipiv[i-1] * v(i)' * v(i)$$

where the first  $i-1$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i] = 1$ .



**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the m-by-n matrix to be factored. On exit, the elements on and below the diagonal contain the factor L; the elements above the diagonal are the  $n - i$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **ipiv** – [out] pointer to type. Array on the GPU of dimension  $\min(m, n)$ .  
The scalar factors of the Householder matrices  $H(i)$ .

**roc solver\_<type>gelqf\_batched()**

```
rocblas_status roc solver_zgelqf_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                       rocblas_double_complex *const A[], const rocblas_int lda,
                                       rocblas_double_complex *ipiv, const rocblas_stride strideP, const
                                       rocblas_int batch_count)
```

```
rocblas_status roc solver_cgelqf_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                       rocblas_float_complex *const A[], const rocblas_int lda,
                                       rocblas_float_complex *ipiv, const rocblas_stride strideP, const
                                       rocblas_int batch_count)
```

```
rocblas_status roc solver_dgelqf_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                       double *const A[], const rocblas_int lda, double *ipiv, const
                                       rocblas_stride strideP, const rocblas_int batch_count)
```

```
rocblas_status roc solver_sgelqf_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int n, float
                                       *const A[], const rocblas_int lda, float *ipiv, const rocblas_stride
                                       strideP, const rocblas_int batch_count)
```

GELQF\_BATCHED computes the LQ factorization of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = [ L_j \ 0 ] * Q_j$$

where  $L_j$  is lower triangular (lower trapezoidal if  $m > n$ ), and  $Q_j$  is a n-by-n orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(k) * H_j(k-1) * \dots * H_j(1), \text{ with } k = \min(m, n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i)' * v_j(i)$$

where the first  $i-1$  elements of Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  in the batch.
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $\text{lda} * n$ .  
On entry, the  $m$ -by- $n$  matrices  $A_j$  to be factored. On exit, the elements on and below the diagonal contain the factor  $L_j$ . The elements above the diagonal are the  $n - i$  elements of vector  $v_j(i)$  for  $i=1, 2, \dots, \min(m, n)$ .
- **lda** – [in] rocblas\_int.  $\text{lda} \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideP`).  
Contains the vectors  $\text{ipiv}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{ipiv}_j$  to the next one  $\text{ipiv}_{j+1}$ . There is no restriction for the value of `strideP`. Normal use is  $\text{strideP} \geq \min(m, n)$ .
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

#### roc solver\_<type>gelqf\_strided\_batched()

```
rocblas_status roc solver_zgelqf_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, rocblas_double_complex *A, const
rocblas_int lda, const rocblas_stride strideA,
rocblas_double_complex *ipiv, const rocblas_stride
strideP, const rocblas_int batch_count)
```

```
rocblas_status roc solver_cgelqf_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, rocblas_float_complex *A, const rocblas_int
lda, const rocblas_stride strideA, rocblas_float_complex
*ipiv, const rocblas_stride strideP, const rocblas_int
batch_count)
```

rocblas\_status **roc solver\_dgelsqf\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgelsqf\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*ipiv, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GELQF\_STRIDED\_BATCHED computes the LQ factorization of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The factorization of matrix  $A_j$  in the batch has the form

$$A_j = [ L_j \ 0 ] * Q_j$$

where  $L_j$  is lower triangular (lower trapezoidal if  $m > n$ ), and  $Q_j$  is a n-by-n orthogonal/unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(k) * H_j(k-1) * \dots * H_j(1), \text{ with } k = \min(m,n)$$

Each Householder matrices  $H_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , and  $i = 1, 2, \dots, k$ , is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] * v_j(i)' * v_j(i)$$

where the first  $i-1$  elements of vector Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  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  $A_j$  to be factored. On exit, the elements on and below the diagonal contain the factor  $L_j$ . The elements above the diagonal are the  $n - i$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, \min(m,n)$ .
- **lda** – [in] rocblas\_int.  $\text{lda} \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $\text{strideA} \geq \text{lda} * n$ .
- **ipiv** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\text{ipiv}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{ipiv}_j$  to the next one  $\text{ipiv}_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $\text{strideP} \geq \min(m,n)$ .

- **batch\_count** – [in] rocblas\_int. batch\_count >= 0.

Number of matrices in the batch.

### 2.7.4.3 Problem and matrix reductions

#### roc solver\_<type>gebd2()

rocblas\_status **roc solver\_zgebd2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, double \*D, double \*E, rocblas\_double\_complex \*tauq, rocblas\_double\_complex \*taup)

rocblas\_status **roc solver\_cgebd2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, float \*D, float \*E, rocblas\_float\_complex \*tauq, rocblas\_float\_complex \*taup)

rocblas\_status **roc solver\_dgebd2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*D, double \*E, double \*tauq, double \*taup)

rocblas\_status **roc solver\_sgebd2**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*D, float \*E, float \*tauq, float \*taup)

GEBD2 computes the bidiagonal form of a general m-by-n matrix A.

(This is the unblocked version of the algorithm).

The bidiagonal form is given by:

$$B = Q' * A * P$$

where B is upper bidiagonal if  $m \geq n$  and lower bidiagonal if  $m < n$ , and Q and P are orthogonal/unitary matrices represented as the product of Householder matrices

$$Q = H(1) * H(2) * \dots * H(n) \text{ and } P = G(1) * G(2) * \dots * G(n-1), \text{ if } m \geq n, \text{ or} \\ Q = H(1) * H(2) * \dots * H(m-1) \text{ and } P = G(1) * G(2) * \dots * G(m), \text{ if } m < n$$

Each Householder matrix H(i) and G(i) is given by

$$H(i) = I - \tau_{q[i-1]} * v(i) * v(i)', \text{ and} \\ G(i) = I - \tau_{p[i-1]} * u(i) * u(i)'$$

If  $m \geq n$ , the first i-1 elements of the Householder vector v(i) are zero, and v(i)[i] = 1; while the first i elements of the Householder vector u(i) are zero, and u(i)[i+1] = 1. If  $m < n$ , the first i elements of the Householder vector v(i) are zero, and v(i)[i+1] = 1; while the first i-1 elements of the Householder vector u(i) are zero, and u(i)[i] = 1.

#### Parameters

- **handle** – [in] rocblas\_handle.

- **m** – [in] rocblas\_int.  $m \geq 0$ .

The number of rows of the matrix A.

- **n** – [in] rocblas\_int.  $n \geq 0$ .

The number of columns of the matrix A.

- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the  $m$ -by- $n$  matrix to be factored. On exit, the elements on the diagonal and superdiagonal (if  $m \geq n$ ), or subdiagonal (if  $m < n$ ) contain the bidiagonal form  $B$ . If  $m \geq n$ , the elements below the diagonal are the  $m - i$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, n$ , and the elements above the superdiagonal are the  $n - i - 1$  elements of vector  $u(i)$  for  $i = 1, 2, \dots, n-1$ . If  $m < n$ , the elements below the subdiagonal are the  $m - i - 1$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, m-1$ , and the elements above the diagonal are the  $n - i$  elements of vector  $u(i)$  for  $i = 1, 2, \dots, m$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
specifies the leading dimension of  $A$ .
- **D** – [out] pointer to real type. Array on the GPU of dimension  $\min(m, n)$ .  
The diagonal elements of  $B$ .
- **E** – [out] pointer to real type. Array on the GPU of dimension  $\min(m, n) - 1$ .  
The off-diagonal elements of  $B$ .
- **tauq** – [out] pointer to type. Array on the GPU of dimension  $\min(m, n)$ .  
The scalar factors of the Householder matrices  $H(i)$ .
- **taup** – [out] pointer to type. Array on the GPU of dimension  $\min(m, n)$ .  
The scalar factors of the Householder matrices  $G(i)$ .

### roc solver\_<type>gebd2\_batched()

rocblas\_status **roc solver\_zgebd2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_double\_complex \*tauq, const rocblas\_stride strideQ, rocblas\_double\_complex \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgebd2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_float\_complex \*tauq, const rocblas\_stride strideQ, rocblas\_float\_complex \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgebd2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, double \*tauq, const rocblas\_stride strideQ, double \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgebd2\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, float \*tauq, const rocblas\_stride strideQ, float \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEBD2\_BATCHED computes the bidiagonal form of a batch of general m-by-n matrices.

(This is the unblocked version of the algorithm).

The bidiagonal form is given by:

$$B_j = Q_j' * A_j * P_j$$

where  $B_j$  is upper bidiagonal if  $m \geq n$  and lower bidiagonal if  $m < n$ , and  $Q_j$  and  $P_j$  are orthogonal/unitary matrices represented as the product of Householder matrices

$$\begin{aligned} Q_j &= H_j(1) * H_j(2) * \dots * H_j(n) \quad \text{and} \quad P_j = G_j(1) * G_j(2) * \dots * G_j(n-1), \\ &\leftarrow \text{if } m \geq n, \text{ or} \\ Q_j &= H_j(1) * H_j(2) * \dots * H_j(m-1) \quad \text{and} \quad P_j = G_j(1) * G_j(2) * \dots * G_j(m), \\ &\leftarrow \text{if } m < n \end{aligned}$$

Each Householder matrix  $H_j(i)$  and  $G_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , is given by

$$\begin{aligned} H_j(i) &= I - \text{tauq}_j[i-1] * v_j(i) * v_j(i)', \quad \text{and} \\ G_j(i) &= I - \text{taup}_j[i-1] * u_j(i) * u_j(i)' \end{aligned}$$

If  $m \geq n$ , the first  $i-1$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ ; while the first  $i$  elements of the Householder vector  $u_j(i)$  are zero, and  $u_j(i)[i+1] = 1$ . If  $m < n$ , the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ ; while the first  $i-1$  elements of the Householder vector  $u_j(i)$  are zero, and  $u_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  in the batch.
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $\text{lda} * n$ .  
On entry, the m-by-n matrices  $A_j$  to be factored. On exit, the elements on the diagonal and superdiagonal (if  $m \geq n$ ), or subdiagonal (if  $m < n$ ) contain the bidiagonal form  $B_j$ . If  $m \geq n$ , the elements below the diagonal are the  $m - i$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, n$ , and the elements above the superdiagonal are the  $n - i - 1$  elements of vector  $u_j(i)$  for  $i = 1, 2, \dots, n-1$ . If  $m < n$ , the elements below the subdiagonal are the  $m - i - 1$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, m-1$ , and the elements above the diagonal are the  $n - i$  elements of vector  $u_j(i)$  for  $i = 1, 2, \dots, m$ .
- **lda** – [in] rocblas\_int.  $\text{lda} \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **D** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideD).  
The diagonal elements of  $B_j$ .

- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $\text{strideD} \geq \min(m,n)$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideE).  
The off-diagonal elements of  $B_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $\text{strideE} \geq \min(m,n)-1$ .
- **tauq** – [out] pointer to type. Array on the GPU (the size depends on the value of strideQ).  
Contains the vectors  $\text{tauq}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideQ** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{tauq}_j$  to the next one  $\text{tauq}_{(j+1)}$ . There is no restriction for the value of strideQ. Normal use is  $\text{strideQ} \geq \min(m,n)$ .
- **taup** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\text{taup}_j$  of scalar factors of the Householder matrices  $G_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{taup}_j$  to the next one  $\text{taup}_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $\text{strideP} \geq \min(m,n)$ .
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>geb2\_strided\_batched()

rocblas\_status **roc solver\_zgeb2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_double\_complex \*tauq, const rocblas\_stride strideQ, rocblas\_double\_complex \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgeb2\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_float\_complex \*tauq, const rocblas\_stride strideQ, rocblas\_float\_complex \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

```
rocblas_status roc solver_dgebd2_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, double *A, const rocblas_int lda, const
rocblas_stride strideA, double *D, const rocblas_stride
strideD, double *E, const rocblas_stride strideE, double
*tauq, const rocblas_stride strideQ, double *taup, const
rocblas_stride strideP, const rocblas_int batch_count)
```

```
rocblas_status roc solver_sgebd2_strided_batched(rocblas_handle handle, const rocblas_int m, const
rocblas_int n, float *A, const rocblas_int lda, const
rocblas_stride strideA, float *D, const rocblas_stride
strideD, float *E, const rocblas_stride strideE, float *tauq,
const rocblas_stride strideQ, float *taup, const
rocblas_stride strideP, const rocblas_int batch_count)
```

GEBD2\_STRIDED\_BATCHED computes the bidiagonal form of a batch of general m-by-n matrices.

(This is the unblocked version of the algorithm).

The bidiagonal form is given by:

$$B_j = Q_j' * A_j * P_j$$

where  $B_j$  is upper bidiagonal if  $m \geq n$  and lower bidiagonal if  $m < n$ , and  $Q_j$  and  $P_j$  are orthogonal/unitary matrices represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(n) \quad \text{and} \quad P_j = G_j(1) * G_j(2) * \dots * G_j(n-1),$$

$\rightarrow$  if  $m \geq n$ , or

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(m-1) \quad \text{and} \quad P_j = G_j(1) * G_j(2) * \dots * G_j(m),$$

$\rightarrow$  if  $m < n$

Each Householder matrix  $H_j(i)$  and  $G_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , is given by

$$H_j(i) = I - \tau_{q_j}[i-1] * v_j(i) * v_j(i)', \quad \text{and}$$

$$G_j(i) = I - \tau_{p_j}[i-1] * u_j(i) * u_j(i)'$$

If  $m \geq n$ , the first  $i-1$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ ; while the first  $i$  elements of the Householder vector  $u_j(i)$  are zero, and  $u_j(i)[i+1] = 1$ . If  $m < n$ , the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ ; while the first  $i-1$  elements of the Householder vector  $u_j(i)$  are zero, and  $u_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  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  $A_j$  to be factored. On exit, the elements on the diagonal and superdiagonal (if  $m \geq n$ ), or subdiagonal (if  $m < n$ ) contain the bidiagonal form  $B_j$ . If  $m \geq n$ , the elements below the diagonal are the  $m - i$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, n$ , and the elements above the superdiagonal are the  $n - i - 1$  elements of vector  $u_j(i)$  for  $i = 1, 2, \dots, n-1$ . If  $m < n$ , the elements below the subdiagonal are the  $m - i - 1$  elements of vector



$v_j(i)$  for  $i = 1, 2, \dots, m-1$ , and the elements above the diagonal are the  $n - i$  elements of vector  $u_j(i)$  for  $i = 1, 2, \dots, m$ .

- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $strideA \geq lda * n$ .
- **D** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideD).  
The diagonal elements of  $B_j$ .
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $strideD \geq \min(m, n)$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideE).  
The off-diagonal elements of  $B_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $strideE \geq \min(m, n) - 1$ .
- **tauq** – [out] pointer to type. Array on the GPU (the size depends on the value of strideQ).  
Contains the vectors  $\tau_{q_j}$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideQ** – [in] rocblas\_stride.  
Stride from the start of one vector  $\tau_{q_j}$  to the next one  $\tau_{q_{(j+1)}}$ . There is no restriction for the value of strideQ. Normal use is  $strideQ \geq \min(m, n)$ .
- **taup** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\tau_{p_j}$  of scalar factors of the Householder matrices  $G_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\tau_{p_j}$  to the next one  $\tau_{p_{(j+1)}}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq \min(m, n)$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>gebrd()

```
rocblas_status roc solver_zgebrd(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                rocblas_double_complex *A, const rocblas_int lda, double *D, double *E,
                                rocblas_double_complex *tauq, rocblas_double_complex *taup)
```

```
rocblas_status roc solver_cgebrd(rocblas_handle handle, const rocblas_int m, const rocblas_int n,
                                rocblas_float_complex *A, const rocblas_int lda, float *D, float *E,
                                rocblas_float_complex *tauq, rocblas_float_complex *taup)
```

```
rocblas_status roc solver_dgebrd(rocblas_handle handle, const rocblas_int m, const rocblas_int n, double *A,
                                const rocblas_int lda, double *D, double *E, double *tauq, double *taup)
```

```
rocblas_status roc solver_sgebrd(rocblas_handle handle, const rocblas_int m, const rocblas_int n, float *A, const
                                rocblas_int lda, float *D, float *E, float *tauq, float *taup)
```

GEBRD computes the bidiagonal form of a general m-by-n matrix A.

(This is the blocked version of the algorithm).

The bidiagonal form is given by:

$$B = Q' * A * P$$

where B is upper bidiagonal if  $m \geq n$  and lower bidiagonal if  $m < n$ , and Q and P are orthogonal/unitary matrices represented as the product of Householder matrices

$$Q = H(1) * H(2) * \dots * H(n) \quad \text{and} \quad P = G(1) * G(2) * \dots * G(n-1), \quad \text{if } m \geq n, \quad \text{or} \\ Q = H(1) * H(2) * \dots * H(m-1) \quad \text{and} \quad P = G(1) * G(2) * \dots * G(m), \quad \text{if } m < n$$

Each Householder matrix H(i) and G(i) is given by

$$H(i) = I - \tau q[i-1] * v(i) * v(i)', \quad \text{and} \\ G(i) = I - \tau p[i-1] * u(i) * u(i)'$$

If  $m \geq n$ , the first  $i-1$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i] = 1$ ; while the first  $i$  elements of the Householder vector  $u(i)$  are zero, and  $u(i)[i+1] = 1$ . If  $m < n$ , the first  $i$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i+1] = 1$ ; while the first  $i-1$  elements of the Householder vector  $u(i)$  are zero, and  $u(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda * n$ .  
On entry, the m-by-n matrix to be factored. On exit, the elements on the diagonal and superdiagonal (if  $m \geq n$ ), or subdiagonal (if  $m < n$ ) contain the bidiagonal form B. If  $m \geq n$ , the elements below the diagonal are the  $m - i$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, n$ , and the elements above the superdiagonal are the  $n - i - 1$  elements of vector  $u(i)$  for  $i = 1, 2, \dots, n-1$ . If  $m < n$ , the elements below the subdiagonal are the  $m - i - 1$  elements of vector  $v(i)$  for  $i = 1, 2, \dots, m-1$ , and the elements above the diagonal are the  $n - i$  elements of vector  $u(i)$  for  $i = 1, 2, \dots, m$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
specifies the leading dimension of A.
- **D** – [out] pointer to real type. Array on the GPU of dimension  $\min(m, n)$ .  
The diagonal elements of B.
- **E** – [out] pointer to real type. Array on the GPU of dimension  $\min(m, n) - 1$ .  
The off-diagonal elements of B.

- **tauq** – [out] pointer to type. Array on the GPU of dimension  $\min(m,n)$ .  
The scalar factors of the Householder matrices  $H(i)$ .
- **taup** – [out] pointer to type. Array on the GPU of dimension  $\min(m,n)$ .  
The scalar factors of the Householder matrices  $G(i)$ .

### roc solver\_<type>gebrd\_batched()

rocblas\_status **roc solver\_zgebrd\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_double\_complex \*tauq, const rocblas\_stride strideQ, rocblas\_double\_complex \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgebrd\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_float\_complex \*tauq, const rocblas\_stride strideQ, rocblas\_float\_complex \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgebrd\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, double \*tauq, const rocblas\_stride strideQ, double \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgebrd\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, float \*tauq, const rocblas\_stride strideQ, float \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEBRD\_BATCHED computes the bidiagonal form of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The bidiagonal form is given by:

$$B_j = Q_j' * A_j * P_j$$

where  $B_j$  is upper bidiagonal if  $m \geq n$  and lower bidiagonal if  $m < n$ , and  $Q_j$  and  $P_j$  are orthogonal/unitary matrices represented as the product of Householder matrices

$$\begin{aligned} Q_j &= H_j(1) * H_j(2) * \dots * H_j(n) \quad \text{and} \quad P_j = G_j(1) * G_j(2) * \dots * G_j(n-1), \quad \leftarrow \\ &\rightarrow \text{if } m \geq n, \quad \text{or} \\ Q_j &= H_j(1) * H_j(2) * \dots * H_j(m-1) \quad \text{and} \quad P_j = G_j(1) * G_j(2) * \dots * G_j(m), \quad \leftarrow \\ &\rightarrow \text{if } m < n \end{aligned}$$

Each Householder matrix  $H_j(i)$  and  $G_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , is given by

$$H_j(i) = I - \tau_{q_j}[i-1] * v_j(i) * v_j(i)', \text{ and}$$

$$G_j(i) = I - \tau_{p_j}[i-1] * u_j(i) * u_j(i)'$$

If  $m \geq n$ , the first  $i-1$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ ; while the first  $i$  elements of the Householder vector  $u_j(i)$  are zero, and  $u_j(i)[i+1] = 1$ . If  $m < n$ , the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ ; while the first  $i-1$  elements of the Householder vector  $u_j(i)$  are zero, and  $u_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  in the batch.
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda*n$ .  
On entry, the  $m$ -by- $n$  matrices  $A_j$  to be factored. On exit, the elements on the diagonal and superdiagonal (if  $m \geq n$ ), or subdiagonal (if  $m < n$ ) contain the bidiagonal form  $B_j$ . If  $m \geq n$ , the elements below the diagonal are the  $m - i$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, n$ , and the elements above the superdiagonal are the  $n - i - 1$  elements of vector  $u_j(i)$  for  $i = 1, 2, \dots, n-1$ . If  $m < n$ , the elements below the subdiagonal are the  $m - i - 1$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, m-1$ , and the elements above the diagonal are the  $n - i$  elements of vector  $u_j(i)$  for  $i = 1, 2, \dots, m$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **D** – [out] pointer to real type. Array on the GPU (the size depends on the value of `strideD`).  
The diagonal elements of  $B_j$ .
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{j+1}$ . There is no restriction for the value of `strideD`. Normal use case is `strideD`  $\geq \min(m,n)$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of `strideE`).  
The off-diagonal elements of  $B_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{j+1}$ . There is no restriction for the value of `strideE`. Normal use case is `strideE`  $\geq \min(m,n)-1$ .
- **tauq** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideQ`).  
Contains the vectors  $\tau_{q_j}$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideQ** – [in] rocblas\_stride.  
Stride from the start of one vector  $\tau_{q_j}$  to the next one  $\tau_{q_{j+1}}$ . There is no restriction for the value of `strideQ`. Normal use is `strideQ`  $\geq \min(m,n)$ .
- **taup** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideP`).  
Contains the vectors  $\tau_{p_j}$  of scalar factors of the Householder matrices  $G_j(i)$ .

- **strideP** – [in] rocblas\_stride.

Stride from the start of one vector `taup_j` to the next one `taup_(j+1)`. There is no restriction for the value of `strideP`. Normal use is `strideP >= min(m,n)`.

- **batch\_count** – [in] rocblas\_int. `batch_count >= 0`.

Number of matrices in the batch.

### roc solver\_<type>gebrd\_strided\_batched()

rocblas\_status **roc solver\_zgebrd\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_double\_complex \*tauq, const rocblas\_stride strideQ, rocblas\_double\_complex \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgebrd\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_float\_complex \*tauq, const rocblas\_stride strideQ, rocblas\_float\_complex \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgebrd\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, double \*tauq, const rocblas\_stride strideQ, double \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgebrd\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, float \*tauq, const rocblas\_stride strideQ, float \*taup, const rocblas\_stride strideP, const rocblas\_int batch\_count)

GEBRD\_STRIDED\_BATCHED computes the bidiagonal form of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The bidiagonal form is given by:

$$B_j = Q_j' * A_j * P_j$$

where  $B_j$  is upper bidiagonal if  $m \geq n$  and lower bidiagonal if  $m < n$ , and  $Q_j$  and  $P_j$  are orthogonal/unitary matrices represented as the product of Householder matrices

```

Q_j = H_j(1) * H_j(2) * ... * H_j(n)  and P_j = G_j(1) * G_j(2) * ... * G_j(n-1),
↪ if m >= n, or
Q_j = H_j(1) * H_j(2) * ... * H_j(m-1) and P_j = G_j(1) * G_j(2) * ... * G_j(m),
↪ if m < n

```

Each Householder matrix  $H_j(i)$  and  $G_j(i)$ , for  $j = 1, 2, \dots, \text{batch\_count}$ , is given by

```

H_j(i) = I - tauq_j[i-1] * v_j(i) * v_j(i)', and
G_j(i) = I - taup_j[i-1] * u_j(i) * u_j(i)'

```

If  $m \geq n$ , the first  $i-1$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ ; while the first  $i$  elements of the Householder vector  $u_j(i)$  are zero, and  $u_j(i)[i+1] = 1$ . If  $m < n$ , the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ ; while the first  $i-1$  elements of the Householder vector  $u_j(i)$  are zero, and  $u_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all the matrices  $A_j$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all the matrices  $A_j$  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  $A_j$  to be factored. On exit, the elements on the diagonal and superdiagonal (if  $m \geq n$ ), or subdiagonal (if  $m < n$ ) contain the bidiagonal form  $B_j$ . If  $m \geq n$ , the elements below the diagonal are the  $m - i$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, n$ , and the elements above the superdiagonal are the  $n - i - 1$  elements of vector  $u_j(i)$  for  $i = 1, 2, \dots, n-1$ . If  $m < n$ , the elements below the subdiagonal are the  $m - i - 1$  elements of vector  $v_j(i)$  for  $i = 1, 2, \dots, m-1$ , and the elements above the diagonal are the  $n - i$  elements of vector  $u_j(i)$  for  $i = 1, 2, \dots, m$ .
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $\text{strideA} \geq lda * n$ .
- **D** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideD).  
The diagonal elements of  $B_j$ .
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $\text{strideD} \geq \min(m, n)$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideE).  
The off-diagonal elements of  $B_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $\text{strideE} \geq \min(m, n) - 1$ .

- **tauq** – [out] pointer to type. Array on the GPU (the size depends on the value of strideQ). Contains the vectors tauq\_j of scalar factors of the Householder matrices H\_j(i).
- **strideQ** – [in] rocblas\_stride. Stride from the start of one vector tauq\_j to the next one tauq\_(j+1). There is no restriction for the value of strideQ. Normal use is strideQ >= min(m,n).
- **taup** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP). Contains the vectors taup\_j of scalar factors of the Householder matrices G\_j(i).
- **strideP** – [in] rocblas\_stride. Stride from the start of one vector taup\_j to the next one taup\_(j+1). There is no restriction for the value of strideP. Normal use is strideP >= min(m,n).
- **batch\_count** – [in] rocblas\_int. batch\_count >= 0. Number of matrices in the batch.

### roc solver\_<type>sytd2()

rocblas\_status **roc solver\_dsyt d2**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*D, double \*E, double \*tau)

rocblas\_status **roc solver\_ssytd2**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*D, float \*E, float \*tau)

SYTD2 computes the tridiagonal form of a real symmetric matrix A.

(This is the unblocked version of the algorithm).

The tridiagonal form is given by:

$$T = Q' * A * Q$$

where T is symmetric tridiagonal and Q is an orthogonal matrix represented as the product of Householder matrices

$$Q = H(1) * H(2) * \dots * H(n-1) \text{ if uplo indicates lower, or} \\ Q = H(n-1) * H(n-2) * \dots * H(1) \text{ if uplo indicates upper.}$$

Each Householder matrix H(i) is given by

$$H(i) = I - \tau[i] * v(i) * v(i)'$$

where tau[i] is the corresponding Householder scalar. When uplo indicates lower, the first i elements of the Householder vector v(i) are zero, and v(i)[i+1] = 1. If uplo indicates upper, the last n-i elements of the Householder vector v(i) are zero, and v(i)[i] = 1.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.

Specifies whether the upper or lower part of the symmetric matrix A is stored. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.

- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the matrix to be factored. On exit, if upper, then the elements on the diagonal and superdiagonal contain the tridiagonal form T; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v(i)$  stored as columns. If lower, then the elements on the diagonal and subdiagonal contain the tridiagonal form T; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v(i)$  stored as columns.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of A.
- **D** – [out] pointer to type. Array on the GPU of dimension  $n$ .  
The diagonal elements of T.
- **E** – [out] pointer to type. Array on the GPU of dimension  $n-1$ .  
The off-diagonal elements of T.
- **tau** – [out] pointer to type. Array on the GPU of dimension  $n-1$ .  
The scalar factors of the Householder matrices  $H(i)$ .

### roc solver\_<type>sytd2\_batched()

rocblas\_status **roc solver\_dsyt d2\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, double \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssytd2\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, float \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

SYTD2\_BATCHED computes the tridiagonal form of a batch of real symmetric matrices  $A_j$ .

(This is the unblocked version of the algorithm).

The tridiagonal form of  $A_j$  is given by:

$$T_j = Q_j' * A_j * Q_j, \text{ for } j = 1, 2, \dots, \text{batch\_count}$$

where  $T_j$  is symmetric tridiagonal and  $Q_j$  is an orthogonal matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(n-1) \text{ if uplo indicates lower, or } \\ Q_j = H_j(n-1) * H_j(n-2) * \dots * H_j(1) \text{ if uplo indicates upper.}$$

Each Householder matrix  $H_j(i)$  is given by

$$H_j(i) = I - \text{tau}_j[i] * v_j(i) * v_j(i)'$$



where  $\tau_j[i]$  is the corresponding Householder scalar. When `uplo` indicates lower, the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ . If `uplo` indicates upper, the last  $n-i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the symmetric matrix  $A_j$  is stored. If `uplo` indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrices  $A_j$ .
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda*n$ .  
On entry, the matrices  $A_j$  to be factored. On exit, if `uplo`, then the elements on the diagonal and superdiagonal of  $A_j$  contain the tridiagonal form  $T_j$ ; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns. If lower, then the elements on the diagonal and subdiagonal contain the tridiagonal form  $T_j$ ; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_j$ .
- **D** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideD`).  
The diagonal elements of  $T_j$ .
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{j+1}$ . There is no restriction for the value of `strideD`. Normal use case is `strideD`  $\geq n$ .
- **E** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideE`).  
The off-diagonal elements of  $T_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{j+1}$ . There is no restriction for the value of `strideE`. Normal use case is `strideE`  $\geq n-1$ .
- **tau** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideP`).  
Contains the vectors  $\tau_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\tau_j$  to the next one  $\tau_{j+1}$ . There is no restriction for the value of `strideP`. Normal use is `strideP`  $\geq n-1$ .
- **batch\_count** – [in] rocblas\_int. `batch_count`  $\geq 0$ .  
Number of matrices in the batch.

**roc solver\_<type>sytd2\_strided\_batched()**

rocblas\_status **roc solver\_dsytd2\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, double \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssytd2\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, float \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

SYTD2\_STRIDED\_BATCHED computes the tridiagonal form of a batch of real symmetric matrices  $A_j$ .

(This is the unblocked version of the algorithm).

The tridiagonal form of  $A_j$  is given by:

$$T_j = Q_j' * A_j * Q_j, \text{ for } j = 1, 2, \dots, \text{batch\_count}$$

where  $T_j$  is symmetric tridiagonal and  $Q_j$  is an orthogonal matrix represented as the product of Householder matrices

$$\begin{aligned} Q_j &= H_j(1) * H_j(2) * \dots * H_j(n-1) \text{ if uplo indicates lower, or} \\ Q_j &= H_j(n-1) * H_j(n-2) * \dots * H_j(1) \text{ if uplo indicates upper.} \end{aligned}$$

Each Householder matrix  $H_j(i)$  is given by

$$H_j(i) = I - \tau_j[i] * v_j(i) * v_j(i)'$$

where  $\tau_j[i]$  is the corresponding Householder scalar. When uplo indicates lower, the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ . If uplo indicates upper, the last  $n-i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

**Parameters**

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the symmetric matrix  $A_j$  is stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrices  $A_j$ .
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of strideA).

On entry, the matrices  $A_j$  to be factored. On exit, if upper, then the elements on the diagonal and superdiagonal of  $A_j$  contain the tridiagonal form  $T_j$ ; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns. If lower, then the elements on the diagonal and subdiagonal contain the tridiagonal form  $T_j$ ; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns.

- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $strideA \geq lda * n$ .
- **D** – [out] pointer to type. Array on the GPU (the size depends on the value of strideD).  
The diagonal elements of  $T_j$ .
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $strideD \geq n$ .
- **E** – [out] pointer to type. Array on the GPU (the size depends on the value of strideE).  
The off-diagonal elements of  $T_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $strideE \geq n - 1$ .
- **tau** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\tau_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\tau_j$  to the next one  $\tau_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq n - 1$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>hetd2()

rocblas\_status **roc solver\_zhetd2**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, double \*D, double \*E, rocblas\_double\_complex \*tau)

rocblas\_status **roc solver\_chetd2**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, float \*D, float \*E, rocblas\_float\_complex \*tau)

HETD2 computes the tridiagonal form of a complex hermitian matrix A.

(This is the unblocked version of the algorithm).

The tridiagonal form is given by:

$$T = Q' * A * Q$$

where T is hermitian tridiagonal and Q is an unitary matrix represented as the product of Householder matrices

$$Q = H(1) * H(2) * \dots * H(n-1) \text{ if uplo indicates lower, or} \\ Q = H(n-1) * H(n-2) * \dots * H(1) \text{ if uplo indicates upper.}$$

Each Householder matrix  $H(i)$  is given by

$$H(i) = I - \tau[i] * v(i) * v(i)'$$

where  $\tau[i]$  is the corresponding Householder scalar. When `uplo` indicates lower, the first  $i$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i+1] = 1$ . If `uplo` indicates upper, the last  $n-i$  elements of the Householder vector  $v(i)$  are zero, and  $v(i)[i] = 1$ .

#### Parameters

- **handle** – [in] `rocblas_handle`.
- **uplo** – [in] `rocblas_fill`.  
Specifies whether the upper or lower part of the hermitian matrix  $A$  is stored. If `uplo` indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] `rocblas_int`.  $n \geq 0$ .  
The number of rows and columns of the matrix  $A$ .
- **A** – [inout] pointer to type. Array on the GPU of dimension `lda*n`.  
On entry, the matrix to be factored. On exit, if `uplo`, then the elements on the diagonal and superdiagonal contain the tridiagonal form  $T$ ; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v(i)$  stored as columns. If `lower`, then the elements on the diagonal and subdiagonal contain the tridiagonal form  $T$ ; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v(i)$  stored as columns.
- **lda** – [in] `rocblas_int`. `lda`  $\geq n$ .  
specifies the leading dimension of  $A$ .
- **D** – [out] pointer to real type. Array on the GPU of dimension  $n$ .  
The diagonal elements of  $T$ .
- **E** – [out] pointer to real type. Array on the GPU of dimension  $n-1$ .  
The off-diagonal elements of  $T$ .
- **tau** – [out] pointer to type. Array on the GPU of dimension  $n-1$ .  
The scalar factors of the Householder matrices  $H(i)$ .

#### `roc solver_<type>hetd2_batched()`

```
rocblas_status roc solver_zhetd2_batched(rocblas_handle handle, const rocblas_fill uplo, const rocblas_int n,
rocblas_double_complex *const A[], const rocblas_int lda, double
*D, const rocblas_stride strideD, double *E, const rocblas_stride
strideE, rocblas_double_complex *tau, const rocblas_stride strideP,
const rocblas_int batch_count)
```

```
rocblas_status roc solver_chetd2_batched(rocblas_handle handle, const rocblas_fill uplo, const rocblas_int n,
rocblas_float_complex *const A[], const rocblas_int lda, float *D,
const rocblas_stride strideD, float *E, const rocblas_stride strideE,
rocblas_float_complex *tau, const rocblas_stride strideP, const
rocblas_int batch_count)
```

HETD2\_BATCHED computes the tridiagonal form of a batch of complex hermitian matrices  $A_j$ .

(This is the unblocked version of the algorithm).

The tridiagonal form of  $A_j$  is given by:

$$T_j = Q_j' * A_j * Q_j, \text{ for } j = 1, 2, \dots, \text{batch\_count}$$

where  $T_j$  is hermitian tridiagonal and  $Q_j$  is a unitary matrix represented as the product of Householder matrices

$$\begin{aligned} Q_j &= H_j(1) * H_j(2) * \dots * H_j(n-1) \text{ if } \text{uplo} \text{ indicates lower, or} \\ Q_j &= H_j(n-1) * H_j(n-2) * \dots * H_j(1) \text{ if } \text{uplo} \text{ indicates upper.} \end{aligned}$$

Each Householder matrix  $H_j(i)$  is given by

$$H_j(i) = I - \text{tau}_j[i] * v_j(i) * v_j(i)'$$

where  $\text{tau}_j[i]$  is the corresponding Householder scalar. When  $\text{uplo}$  indicates lower, the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ . If  $\text{uplo}$  indicates upper, the last  $n-i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the hermitian matrix  $A_j$  is stored. If  $\text{uplo}$  indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrices  $A_j$ .
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $\text{lda} * n$ .  
On entry, the matrices  $A_j$  to be factored. On exit, if upper, then the elements on the diagonal and superdiagonal of  $A_j$  contain the tridiagonal form  $T_j$ ; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns. If lower, then the elements on the diagonal and subdiagonal contain the tridiagonal form  $T_j$ ; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns.
- **lda** – [in] rocblas\_int.  $\text{lda} \geq n$ .  
specifies the leading dimension of  $A_j$ .
- **D** – [out] pointer to real type. Array on the GPU (the size depends on the value of  $\text{strideD}$ ).  
The diagonal elements of  $T_j$ .
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of  $\text{strideD}$ . Normal use case is  $\text{strideD} \geq n$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of  $\text{strideE}$ ).  
The off-diagonal elements of  $T_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of  $\text{strideE}$ . Normal use case is  $\text{strideE} \geq n-1$ .
- **tau** – [out] pointer to type. Array on the GPU (the size depends on the value of  $\text{strideP}$ ).  
Contains the vectors  $\text{tau}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .

- **strideP** – [in] rocblas\_stride.

Stride from the start of one vector  $\tau_j$  to the next one  $\tau_{j+1}$ . There is no restriction for the value of strideP. Normal use is  $\text{strideP} \geq n-1$ .

- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .

Number of matrices in the batch.

### roc solver\_<type>hetd2\_strided\_batched()

rocblas\_status **roc solver\_zhetd2\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_double\_complex \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_chetd2\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_float\_complex \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

HETD2\_STRIDED\_BATCHED computes the tridiagonal form of a batch of complex hermitian matrices  $A_j$ .

(This is the unblocked version of the algorithm).

The tridiagonal form of  $A_j$  is given by:

$$T_j = Q_j' * A_j * Q_j, \text{ for } j = 1, 2, \dots, \text{batch\_count}$$

where  $T_j$  is hermitian tridiagonal and  $Q_j$  is a unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(n-1) \text{ if uplo indicates lower, or } \\ Q_j = H_j(n-1) * H_j(n-2) * \dots * H_j(1) \text{ if uplo indicates upper.}$$

Each Householder matrix  $H_j(i)$  is given by

$$H_j(i) = I - \tau_j[i] * v_j(i) * v_j(i)'$$

where  $\tau_j[i]$  is the corresponding Householder scalar. When uplo indicates lower, the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ . If uplo indicates upper, the last  $n-i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

#### Parameters

- **handle** – [in] rocblas\_handle.

- **uplo** – [in] rocblas\_fill.

Specifies whether the upper or lower part of the hermitian matrix  $A_j$  is stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.

- **n** – [in] rocblas\_int.  $n \geq 0$ .

The number of rows and columns of the matrices  $A_j$ .

- **A** – **[inout]** pointer to type. Array on the GPU (the size depends on the value of strideA).  
On entry, the matrices  $A_j$  to be factored. On exit, if upper, then the elements on the diagonal and superdiagonal of  $A_j$  contain the tridiagonal form  $T_j$ ; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns. If lower, then the elements on the diagonal and subdiagonal contain the tridiagonal form  $T_j$ ; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns.
- **lda** – **[in]** rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_j$ .
- **strideA** – **[in]** rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $strideA \geq lda * n$ .
- **D** – **[out]** pointer to real type. Array on the GPU (the size depends on the value of strideD).  
The diagonal elements of  $T_j$ .
- **strideD** – **[in]** rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $strideD \geq n$ .
- **E** – **[out]** pointer to real type. Array on the GPU (the size depends on the value of strideE).  
The off-diagonal elements of  $T_j$ .
- **strideE** – **[in]** rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $strideE \geq n-1$ .
- **tau** – **[out]** pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\tau_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – **[in]** rocblas\_stride.  
Stride from the start of one vector  $\tau_j$  to the next one  $\tau_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq n-1$ .
- **batch\_count** – **[in]** rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>sytrd()

rocblas\_status **roc solver\_dsytrd**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*D, double \*E, double \*tau)

rocblas\_status **roc solver\_ssytrd**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*D, float \*E, float \*tau)

SYTRD computes the tridiagonal form of a real symmetric matrix A.

(This is the blocked version of the algorithm).

The tridiagonal form is given by:

$$T = Q' * A * Q$$

where T is symmetric tridiagonal and Q is an orthogonal matrix represented as the product of Householder matrices

$$Q = H(1) * H(2) * \dots * H(n-1) \text{ if uplo indicates lower, or}$$

$$Q = H(n-1) * H(n-2) * \dots * H(1) \text{ if uplo indicates upper.}$$

Each Householder matrix H(i) is given by

$$H(i) = I - \tau[i] * v(i) * v(i)'$$

where tau[i] is the corresponding Householder scalar. When uplo indicates lower, the first i elements of the Householder vector v(i) are zero, and v(i)[i+1] = 1. If uplo indicates upper, the last n-i elements of the Householder vector v(i) are zero, and v(i)[i] = 1.

### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the symmetric matrix A is stored. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **n** – [in] rocblas\_int. n >= 0.  
The number of rows and columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension lda\*n.  
On entry, the matrix to be factored. On exit, if upper, then the elements on the diagonal and superdiagonal contain the tridiagonal form T; the elements above the superdiagonal contain the i-1 non-zero elements of vectors v(i) stored as columns. If lower, then the elements on the diagonal and subdiagonal contain the tridiagonal form T; the elements below the subdiagonal contain the n-i-1 non-zero elements of vectors v(i) stored as columns.
- **lda** – [in] rocblas\_int. lda >= n.  
specifies the leading dimension of A.
- **D** – [out] pointer to type. Array on the GPU of dimension n.  
The diagonal elements of T.
- **E** – [out] pointer to type. Array on the GPU of dimension n-1.  
The off-diagonal elements of T.
- **tau** – [out] pointer to type. Array on the GPU of dimension n-1.  
The scalar factors of the Householder matrices H(i).



**roc solver\_<type>sytrd\_batched()**

rocblas\_status **roc solver\_dsytrd\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, double \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssytrd\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, float \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

SYTRD\_BATCHED computes the tridiagonal form of a batch of real symmetric matrices  $A_j$ .

(This is the blocked version of the algorithm).

The tridiagonal form of  $A_j$  is given by:

$$T_j = Q_j' * A_j * Q_j, \text{ for } j = 1, 2, \dots, \text{batch\_count}$$

where  $T_j$  is symmetric tridiagonal and  $Q_j$  is an orthogonal matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(n-1) \text{ if uplo indicates lower, or } \\ Q_j = H_j(n-1) * H_j(n-2) * \dots * H_j(1) \text{ if uplo indicates upper.}$$

Each Householder matrix  $H_j(i)$  is given by

$$H_j(i) = I - \tau_j[i] * v_j(i) * v_j(i)'$$

where  $\tau_j[i]$  is the corresponding Householder scalar. When uplo indicates lower, the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ . If uplo indicates upper, the last  $n-i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

**Parameters**

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the symmetric matrix  $A_j$  is stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrices  $A_j$ .
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda * n$ .  
On entry, the matrices  $A_j$  to be factored. On exit, if upper, then the elements on the diagonal and superdiagonal of  $A_j$  contain the tridiagonal form  $T_j$ ; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns. If lower, then the elements on the diagonal and subdiagonal contain the tridiagonal form  $T_j$ ; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_j$ .

- **D** – [out] pointer to type. Array on the GPU (the size depends on the value of strideD).  
The diagonal elements of  $T_j$ .
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $\text{strideD} \geq n$ .
- **E** – [out] pointer to type. Array on the GPU (the size depends on the value of strideE).  
The off-diagonal elements of  $T_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $\text{strideE} \geq n-1$ .
- **tau** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\tau_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\tau_j$  to the next one  $\tau_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $\text{strideP} \geq n-1$ .
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>sytrd\_strided\_batched()

rocblas\_status **roc solver\_dsytrd\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, double \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssytrd\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, float \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

SYTRD\_STRIDED\_BATCHED computes the tridiagonal form of a batch of real symmetric matrices  $A_j$ .

(This is the blocked version of the algorithm).

The tridiagonal form of  $A_j$  is given by:

$$T_j = Q_j' * A_j * Q_j, \text{ for } j = 1, 2, \dots, \text{batch\_count}$$

where  $T_j$  is symmetric tridiagonal and  $Q_j$  is an orthogonal matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(n-1) \text{ if uplo indicates lower, or } \\ Q_j = H_j(n-1) * H_j(n-2) * \dots * H_j(1) \text{ if uplo indicates upper.}$$

Each Householder matrix  $H_j(i)$  is given by

$$H_j(i) = I - \tau_j[i] * v_j(i) * v_j(i)'$$

where  $\tau_j[i]$  is the corresponding Householder scalar. When `uplo` indicates lower, the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ . If `uplo` indicates upper, the last  $n-i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the symmetric matrix  $A_j$  is stored. If `uplo` indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrices  $A_j$ .
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of `strideA`).  
On entry, the matrices  $A_j$  to be factored. On exit, if `uplo`, then the elements on the diagonal and superdiagonal of  $A_j$  contain the tridiagonal form  $T_j$ ; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns. If `lower`, then the elements on the diagonal and subdiagonal contain the tridiagonal form  $T_j$ ; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of `strideA`. Normal use case is `strideA`  $\geq lda * n$ .
- **D** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideD`).  
The diagonal elements of  $T_j$ .
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of `strideD`. Normal use case is `strideD`  $\geq n$ .
- **E** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideE`).  
The off-diagonal elements of  $T_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of `strideE`. Normal use case is `strideE`  $\geq n-1$ .
- **tau** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideP`).  
Contains the vectors  $\tau_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\tau_j$  to the next one  $\tau_{(j+1)}$ . There is no restriction for the value of `strideP`. Normal use is `strideP`  $\geq n-1$ .

- **batch\_count** – [in] rocblas\_int. batch\_count >= 0.

Number of matrices in the batch.

### roc solver\_<type>hetrd()

rocblas\_status **roc solver\_zhetrd**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, double \*D, double \*E, rocblas\_double\_complex \*tau)

rocblas\_status **roc solver\_chetrd**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, float \*D, float \*E, rocblas\_float\_complex \*tau)

HETRD computes the tridiagonal form of a complex hermitian matrix A.

(This is the blocked version of the algorithm).

The tridiagonal form is given by:

$$T = Q' * A * Q$$

where T is hermitian tridiagonal and Q is an unitary matrix represented as the product of Householder matrices

$$Q = H(1) * H(2) * \dots * H(n-1) \text{ if uplo indicates lower, or} \\ Q = H(n-1) * H(n-2) * \dots * H(1) \text{ if uplo indicates upper.}$$

Each Householder matrix H(i) is given by

$$H(i) = I - \tau[i] * v(i) * v(i)'$$

where tau[i] is the corresponding Householder scalar. When uplo indicates lower, the first i elements of the Householder vector v(i) are zero, and v(i)[i+1] = 1. If uplo indicates upper, the last n-i elements of the Householder vector v(i) are zero, and v(i)[i] = 1.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the hermitian matrix A is stored. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **n** – [in] rocblas\_int. n >= 0.  
The number of rows and columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension lda\*n.  
On entry, the matrix to be factored. On exit, if upper, then the elements on the diagonal and superdiagonal contain the tridiagonal form T; the elements above the superdiagonal contain the i-1 non-zero elements of vectors v(i) stored as columns. If lower, then the elements on the diagonal and subdiagonal contain the tridiagonal form T; the elements below the subdiagonal contain the n-i-1 non-zero elements of vectors v(i) stored as columns.
- **lda** – [in] rocblas\_int. lda >= n.  
specifies the leading dimension of A.

- **D** – [out] pointer to real type. Array on the GPU of dimension n.  
The diagonal elements of T.
- **E** – [out] pointer to real type. Array on the GPU of dimension n-1.  
The off-diagonal elements of T.
- **tau** – [out] pointer to type. Array on the GPU of dimension n-1.  
The scalar factors of the Householder matrices H(i).

### roc solver\_<type>hetrd\_batched()

rocblas\_status **roc solver\_zhetrd\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_double\_complex \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_chetrd\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_float\_complex \*tau, const rocblas\_stride strideP, const rocblas\_int batch\_count)

HETRD\_BATCHED computes the tridiagonal form of a batch of complex hermitian matrices  $A_j$ .

(This is the blocked version of the algorithm).

The tridiagonal form of  $A_j$  is given by:

$$T_j = Q_j' * A_j * Q_j, \text{ for } j = 1, 2, \dots, \text{batch\_count}$$

where  $T_j$  is hermitian tridiagonal and  $Q_j$  is a unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(n-1) \text{ if uplo indicates lower, or } \\ Q_j = H_j(n-1) * H_j(n-2) * \dots * H_j(1) \text{ if uplo indicates upper.}$$

Each Householder matrix  $H_j(i)$  is given by

$$H_j(i) = I - \text{tau}_j[i] * v_j(i) * v_j(i)'$$

where  $\text{tau}_j[i]$  is the corresponding Householder scalar. When uplo indicates lower, the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ . If uplo indicates upper, the last  $n-i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

#### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the hermitian matrix  $A_j$  is stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrices  $A_j$ .

- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda \times n$ .

On entry, the matrices  $A_j$  to be factored. On exit, if upper, then the elements on the diagonal and superdiagonal of  $A_j$  contain the tridiagonal form  $T_j$ ; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns. If lower, then the elements on the diagonal and subdiagonal contain the tridiagonal form  $T_j$ ; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns.

- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_j$ .
- **D** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideD).  
The diagonal elements of  $T_j$ .
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $strideD \geq n$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideE).  
The off-diagonal elements of  $T_j$ .
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $strideE \geq n-1$ .
- **tau** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\tau_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\tau_j$  to the next one  $\tau_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $strideP \geq n-1$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>hetrd\_strided\_batched()

```
rocblas_status roc solver_zhetrd_strided_batched(rocblas_handle handle, const rocblas_fill uplo, const
rocblas_int n, rocblas_double_complex *A, const
rocblas_int lda, const rocblas_stride strideA, double *D,
const rocblas_stride strideD, double *E, const
rocblas_stride strideE, rocblas_double_complex *tau, const
rocblas_stride strideP, const rocblas_int batch_count)
```

```
rocblas_status roc solver_chetrd_strided_batched(rocblas_handle handle, const rocblas_fill uplo, const
rocblas_int n, rocblas_float_complex *A, const rocblas_int
lda, const rocblas_stride strideA, float *D, const
rocblas_stride strideD, float *E, const rocblas_stride
strideE, rocblas_float_complex *tau, const rocblas_stride
strideP, const rocblas_int batch_count)
```

HETRD\_STRIDED\_BATCHED computes the tridiagonal form of a batch of complex hermitian matrices  $A_j$ .

(This is the blocked version of the algorithm).

The tridiagonal form of  $A_j$  is given by:

$$T_j = Q_j' * A_j * Q_j, \text{ for } j = 1, 2, \dots, \text{batch\_count}$$

where  $T_j$  is hermitian tridiagonal and  $Q_j$  is a unitary matrix represented as the product of Householder matrices

$$Q_j = H_j(1) * H_j(2) * \dots * H_j(n-1) \text{ if } \text{uplo} \text{ indicates lower, or}$$

$$Q_j = H_j(n-1) * H_j(n-2) * \dots * H_j(1) \text{ if } \text{uplo} \text{ indicates upper.}$$

Each Householder matrix  $H_j(i)$  is given by

$$H_j(i) = I - \tau_j[i] * v_j(i) * v_j(i)'$$

where  $\tau_j[i]$  is the corresponding Householder scalar. When `uplo` indicates lower, the first  $i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i+1] = 1$ . If `uplo` indicates upper, the last  $n-i$  elements of the Householder vector  $v_j(i)$  are zero, and  $v_j(i)[i] = 1$ .

### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the hermitian matrix  $A_j$  is stored. If `uplo` indicates lower (or upper), then the upper (or lower) part of  $A$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrices  $A_j$ .
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of `strideA`).  
On entry, the matrices  $A_j$  to be factored. On exit, if `uplo`, then the elements on the diagonal and superdiagonal of  $A_j$  contain the tridiagonal form  $T_j$ ; the elements above the superdiagonal contain the  $i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns. If `lower`, then the elements on the diagonal and subdiagonal contain the tridiagonal form  $T_j$ ; the elements below the subdiagonal contain the  $n-i-1$  non-zero elements of vectors  $v_j(i)$  stored as columns.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
specifies the leading dimension of  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of `strideA`. Normal use case is `strideA`  $\geq lda * n$ .
- **D** – [out] pointer to real type. Array on the GPU (the size depends on the value of `strided`).  
The diagonal elements of  $T_j$ .
- **strided** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  and the next one  $D_{(j+1)}$ . There is no restriction for the value of `strided`. Normal use case is `strided`  $\geq n$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of `strideE`).  
The off-diagonal elements of  $T_j$ .

- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  and the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $\text{strideE} \geq n-1$ .
- **tau** – [out] pointer to type. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $\text{tau}_j$  of scalar factors of the Householder matrices  $H_j(i)$ .
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{tau}_j$  to the next one  $\text{tau}_{(j+1)}$ . There is no restriction for the value of strideP. Normal use is  $\text{strideP} \geq n-1$ .
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>sygs2()

rocblas\_status **roc solver\_dsygs2**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*B, const rocblas\_int ldb)

rocblas\_status **roc solver\_ssygs2**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*B, const rocblas\_int ldb)

SYGS2 reduces a real symmetric-definite generalized eigenproblem to standard form.

(This is the unblocked version of the algorithm).

The problem solved by this function is either of the form

```
A * X = lambda * B * X (1st form), or
A * B * X = lambda * X (2nd form), or
B * A * X = lambda * X (3rd form),
```

depending on the value of itype.

If the problem is of the 1st form, then A is overwritten as

```
inv(U') * A * inv(U), or
inv(L) * A * inv(L'),
```

where B has been factorized as either  $U' * U$  or  $L * L'$  as returned by POTRF, depending on the value of uplo.

If the problem is of the 2nd or 3rd form, then A is overwritten as

```
U * A * U', or
L' * A * L,
```

where B has been factorized as either  $U' * U$  or  $L * L'$  as returned by POTRF, depending on the value of uplo.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **itype** – [in] *rocblas\_iform* .  
Specifies the form of the generalized eigenproblem.



- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the matrix A is stored, and whether the factorization applied to B was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) parts of A and B are not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the matrix A. On exit, the transformed matrix associated with the equivalent standard eigenvalue problem.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of A.
- **B** – [out] pointer to type. Array on the GPU of dimension  $ldb \times n$ .  
The triangular factor of the matrix B, as returned by POTRF.
- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .  
Specifies the leading dimension of B.

### roc solver\_<type>sygs2\_batched()

rocblas\_status **roc solver\_dsygs2\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssygs2\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

SYGS2\_BATCHED reduces a batch of real symmetric-definite generalized eigenproblems to standard form.

(This is the unblocked version of the algorithm).

The problem solved by this function is either of the form

```
A_i * X_i = lambda_i * B_i * X_i (1st form), or
A_i * B_i * X_i = lambda_i * X_i (2nd form), or
B_i * A_i * X_i = lambda_i * X_i (3rd form),
```

depending on the value of itype.

If the problem is of the 1st form, then A\_i is overwritten as

```
inv(U_i') * A_i * inv(U_i), or
inv(L_i) * A_i * inv(L_i'),
```

where B\_i has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of uplo.

If the problem is of the 2nd or 3rd form, then A\_i is overwritten as

$$U_i * A_i * U_i', \text{ or}$$

$$L_i' * A_i * L_i,$$

where  $B_i$  has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of `uplo`.

#### Parameters

- **handle** – [in] `rocblas_handle`.
- **itype** – [in] `rocblas_iform`.  
Specifies the form of the generalized eigenproblems.
- **uplo** – [in] `rocblas_fill`.  
Specifies whether the upper or lower part of the matrices  $A_i$  are stored, and whether the factorization applied to  $B_i$  was upper or lower triangular. If `uplo` indicates lower (or upper), then the upper (or lower) parts of  $A_i$  and  $B_i$  are not used.
- **n** – [in] `rocblas_int`.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda * n$ .  
On entry, the matrices  $A_i$ . On exit, the transformed matrices associated with the equivalent standard eigenvalue problems.
- **lda** – [in] `rocblas_int`.  $lda \geq n$ .  
Specifies the leading dimension of  $A_i$ .
- **B** – [out] array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldb * n$ .  
The triangular factors of the matrices  $B_i$ , as returned by POTRF.
- **ldb** – [in] `rocblas_int`.  $ldb \geq n$ .  
Specifies the leading dimension of  $B_i$ .
- **batch\_count** – [in] `rocblas_int`.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

#### `roc solver_<type>sygs2_strided_batched()`

`rocblas_status roc solver_dsygs2_strided_batched`(`rocblas_handle` handle, const `rocblas_iform` itype, const `rocblas_fill` uplo, const `rocblas_int` n, double \*A, const `rocblas_int` lda, const `rocblas_stride` strideA, double \*B, const `rocblas_int` ldb, const `rocblas_stride` strideB, const `rocblas_int` batch\_count)

`rocblas_status roc solver_ssygs2_strided_batched`(`rocblas_handle` handle, const `rocblas_iform` itype, const `rocblas_fill` uplo, const `rocblas_int` n, float \*A, const `rocblas_int` lda, const `rocblas_stride` strideA, float \*B, const `rocblas_int` ldb, const `rocblas_stride` strideB, const `rocblas_int` batch\_count)

`SYGS2_STRIDED_BATCHED` reduces a batch of real symmetric-definite generalized eigenproblems to stan-

standard form.

(This is the unblocked version of the algorithm).

The problem solved by this function is either of the form

```
A_i * X_i = lambda_i * B_i * X_i (1st form), or
A_i * B_i * X_i = lambda_i * X_i (2nd form), or
B_i * A_i * X_i = lambda_i * X_i (3rd form),
```

depending on the value of `itype`.

If the problem is of the 1st form, then `A_i` is overwritten as

```
inv(U_i') * A_i * inv(U_i), or
inv(L_i) * A_i * inv(L_i'),
```

where `B_i` has been factorized as either `U_i' * U_i` or `L_i * L_i'` as returned by POTRF, depending on the value of `uplo`.

If the problem is of the 2nd or 3rd form, then `A_i` is overwritten as

```
U_i * A_i * U_i', or
L_i' * A_i * L_i,
```

where `B_i` has been factorized as either `U_i' * U_i` or `L_i * L_i'` as returned by POTRF, depending on the value of `uplo`.

### Parameters

- **handle** – [in] `rocblas_handle`.
- **itype** – [in] `rocblas_iform`.  
Specifies the form of the generalized eigenproblems.
- **uplo** – [in] `rocblas_fill`.  
Specifies whether the upper or lower part of the matrices `A_i` are stored, and whether the factorization applied to `B_i` was upper or lower triangular. If `uplo` indicates lower (or upper), then the upper (or lower) parts of `A_i` and `B_i` are not used.
- **n** – [in] `rocblas_int`.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of `strideA`).  
On entry, the matrices `A_i`. On exit, the transformed matrices associated with the equivalent standard eigenvalue problems.
- **lda** – [in] `rocblas_int`.  $lda \geq n$ .  
Specifies the leading dimension of `A_i`.
- **strideA** – [in] `rocblas_stride`.  
Stride from the start of one matrix `A_i` and the next one `A_{(i+1)}`. There is no restriction for the value of `strideA`. Normal use case is  $strideA \geq lda * n$ .
- **B** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideB`).  
The triangular factors of the matrices `B_i`, as returned by POTRF.

- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .  
Specifies the leading dimension of  $B_i$ .
- **strideB** – [in] rocblas\_stride.  
Stride from the start of one matrix  $B_i$  and the next one  $B_{(i+1)}$ . There is no restriction for the value of strideB. Normal use case is  $strideB \geq ldb * n$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>hegs2()

rocblas\_status **roc solver\_zhegs2**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*B, const rocblas\_int ldb)

rocblas\_status **roc solver\_chegs2**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*B, const rocblas\_int ldb)

HEGS2 reduces a hermitian-definite generalized eigenproblem to standard form.

(This is the unblocked version of the algorithm).

The problem solved by this function is either of the form

```
A * X = lambda * B * X (1st form), or
A * B * X = lambda * X (2nd form), or
B * A * X = lambda * X (3rd form),
```

depending on the value of itype.

If the problem is of the 1st form, then A is overwritten as

```
inv(U') * A * inv(U), or
inv(L) * A * inv(L'),
```

where B has been factorized as either  $U' * U$  or  $L * L'$  as returned by POTRF, depending on the value of uplo.

If the problem is of the 2nd or 3rd form, then A is overwritten as

```
U * A * U', or
L' * A * L,
```

where B has been factorized as either  $U' * U$  or  $L * L'$  as returned by POTRF, depending on the value of uplo.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **itype** – [in] *rocblas\_iform*.  
Specifies the form of the generalized eigenproblem.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the matrix A is stored, and whether the factorization applied to B was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) parts of A and B are not used.

- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $\text{lda} * n$ .  
On entry, the matrix A. On exit, the transformed matrix associated with the equivalent standard eigenvalue problem.
- **lda** – [in] rocblas\_int.  $\text{lda} \geq n$ .  
Specifies the leading dimension of A.
- **B** – [out] pointer to type. Array on the GPU of dimension  $\text{ldb} * n$ .  
The triangular factor of the matrix B, as returned by POTRF.
- **ldb** – [in] rocblas\_int.  $\text{ldb} \geq n$ .  
Specifies the leading dimension of B.

### roc solver\_<type>hegs2\_batched()

rocblas\_status **roc solver\_zhegs2\_batched**(rocblas\_handle handle, const rocblas\_iform itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_double\_complex \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_chegs2\_batched**(rocblas\_handle handle, const rocblas\_iform itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_float\_complex \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

HEGS2\_BATCHED reduces a batch of hermitian-definite generalized eigenproblems to standard form.

(This is the unblocked version of the algorithm).

The problem solved by this function is either of the form

$A_i * X_i = \text{lambda}_i * B_i * X_i \text{ (1st form), or}$ $A_i * B_i * X_i = \text{lambda}_i * X_i \text{ (2nd form), or}$ $B_i * A_i * X_i = \text{lambda}_i * X_i \text{ (3rd form),}$
---

depending on the value of itype.

If the problem is of the 1st form, then  $A_i$  is overwritten as

$\text{inv}(U_i') * A_i * \text{inv}(U_i), \text{ or}$ $\text{inv}(L_i) * A_i * \text{inv}(L_i'),$
--

where  $B_i$  has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of uplo.

If the problem is of the 2nd or 3rd form, then  $A_i$  is overwritten as

$U_i * A_i * U_i', \text{ or}$ $L_i' * A_i * L_i,$
--

where  $B_i$  has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of uplo.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **itype** – [in] *rocblas\_iform* .  
Specifies the form of the generalized eigenproblems.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the matrices  $A_i$  are stored, and whether the factorization applied to  $B_i$  was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) parts of  $A_i$  and  $B_i$  are not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda \times n$ .  
On entry, the matrices  $A_i$ . On exit, the transformed matrices associated with the equivalent standard eigenvalue problems.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of  $A_i$ .
- **B** – [out] array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldb \times n$ .  
The triangular factors of the matrices  $B_i$ , as returned by POTRF.
- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .  
Specifies the leading dimension of  $B_i$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

**roc solver\_<type>hegs2\_strided\_batched()**

```
rocblas_status roc solver_zhegs2_strided_batched(rocblas_handle handle, const rocblas_iform itype, const
rocblas_fill uplo, const rocblas_int n,
rocblas_double_complex *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_double_complex *B, const
rocblas_int ldb, const rocblas_stride strideB, const
rocblas_int batch_count)
```

```
rocblas_status roc solver_chegs2_strided_batched(rocblas_handle handle, const rocblas_iform itype, const
rocblas_fill uplo, const rocblas_int n,
rocblas_float_complex *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_float_complex *B, const
rocblas_int ldb, const rocblas_stride strideB, const
rocblas_int batch_count)
```

HEGS2\_STRIDED\_BATCHED reduces a batch of hermitian-definite generalized eigenproblems to standard form.

(This is the unblocked version of the algorithm).

The problem solved by this function is either of the form

```
A_i * X_i = lambda_i * B_i * X_i (1st form), or
A_i * B_i * X_i = lambda_i * X_i (2nd form), or
B_i * A_i * X_i = lambda_i * X_i (3rd form),
```

depending on the value of `itype`.

If the problem is of the 1st form, then `A_i` is overwritten as

```
inv(U_i') * A_i * inv(U_i), or
inv(L_i) * A_i * inv(L_i'),
```

where `B_i` has been factorized as either `U_i' * U_i` or `L_i * L_i'` as returned by POTRF, depending on the value of `uplo`.

If the problem is of the 2nd or 3rd form, then `A_i` is overwritten as

```
U_i * A_i * U_i', or
L_i' * A_i * L_i,
```

where `B_i` has been factorized as either `U_i' * U_i` or `L_i * L_i'` as returned by POTRF, depending on the value of `uplo`.

### Parameters

- **handle** – [in] `rocblas_handle`.
- **itype** – [in] `rocblas_iform`.  
Specifies the form of the generalized eigenproblems.
- **uplo** – [in] `rocblas_fill`.  
Specifies whether the upper or lower part of the matrices `A_i` are stored, and whether the factorization applied to `B_i` was upper or lower triangular. If `uplo` indicates lower (or upper), then the upper (or lower) parts of `A_i` and `B_i` are not used.
- **n** – [in] `rocblas_int`. `n >= 0`.  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of `strideA`).  
On entry, the matrices `A_i`. On exit, the transformed matrices associated with the equivalent standard eigenvalue problems.
- **lda** – [in] `rocblas_int`. `lda >= n`.  
Specifies the leading dimension of `A_i`.
- **strideA** – [in] `rocblas_stride`.  
Stride from the start of one matrix `A_i` and the next one `A_(i+1)`. There is no restriction for the value of `strideA`. Normal use case is `strideA >= lda*n`.
- **B** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideB`).  
The triangular factors of the matrices `B_i`, as returned by POTRF.
- **ldb** – [in] `rocblas_int`. `ldb >= n`.  
Specifies the leading dimension of `B_i`.

- **strideB** – [in] rocblas\_stride.

Stride from the start of one matrix  $B_i$  and the next one  $B_{(i+1)}$ . There is no restriction for the value of strideB. Normal use case is  $\text{strideB} \geq \text{ldb} * n$ .

- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .

Number of matrices in the batch.

### roc solver\_<type>sygst()

rocblas\_status **roc solver\_dsygst**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*B, const rocblas\_int ldb)

rocblas\_status **roc solver\_ssygst**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*B, const rocblas\_int ldb)

SYGST reduces a real symmetric-definite generalized eigenproblem to standard form.

(This is the blocked version of the algorithm).

The problem solved by this function is either of the form

```
A * X = lambda * B * X (1st form), or
A * B * X = lambda * X (2nd form), or
B * A * X = lambda * X (3rd form),
```

depending on the value of itype.

If the problem is of the 1st form, then A is overwritten as

```
inv(U') * A * inv(U), or
inv(L) * A * inv(L'),
```

where B has been factorized as either  $U' * U$  or  $L * L'$  as returned by POTRF, depending on the value of uplo.

If the problem is of the 2nd or 3rd form, then A is overwritten as

```
U * A * U', or
L' * A * L,
```

where B has been factorized as either  $U' * U$  or  $L * L'$  as returned by POTRF, depending on the value of uplo.

#### Parameters

- **handle** – [in] rocblas\_handle.

- **itype** – [in] *rocblas\_iform*.

Specifies the form of the generalized eigenproblem.

- **uplo** – [in] rocblas\_fill.

Specifies whether the upper or lower part of the matrix A is stored, and whether the factorization applied to B was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) parts of A and B are not used.

- **n** – [in] rocblas\_int.  $n \geq 0$ .

The matrix dimensions.



- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the matrix A. On exit, the transformed matrix associated with the equivalent standard eigenvalue problem.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of A.
- **B** – [out] pointer to type. Array on the GPU of dimension  $ldb \times n$ .  
The triangular factor of the matrix B, as returned by POTRF.
- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .  
Specifies the leading dimension of B.

### roc solver\_<type>sygst\_batched()

rocblas\_status **roc solver\_dsygst\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssygst\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

SYGST\_BATCHED reduces a batch of real symmetric-definite generalized eigenproblems to standard form.

(This is the blocked version of the algorithm).

The problem solved by this function is either of the form

$A_i * X_i = \lambda_i * B_i * X_i$ (1st form), <b>or</b> $A_i * B_i * X_i = \lambda_i * X_i$ (2nd form), <b>or</b> $B_i * A_i * X_i = \lambda_i * X_i$ (3rd form),
---

depending on the value of itype.

If the problem is of the 1st form, then  $A_i$  is overwritten as

$inv(U_i') * A_i * inv(U_i)$ , <b>or</b> $inv(L_i) * A_i * inv(L_i')$ ,
--

where  $B_i$  has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of uplo.

If the problem is of the 2nd or 3rd form, then  $A_i$  is overwritten as

$U_i * A_i * U_i'$ , <b>or</b> $L_i' * A_i * L_i$ ,
--

where  $B_i$  has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of uplo.

#### Parameters

- **handle** – [in] rocblas\_handle.

- **itype** – [in] *rocblas\_iform* .  
Specifies the form of the generalized eigenproblems.
- **uplo** – [in] *rocblas\_fill*.  
Specifies whether the upper or lower part of the matrices  $A_i$  are stored, and whether the factorization applied to  $B_i$  was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) parts of  $A_i$  and  $B_i$  are not used.
- **n** – [in] *rocblas\_int*.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda \times n$ .  
On entry, the matrices  $A_i$ . On exit, the transformed matrices associated with the equivalent standard eigenvalue problems.
- **lda** – [in] *rocblas\_int*.  $lda \geq n$ .  
Specifies the leading dimension of  $A_i$ .
- **B** – [out] array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldb \times n$ .  
The triangular factors of the matrices  $B_i$ , as returned by POTRF.
- **ldb** – [in] *rocblas\_int*.  $ldb \geq n$ .  
Specifies the leading dimension of  $B_i$ .
- **batch\_count** – [in] *rocblas\_int*.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>sygst\_strided\_batched()

rocblas\_status **roc solver\_dsygst\_strided\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*B, const rocblas\_int ldb, const rocblas\_stride strideB, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssygst\_strided\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*B, const rocblas\_int ldb, const rocblas\_stride strideB, const rocblas\_int batch\_count)

SYGST\_STRIDED\_BATCHED reduces a batch of real symmetric-definite generalized eigenproblems to standard form.

(This is the blocked version of the algorithm).

The problem solved by this function is either of the form

$A_i * X_i = \lambda_i * B_i * X_i$  (1st form), **or**  
 $A_i * B_i * X_i = \lambda_i * X_i$  (2nd form), **or**  
 $B_i * A_i * X_i = \lambda_i * X_i$  (3rd form),

depending on the value of `itype`.

If the problem is of the 1st form, then `A_i` is overwritten as

```
inv(U_i') * A_i * inv(U_i), or
inv(L_i) * A_i * inv(L_i'),
```

where `B_i` has been factorized as either `U_i' * U_i` or `L_i * L_i'` as returned by POTRF, depending on the value of `uplo`.

If the problem is of the 2nd or 3rd form, then `A_i` is overwritten as

```
U_i * A_i * U_i', or
L_i' * A_i * L_i,
```

where `B_i` has been factorized as either `U_i' * U_i` or `L_i * L_i'` as returned by POTRF, depending on the value of `uplo`.

### Parameters

- **handle** – [in] `rocblas_handle`.
- **itype** – [in] `rocblas_iform`.  
Specifies the form of the generalized eigenproblems.
- **uplo** – [in] `rocblas_fill`.  
Specifies whether the upper or lower part of the matrices `A_i` are stored, and whether the factorization applied to `B_i` was upper or lower triangular. If `uplo` indicates lower (or upper), then the upper (or lower) parts of `A_i` and `B_i` are not used.
- **n** – [in] `rocblas_int`. `n >= 0`.  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of `strideA`).  
On entry, the matrices `A_i`. On exit, the transformed matrices associated with the equivalent standard eigenvalue problems.
- **lda** – [in] `rocblas_int`. `lda >= n`.  
Specifies the leading dimension of `A_i`.
- **strideA** – [in] `rocblas_stride`.  
Stride from the start of one matrix `A_i` and the next one `A_(i+1)`. There is no restriction for the value of `strideA`. Normal use case is `strideA >= lda*n`.
- **B** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideB`).  
The triangular factors of the matrices `B_i`, as returned by POTRF.
- **ldb** – [in] `rocblas_int`. `ldb >= n`.  
Specifies the leading dimension of `B_i`.
- **strideB** – [in] `rocblas_stride`.  
Stride from the start of one matrix `B_i` and the next one `B_(i+1)`. There is no restriction for the value of `strideB`. Normal use case is `strideB >= ldb*n`.
- **batch\_count** – [in] `rocblas_int`. `batch_count >= 0`.  
Number of matrices in the batch.

**roc solver\_<type>hegst()**

rocblas\_status **roc solver\_zhegst**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*B, const rocblas\_int ldb)

rocblas\_status **roc solver\_chegst**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*B, const rocblas\_int ldb)

HEGST reduces a hermitian-definite generalized eigenproblem to standard form.

(This is the blocked version of the algorithm).

The problem solved by this function is either of the form

```
A * X = lambda * B * X (1st form), or
A * B * X = lambda * X (2nd form), or
B * A * X = lambda * X (3rd form),
```

depending on the value of itype.

If the problem is of the 1st form, then A is overwritten as

```
inv(U') * A * inv(U), or
inv(L) * A * inv(L'),
```

where B has been factorized as either  $U' * U$  or  $L * L'$  as returned by POTRF, depending on the value of uplo.

If the problem is of the 2nd or 3rd form, then A is overwritten as

```
U * A * U', or
L' * A * L,
```

where B has been factorized as either  $U' * U$  or  $L * L'$  as returned by POTRF, depending on the value of uplo.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **itype** – [in] *rocblas\_iform* .  
Specifies the form of the generalized eigenproblem.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the matrix A is stored, and whether the factorization applied to B was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) parts of A and B are not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda * n$ .  
On entry, the matrix A. On exit, the transformed matrix associated with the equivalent standard eigenvalue problem.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of A.

- **B** – [out] pointer to type. Array on the GPU of dimension  $ldb \times n$ .  
The triangular factor of the matrix B, as returned by POTRF.
- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .  
Specifies the leading dimension of B.

### roc solver\_<type>hegst\_batched()

rocblas\_status **roc solver\_zhegst\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_double\_complex \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_chegst\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_float\_complex \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

HEGST\_BATCHED reduces a batch of hermitian-definite generalized eigenproblems to standard form.

(This is the blocked version of the algorithm).

The problem solved by this function is either of the form

```
A_i * X_i = lambda_i * B_i * X_i (1st form), or
A_i * B_i * X_i = lambda_i * X_i (2nd form), or
B_i * A_i * X_i = lambda_i * X_i (3rd form),
```

depending on the value of itype.

If the problem is of the 1st form, then A\_i is overwritten as

```
inv(U_i') * A_i * inv(U_i), or
inv(L_i) * A_i * inv(L_i'),
```

where B\_i has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of uplo.

If the problem is of the 2nd or 3rd form, then A\_i is overwritten as

```
U_i * A_i * U_i', or
L_i' * A_i * L_i,
```

where B\_i has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of uplo.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **itype** – [in] *rocblas\_iform*.  
Specifies the form of the generalized eigenproblems.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the matrices A\_i are stored, and whether the factorization applied to B\_i was upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) parts of A\_i and B\_i are not used.

- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda \cdot n$ .  
On entry, the matrices  $A_i$ . On exit, the transformed matrices associated with the equivalent standard eigenvalue problems.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of  $A_i$ .
- **B** – [out] array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldb \cdot n$ .  
The triangular factors of the matrices  $B_i$ , as returned by POTRF.
- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .  
Specifies the leading dimension of  $B_i$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>hegst\_strided\_batched()

rocblas\_status **roc solver\_zhegst\_strided\_batched**(rocblas\_handle handle, const rocblas\_iform itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_double\_complex \*B, const rocblas\_int ldb, const rocblas\_stride strideB, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_chegst\_strided\_batched**(rocblas\_handle handle, const rocblas\_iform itype, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_float\_complex \*B, const rocblas\_int ldb, const rocblas\_stride strideB, const rocblas\_int batch\_count)

HEGST\_STRIDED\_BATCHED reduces a batch of hermitian-definite generalized eigenproblems to standard form.

(This is the blocked version of the algorithm).

The problem solved by this function is either of the form

$$\begin{aligned} A_i * X_i &= \lambda_i * B_i * X_i \text{ (1st form), } \text{or} \\ A_i * B_i * X_i &= \lambda_i * X_i \text{ (2nd form), } \text{or} \\ B_i * A_i * X_i &= \lambda_i * X_i \text{ (3rd form),} \end{aligned}$$

depending on the value of itype.

If the problem is of the 1st form, then  $A_i$  is overwritten as

$$\begin{aligned} \text{inv}(U_i') * A_i * \text{inv}(U_i), \text{ or} \\ \text{inv}(L_i) * A_i * \text{inv}(L_i'), \end{aligned}$$

where  $B_i$  has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of `uplo`.

If the problem is of the 2nd or 3rd form, then  $A_i$  is overwritten as

```
U_i * A_i * U_i', or
L_i' * A_i * L_i,
```

where  $B_i$  has been factorized as either  $U_i' * U_i$  or  $L_i * L_i'$  as returned by POTRF, depending on the value of `uplo`.

### Parameters

- **handle** – [in] `rocblas_handle`.
- **itype** – [in] `rocblas_iform`.  
Specifies the form of the generalized eigenproblems.
- **uplo** – [in] `rocblas_fill`.  
Specifies whether the upper or lower part of the matrices  $A_i$  are stored, and whether the factorization applied to  $B_i$  was upper or lower triangular. If `uplo` indicates lower (or upper), then the upper (or lower) parts of  $A_i$  and  $B_i$  are not used.
- **n** – [in] `rocblas_int`.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of `strideA`).  
On entry, the matrices  $A_i$ . On exit, the transformed matrices associated with the equivalent standard eigenvalue problems.
- **lda** – [in] `rocblas_int`.  $lda \geq n$ .  
Specifies the leading dimension of  $A_i$ .
- **strideA** – [in] `rocblas_stride`.  
Stride from the start of one matrix  $A_i$  and the next one  $A_{(i+1)}$ . There is no restriction for the value of `strideA`. Normal use case is `strideA`  $\geq lda * n$ .
- **B** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideB`).  
The triangular factors of the matrices  $B_i$ , as returned by POTRF.
- **ldb** – [in] `rocblas_int`.  $ldb \geq n$ .  
Specifies the leading dimension of  $B_i$ .
- **strideB** – [in] `rocblas_stride`.  
Stride from the start of one matrix  $B_i$  and the next one  $B_{(i+1)}$ . There is no restriction for the value of `strideB`. Normal use case is `strideB`  $\geq ldb * n$ .
- **batch\_count** – [in] `rocblas_int`. `batch_count`  $\geq 0$ .  
Number of matrices in the batch.

#### 2.7.4.4 Linear-systems solvers

##### roc solver\_<type>trtri()

rocblas\_status **roc solver\_ztrtri**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_ctrtri**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_dtrtri**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, double \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_strtri**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, float \*A, const rocblas\_int lda, rocblas\_int \*info)

TRTRI inverts a triangular n-by-n matrix A.

A can be upper or lower triangular, depending on the value of uplo, and unit or non-unit triangular, depending on the value of diag.

##### Parameters

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the matrix A is stored. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **diag** – [in] rocblas\_diagonal.  
If diag indicates unit, then the diagonal elements of A are also not referenced and assumed to be one.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the triangular matrix. On exit, the inverse of A if  $info = 0$ .
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of A.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If  $info = 0$ , successful exit. If  $info = i > 0$ , A is singular. A(i,i) is the first zero element in the diagonal.



**roc solver\_<type>trtri\_batched()**

rocblas\_status **roc solver\_ztrtri\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ctrtri\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dtrtri\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, double \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_strtri\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, float \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

TRTRI\_BATCHED inverts a batch of triangular n-by-n matrices  $A_j$ .

$A_j$  can be upper or lower triangular, depending on the value of uplo, and unit or non-unit triangular, depending on the value of diag.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the matrices  $A_j$  are stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A_j$  is not used.
- **diag** – [in] rocblas\_diagonal.  
If diag indicates unit, then the diagonal elements of matrices  $A_j$  are also not referenced and assumed to be one.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of all matrices  $A_j$  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 triangular matrices  $A_j$ . On exit, the inverses of  $A_j$  if  $info_j = 0$ .
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_j = 0$ , successful exit for inversion of  $A_j$ . If  $info_j = i > 0$ ,  $A_j$  is singular.  $A_j(i,i)$  is the first zero element in the diagonal.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

**roc solver\_<type>trtri\_strided\_batched()**

rocblas\_status **roc solver\_ztrtri\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, const rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ctrtri\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, const rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dtrtri\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_strtri\_strided\_batched**(rocblas\_handle handle, const rocblas\_fill uplo, const rocblas\_diagonal diag, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

TRTRI\_STRIDED\_BATCHED inverts a batch of triangular n-by-n matrices  $A_j$ .

$A_j$  can be upper or lower triangular, depending on the value of uplo, and unit or non-unit triangular, depending on the value of diag.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the matrices  $A_j$  are stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A_j$  is not used.
- **diag** – [in] rocblas\_diagonal.  
If diag indicates unit, then the diagonal elements of matrices  $A_j$  are also not referenced and assumed to be one.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns 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 triangular matrices  $A_j$ . On exit, the inverses of  $A_j$  if info\_j = 0.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $strideA \geq lda * n$

- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If info\_j = 0, successful exit for inversion of A\_j. If info\_j = i > 0, A\_j is singular. A\_j(i,i) is the first zero element in the diagonal.
- **batch\_count** – [in] rocblas\_int. batch\_count >= 0.  
Number of matrices in the batch.

### roc solver\_<type>getri()

rocblas\_status **roc solver\_zgetri**(rocblas\_handle handle, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

rocblas\_status **roc solver\_cgetri**(rocblas\_handle handle, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

rocblas\_status **roc solver\_dgetri**(rocblas\_handle handle, const rocblas\_int n, double \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

rocblas\_status **roc solver\_sgetri**(rocblas\_handle handle, const rocblas\_int n, float \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_int \*info)

GETRI inverts a general n-by-n matrix A using the LU factorization computed by GETRF.

The inverse is computed by solving the linear system

$$\text{inv}(A) * L = \text{inv}(U)$$

where L is the lower triangular factor of A with unit diagonal elements, and U is the upper triangular factor.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **n** – [in] rocblas\_int. n >= 0.  
The number of rows and columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension lda\*n.  
On entry, the factors L and U of the factorization  $A = P*L*U$  returned by GETRF. On exit, the inverse of A if info = 0; otherwise undefined.
- **lda** – [in] rocblas\_int. lda >= n.  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU of dimension n.  
The pivot indices returned by GETRF.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If info = 0, successful exit. If info = i > 0, U is singular. U(i,i) is the first zero pivot.

**roc solver\_<type>getri\_batched()**

rocblas\_status **roc solver\_zgetri\_batched**(rocblas\_handle handle, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgetri\_batched**(rocblas\_handle handle, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgetri\_batched**(rocblas\_handle handle, const rocblas\_int n, double \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgetri\_batched**(rocblas\_handle handle, const rocblas\_int n, float \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

GETRI\_BATCHED inverts a batch of general n-by-n matrices using the LU factorization computed by GETRF\_BATCHED.

The inverse is computed by solving the linear system

$$\text{inv}(A_j) * L_j = \text{inv}(U_j)$$

where  $L_j$  is the lower triangular factor of  $A_j$  with unit diagonal elements, and  $U_j$  is the upper triangular factor.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of all matrices  $A_j$  in the batch.
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda * n$ .  
On entry, the factors  $L_j$  and  $U_j$  of the factorization  $A = P_j * L_j * U_j$  returned by GETRF\_BATCHED. On exit, the inverses of  $A_j$  if  $info_j = 0$ ; otherwise undefined.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU (the size depends on the value of strideP).  
The pivot indices returned by GETRF\_BATCHED.
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_j$  to the next one  $ipiv_{(i+j)}$ . There is no restriction for the value of strideP. Normal use case is  $strideP \geq n$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_j = 0$ , successful exit for inversion of  $A_j$ . If  $info_j = i > 0$ ,  $U_j$  is singular.  $U_j(i,i)$  is the first zero pivot.

- **batch\_count** – [in] rocblas\_int. batch\_count >= 0.

Number of matrices in the batch.

### roc solver\_<type>getri\_strided\_batched()

rocblas\_status **roc solver\_zgetri\_strided\_batched**(rocblas\_handle handle, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgetri\_strided\_batched**(rocblas\_handle handle, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgetri\_strided\_batched**(rocblas\_handle handle, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgetri\_strided\_batched**(rocblas\_handle handle, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_int \*info, const rocblas\_int batch\_count)

GETRI\_STRIDED\_BATCHED inverts a batch of general n-by-n matrices using the LU factorization computed by GETRF\_STRIDED\_BATCHED.

The inverse is computed by solving the linear system

$$\text{inv}(A_j) * L_j = \text{inv}(U_j)$$

where  $L_j$  is the lower triangular factor of  $A_j$  with unit diagonal elements, and  $U_j$  is the upper triangular factor.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **n** – [in] rocblas\_int. n >= 0.  
The number of rows and columns 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 factors  $L_j$  and  $U_j$  of the factorization  $A_j = P_j * L_j * U_j$  returned by GETRF\_STRIDED\_BATCHED. On exit, the inverses of  $A_j$  if info\_j = 0; otherwise undefined.
- **lda** – [in] rocblas\_int. lda >= n.  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is strideA >= lda\*n

- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU (the size depends on the value of strideP).

The pivot indices returned by GETRF\_STRIDED\_BATCHED.

- **strideP** – [in] rocblas\_stride.

Stride from the start of one vector ipiv\_j to the next one ipiv\_(j+1). There is no restriction for the value of strideP. Normal use case is strideP >= n.

- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.

If info\_j = 0, successful exit for inversion of A\_j. If info\_j = i > 0, U\_j is singular. U\_j(i,i) is the first zero pivot.

- **batch\_count** – [in] rocblas\_int. batch\_count >= 0.

Number of matrices in the batch.

### roc solver\_<type>getrs()

rocblas\_status **roc solver\_zgetrs**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_int \*ipiv, rocblas\_double\_complex \*B, const rocblas\_int ldb)

rocblas\_status **roc solver\_cgetrs**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_int \*ipiv, rocblas\_float\_complex \*B, const rocblas\_int ldb)

rocblas\_status **roc solver\_dgetrs**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, double \*A, const rocblas\_int lda, const rocblas\_int \*ipiv, double \*B, const rocblas\_int ldb)

rocblas\_status **roc solver\_sgetrs**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, float \*A, const rocblas\_int lda, const rocblas\_int \*ipiv, float \*B, const rocblas\_int ldb)

GETRS solves a system of n linear equations on n variables using the LU factorization computed by GETRF.

It solves one of the following systems:

$\begin{aligned} A * X &= B \text{ (no transpose),} \\ A' * X &= B \text{ (transpose), or} \\ A^* * X &= B \text{ (conjugate transpose)} \end{aligned}$
---

depending on the value of trans.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **trans** – [in] rocblas\_operation.

Specifies the form of the system of equations.

- **n** – [in] rocblas\_int. n >= 0.

The order of the system, i.e. the number of columns and rows of A.

- **nrhs** – [in] rocblas\_int. nrhs >= 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 lda\*n.  
The factors L and U of the factorization  $A = P*L*U$  returned by GETRF.
- **lda** – [in] rocblas\_int. lda >= n.  
The leading dimension of A.
- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU of dimension n.  
The pivot indices returned by GETRF.
- **B** – [inout] pointer to type. Array on the GPU of dimension ldb\*nrhs.  
On entry, the right hand side matrix B. On exit, the solution matrix X.
- **ldb** – [in] rocblas\_int. ldb >= n.  
The leading dimension of B.

### roc solver\_<type>getrs\_batched()

rocblas\_status **roc solver\_zgetrs\_batched**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, rocblas\_double\_complex \*const A[], const rocblas\_int lda, const rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_double\_complex \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgetrs\_batched**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, rocblas\_float\_complex \*const A[], const rocblas\_int lda, const rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_float\_complex \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgetrs\_batched**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, double \*const A[], const rocblas\_int lda, const rocblas\_int \*ipiv, const rocblas\_stride strideP, double \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgetrs\_batched**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, float \*const A[], const rocblas\_int lda, const rocblas\_int \*ipiv, const rocblas\_stride strideP, float \*const B[], const rocblas\_int ldb, const rocblas\_int batch\_count)

GETRS\_BATCHED solves a batch of systems of n linear equations on n variables using the LU factorization computed by GETRF\_BATCHED.

For each instance j in the batch, it solves one of the following systems:

$$\begin{aligned} A_{-j} * X_{-j} &= B_{-j} \text{ (no transpose),} \\ A_{-j}' * X_{-j} &= B_{-j} \text{ (transpose), or} \\ A_{-j}^* * X_{-j} &= B_{-j} \text{ (conjugate transpose)} \end{aligned}$$

depending on the value of trans.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **trans** – [in] rocblas\_operation.  
Specifies the form of the system of equations of each instance in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The order of the system, i.e. the number of columns and rows of all  $A_j$  matrices.
- **nrhs** – [in] rocblas\_int.  $nrhs \geq 0$ .  
The number of right hand sides, i.e., the number of columns of all the matrices  $B_j$ .
- **A** – [in] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda*n$ .  
The factors  $L_j$  and  $U_j$  of the factorization  $A_j = P_j*L_j*U_j$  returned by GETRF\_BATCHED.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
The leading dimension of matrices  $A_j$ .
- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors  $ipiv_j$  of pivot indices returned by GETRF\_BATCHED.
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_j$  to the next one  $ipiv_{(j+1)}$ . There is no restriction for the value of strideP. Normal use case is  $strideP \geq n$ .
- **B** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldb*nrhs$ .  
On entry, the right hand side matrices  $B_j$ . On exit, the solution matrix  $X_j$  of each system in the batch.
- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .  
The leading dimension of matrices  $B_j$ .
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of instances (systems) in the batch.

**roc solver\_<type>getrs\_strided\_batched()**

```
rocblas_status roc solver_zgetrs_strided_batched(rocblas_handle handle, const rocblas_operation trans, const
rocblas_int n, const rocblas_int nrhs,
rocblas_double_complex *A, const rocblas_int lda, const
rocblas_stride strideA, const rocblas_int *ipiv, const
rocblas_stride strideP, rocblas_double_complex *B, const
rocblas_int ldb, const rocblas_stride strideB, const
rocblas_int batch_count)
```



rocblas\_status **roc solver\_cgetrs\_strided\_batched**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, const rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, const rocblas\_int \*ipiv, const rocblas\_stride strideP, const rocblas\_float\_complex \*B, const rocblas\_int ldb, const rocblas\_stride strideB, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgetrs\_strided\_batched**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, const rocblas\_int \*ipiv, const rocblas\_stride strideP, double \*B, const rocblas\_int ldb, const rocblas\_stride strideB, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgetrs\_strided\_batched**(rocblas\_handle handle, const rocblas\_operation trans, const rocblas\_int n, const rocblas\_int nrhs, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, const rocblas\_int \*ipiv, const rocblas\_stride strideP, float \*B, const rocblas\_int ldb, const rocblas\_stride strideB, const rocblas\_int batch\_count)

GETRS\_STRIDED\_BATCHED solves a batch of systems of  $n$  linear equations on  $n$  variables using the LU factorization computed by GETRF\_STRIDED\_BATCHED.

For each instance  $j$  in the batch, it solves one of the following systems:

$A_j * X_j = B_j$  (no transpose),  
 $A_j^T * X_j = B_j$  (transpose), or  
 $A_j^* * X_j = B_j$  (conjugate transpose)

depending on the value of trans.

### Parameters

- **handle** – [in] rocblas\_handle.
- **trans** – [in] rocblas\_operation.  
Specifies the form of the system of equations of each instance in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The order of the system, i.e. the number of columns and rows of all  $A_j$  matrices.
- **nrhs** – [in] rocblas\_int.  $\text{nrhs} \geq 0$ .  
The number of right hand sides, i.e., the number of columns of all the matrices  $B_j$ .
- **A** – [in] pointer to type. Array on the GPU (the size depends on the value of strideA).  
The factors  $L_j$  and  $U_j$  of the factorization  $A_j = P_j * L_j * U_j$  returned by GETRF\_STRIDED\_BATCHED.
- **lda** – [in] rocblas\_int.  $\text{lda} \geq n$ .  
The leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $\text{strideA} \geq \text{lda} * n$ .

- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU (the size depends on the value of strideP).  
Contains the vectors ipiv\_j of pivot indices returned by GETRF\_STRIDED\_BATCHED.
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector ipiv\_j to the next one ipiv\_(j+1). There is no restriction for the value of strideP. Normal use case is strideP >= 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\_j. On exit, the solution matrix X\_j of each system in the batch.
- **ldb** – [in] rocblas\_int. ldb >= n.  
The leading dimension of matrices B\_j.
- **strideB** – [in] rocblas\_stride.  
Stride from the start of one matrix B\_j and the next one B\_(j+1). There is no restriction for the value of strideB. Normal use case is strideB >= ldb\*nrhs.
- **batch\_count** – [in] rocblas\_int. batch\_count >= 0.  
Number of instances (systems) in the batch.

#### 2.7.4.5 Least-squares solvers

##### roc solver\_<type>gels()

rocblas\_status **roc solver\_zgels**(rocblas\_handle handle, rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int nrhs, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*B, const rocblas\_int ldb, rocblas\_int \*info)

rocblas\_status **roc solver\_cgels**(rocblas\_handle handle, rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int nrhs, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*B, const rocblas\_int ldb, rocblas\_int \*info)

rocblas\_status **roc solver\_dgels**(rocblas\_handle handle, rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int nrhs, double \*A, const rocblas\_int lda, double \*B, const rocblas\_int ldb, rocblas\_int \*info)

rocblas\_status **roc solver\_sgels**(rocblas\_handle handle, rocblas\_operation trans, const rocblas\_int m, const rocblas\_int n, const rocblas\_int nrhs, float \*A, const rocblas\_int lda, float \*B, const rocblas\_int ldb, rocblas\_int \*info)

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).

The problem solved by this function is either of the form

$A * X = B$  (no transpose), **or**  
 $A' * X = B$  (transpose/conjugate transpose)

depending on the value of `trans`.

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

```
|| B - A * X || (no transpose), or
|| B - A' * X || (transpose/conjugate transpose)
```

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

### Parameters

- **handle** – [in] rocblas\_handle.
- **trans** – [in] rocblas\_operation.  
Specifies the form of the system of equations.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of matrix A.
- **nrhs** – [in] rocblas\_int.  $nrhs \geq 0$ .  
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  $lda \cdot n$ .  
On entry, the  $m$ -by- $n$  matrix A. On exit, the QR (or LQ) factorization of A as returned by GEQRF (or GELQF).
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrix A.
- **B** – [inout] pointer to type. Array on the GPU of dimension  $ldb \cdot nrhs$ .  
On entry, the matrix B is  $m$ -by- $nrhs$  if non-transposed, or  $n$ -by- $nrhs$  if transposed. On exit, when `info = 0`, B is overwritten by the solution vectors (and the residuals in the overdetermined cases) stored as columns.
- **ldb** – [in] rocblas\_int.  $ldb \geq \max(m, n)$ .  
Specifies the leading dimension of matrix B.
- **info** – [out] pointer to rocblas\_int on the GPU.  
If `info = 0`, successful exit. If `info = j > 0`, the solution could not be computed because input matrix A is rank deficient; the  $j$ -th diagonal element of its triangular factor is zero.

**roc solver\_<type>gels\_batched()**

```
rocblas_status roc solver_zgels_batched(rocblas_handle handle, rocblas_operation trans, const rocblas_int m,
    const rocblas_int n, const rocblas_int nrhs, rocblas_double_complex
    *const A[], const rocblas_int lda, rocblas_double_complex *const B[],
    const rocblas_int ldb, rocblas_int *info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_cgels_batched(rocblas_handle handle, rocblas_operation trans, const rocblas_int m,
    const rocblas_int n, const rocblas_int nrhs, rocblas_float_complex
    *const A[], const rocblas_int lda, rocblas_float_complex *const B[],
    const rocblas_int ldb, rocblas_int *info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_dgels_batched(rocblas_handle handle, rocblas_operation trans, const rocblas_int m,
    const rocblas_int n, const rocblas_int nrhs, double *const A[], const
    rocblas_int lda, double *const B[], const rocblas_int ldb, rocblas_int
    *info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_sgels_batched(rocblas_handle handle, rocblas_operation trans, const rocblas_int m,
    const rocblas_int n, const rocblas_int nrhs, float *const A[], const
    rocblas_int lda, float *const B[], const rocblas_int ldb, rocblas_int
    *info, const rocblas_int batch_count)
```

GELS\_BATCHED solves batches of overdetermined (or underdetermined) linear systems defined by the array of  $m$ -by- $n$  matrices  $A$ , and an array of corresponding matrices  $B$ , using the QR factorizations computed by GEQRF (or the LQ factorizations computed by GELQF).

The problem solved by this function is either of the form

$$A_i * X_i = B_i \text{ (no transpose), or}$$

$$A_i' * X_i = B_i \text{ (transpose/conjugate transpose)}$$

depending on the value of `trans`.

If  $m \geq n$  (or  $n < m$  in the case of transpose/conjugate transpose), the systems are overdetermined and least-squares solutions approximating  $X_i$  are found minimizing

$$\| B_i - A_i * X_i \|^2 \text{ (no transpose), or}$$

$$\| B_i - A_i' * X_i \|^2 \text{ (transpose/conjugate transpose)}$$

If  $n < m$  (or  $m \geq n$  in the case of transpose/conjugate transpose), the system is underdetermined and a unique solution for  $X_i$  is chosen minimizing  $\| X_i \|^2$

**Parameters**

- **handle** – [in] rocblas\_handle.
- **trans** – [in] rocblas\_operation.  
Specifies the form of the system of equations.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all matrices  $A_i$  in the batch.

- **nrhs** – [in] rocblas\_int. nrhs  $\geq 0$ .  
The number of columns of all matrices  $B_i$  and  $X_i$  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  $lda \cdot n$ .  
On entry, the  $m$ -by- $n$  matrices  $A_i$ . On exit, the QR (or LQ) factorizations of  $A_i$  as returned by GEQRF (or GELQF).
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **B** – [inout] array of pointer to type. Each pointer points to an array on the GPU of dimension  $ldb \cdot nrhs$ .  
On entry, the matrices  $B_i$  are  $m$ -by- $nrhs$  if non-transposed, or  $n$ -by- $nrhs$  if transposed. On exit, when  $info = 0$ , each  $B_i$  is overwritten by the solution vectors (and the residuals in the overdetermined cases) stored as columns.
- **ldb** – [in] rocblas\_int.  $ldb \geq \max(m, n)$ .  
Specifies the leading dimension of matrices  $B_i$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_i = 0$ , successful exit for solution of  $A_i$ . If  $info_i = j > 0$ , the solution of  $A_i$  could not be computed because input matrix  $A_i$  is rank deficient; the  $j$ -th diagonal element of its triangular factor is zero.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>gels\_strided\_batched()

```
rocblas_status roc solver_zgels_strided_batched(rocblas_handle handle, rocblas_operation trans, const
rocblas_int m, const rocblas_int n, const rocblas_int nrhs,
rocblas_double_complex *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_double_complex *B, const
rocblas_int ldb, const rocblas_stride strideB, rocblas_int
*info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_cgels_strided_batched(rocblas_handle handle, rocblas_operation trans, const
rocblas_int m, const rocblas_int n, const rocblas_int nrhs,
rocblas_float_complex *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_float_complex *B, const
rocblas_int ldb, const rocblas_stride strideB, rocblas_int
*info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_dgels_strided_batched(rocblas_handle handle, rocblas_operation trans, const
                                                rocblas_int m, const rocblas_int n, const rocblas_int nrhs,
                                                double *A, const rocblas_int lda, const rocblas_stride
                                                strideA, double *B, const rocblas_int ldb, const
                                                rocblas_stride strideB, rocblas_int *info, const rocblas_int
                                                batch_count)
```

```
rocblas_status roc solver_sgels_strided_batched(rocblas_handle handle, rocblas_operation trans, const
                                                rocblas_int m, const rocblas_int n, const rocblas_int nrhs,
                                                float *A, const rocblas_int lda, const rocblas_stride strideA,
                                                float *B, const rocblas_int ldb, const rocblas_stride strideB,
                                                rocblas_int *info, const rocblas_int batch_count)
```

GELS\_STRIDED\_BATCHED solves batches of overdetermined (or underdetermined) linear systems defined by the array of m-by-n matrices A, and an array of corresponding matrices B, using the QR factorizations computed by GEQRF (or the LQ factorizations computed by GELQF).

The problem solved by this function is either of the form

```
A_i * X_i = B_i (no transpose), or
A_i' * X_i = B_i (transpose/conjugate transpose)
```

depending on the value of trans.

If  $m \geq n$  (or  $n < m$  in the case of transpose/conjugate transpose), the systems are overdetermined and least-squares solutions approximating  $X_i$  are found minimizing

```
|| B_i - A_i * X_i || (no transpose), or
|| B_i - A_i' * X_i || (transpose/conjugate transpose)
```

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

### Parameters

- **handle** – [in] rocblas\_handle.
- **trans** – [in] rocblas\_operation.  
Specifies the form of the system of equations.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of all matrices  $A_i$  in the batch.
- **nrhs** – [in] rocblas\_int.  $nrhs \geq 0$ .  
The number of columns of all matrices  $B_i$  and  $X_i$  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 m-by-n matrices  $A_i$ . On exit, the QR (or LQ) factorizations of  $A_i$  as returned by GEQRF (or GELQF).
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .

- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_i$  and the next one  $A_{(i+1)}$ . There is no restriction for the value of strideA. Normal use case is  $\text{strideA} \geq \text{lda} * n$
- **B** – [inout] pointer to type. Array on the GPU (the size depends on the value of strideB).  
On entry, the matrices  $B_i$  are m-by-nrhs if non-transposed, or n-by-nrhs if transposed. On exit, when  $\text{info} = 0$ , each  $B_i$  is overwritten by the solution vectors (and the residuals in the overdetermined cases) stored as columns.
- **ldb** – [in] rocblas\_int.  $\text{ldb} \geq \max(m, n)$ .  
Specifies the leading dimension of matrices  $B_i$ .
- **strideB** – [in] rocblas\_stride.  
Stride from the start of one matrix  $B_i$  and 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 rocblas\_int. Array of batch\_count integers on the GPU.  
If  $\text{info}_i = 0$ , successful exit for solution of  $A_i$ . If  $\text{info}_i = j > 0$ , the solution of  $A_i$  could not be computed because input matrix  $A_i$  is rank deficient; the j-th diagonal element of its triangular factor is zero.
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

#### 2.7.4.6 Symmetric Eigensolvers

##### roc solver\_<type>syev()

rocblas\_status **roc solver\_dsyev**(rocblas\_handle handle, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*D, double \*E, rocblas\_int \*info)

rocblas\_status **roc solver\_ssyev**(rocblas\_handle handle, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*D, float \*E, rocblas\_int \*info)

SYEV computes the eigenvalues and optionally the eigenvectors of a real symmetric matrix A.

The eigenvalues are returned in ascending order. The eigenvectors are computed depending on the value of evect. The computed eigenvectors are orthonormal.

##### Parameters

- **handle** – [in] rocblas\_handle.
- **evect** – [in] rocblas\_evect .  
Specifies whether the eigenvectors are to be computed. If evect is rocblas\_evect\_original, then the eigenvectors are computed. rocblas\_evect\_tridiagonal is not supported.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the symmetric matrix A is stored. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$   
Number of rows and columns of matrix A.

- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the matrix A. On exit, the eigenvectors of A if they were computed and the algorithm converged; otherwise contents of A are destroyed.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of matrix A.
- **D** – [out] pointer to type. Array on the GPU of dimension n.  
The eigenvalues of A in increasing order.
- **E** – [out] pointer to type. Array on the GPU of dimension n.  
This array is used to work internally with the tridiagonal matrix T associated to A. On exit, if  $info > 0$ , it contains the unconverged off-diagonal elements of T (or properly speaking, a tridiagonal matrix equivalent to T). The diagonal elements of this matrix are in D; those that converged correspond to a subset of the eigenvalues of A (not necessarily ordered).
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If  $info = 0$ , successful exit. If  $info = i > 0$ , the algorithm did not converge. i elements of E did not converge to zero.

### roc solver\_<type>syev\_batched()

rocblas\_status **roc solver\_dsyev\_batched**(rocblas\_handle handle, const *rocblas\_evect* evect, const rocblas\_fill uplo, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssyev\_batched**(rocblas\_handle handle, const *rocblas\_evect* evect, const rocblas\_fill uplo, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

SYEV\_BATCHED computes the eigenvalues and optionally the eigenvectors of a batch of real symmetric matrices  $A_j$ .

The eigenvalues are returned in ascending order. The eigenvectors are computed depending on the value of evect. The computed eigenvectors are orthonormal.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **evect** – [in] *rocblas\_evect* .  
Specifies whether the eigenvectors are to be computed. If evect is rocblas\_evect\_original, then the eigenvectors are computed. rocblas\_evect\_tridiagonal is not supported.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the symmetric matrices  $A_j$  is stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A_j$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$   
Number of rows and columns of matrices  $A_j$ .



- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda \cdot n$ .  
On entry, the matrices  $A_j$ . On exit, the eigenvectors of  $A_j$  if they were computed and the algorithm converged; otherwise contents of  $A_j$  are destroyed.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **D** – [out] pointer to type. Array on the GPU (the side depends on the value of strideD).  
The eigenvalues of  $A_j$  in increasing order.
- **strided** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  to the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $strideD \geq n$ .
- **E** – [out] pointer to type. Array on the GPU (the size depends on the value of strideE).  
This array is used to work internally with the tridiagonal matrix  $T_j$  associated to  $A_j$ . On exit, if  $info > 0$ , it contains the unconverged off-diagonal elements of  $T_j$  (or properly speaking, a tridiagonal matrix equivalent to  $T_j$ ). The diagonal elements of this matrix are in  $D_j$ ; those that converged correspond to a subset of the eigenvalues of  $A_j$  (not necessarily ordered).
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  to the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $strideE \geq n$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_j = 0$ , successful exit for matrix  $A_j$ . If  $info_j = i > 0$ , the algorithm did not converge.  $i$  elements of  $E_j$  did not converge to zero.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>syev\_strided\_batched()

rocblas\_status **roc solver\_dsyev\_strided\_batched**(rocblas\_handle handle, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssyev\_strided\_batched**(rocblas\_handle handle, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

SYEV\_STRIDED\_BATCHED computes the eigenvalues and optionally the eigenvectors of a batch of real symmetric matrices  $A_j$ .

The eigenvalues are returned in ascending order. The eigenvectors are computed depending on the value of evect. The computed eigenvectors are orthonormal.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **evect** – [in] rocblas\_evect .  
Specifies whether the eigenvectors are to be computed. If `evect` is `rocblas_evect_original`, then the eigenvectors are computed. `rocblas_evect_tridiagonal` is not supported.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the symmetric matrices  $A_j$  is stored. If `uplo` indicates lower (or upper), then the upper (or lower) part of  $A_j$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$   
Number of rows and columns of matrices  $A_j$ .
- **A** – [inout] pointer to type. Array on the GPU (the side depends on the value of `strideA`).  
On entry, the matrices  $A_j$ . On exit, the eigenvectors of  $A_j$  if they were computed and the algorithm converged; otherwise contents of  $A_j$  are destroyed.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
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 `strideA`  $\geq lda * n$ .
- **D** – [out] pointer to type. Array on the GPU (the side depends on the value of `strideD`).  
The eigenvalues of  $A_j$  in increasing order.
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  to the next one  $D_{(j+1)}$ . There is no restriction for the value of `strideD`. Normal use case is `strideD`  $\geq n$ .
- **E** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideE`).  
This array is used to work internally with the tridiagonal matrix  $T_j$  associated to  $A_j$ . On exit, if `info`  $> 0$ , it contains the unconverged off-diagonal elements of  $T_j$  (or properly speaking, a tridiagonal matrix equivalent to  $T_j$ ). The diagonal elements of this matrix are in  $D_j$ ; those that converged correspond to a subset of the eigenvalues of  $A_j$  (not necessarily ordered).
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  to the next one  $E_{(j+1)}$ . There is no restriction for the value of `strideE`. Normal use case is `strideE`  $\geq n$ .
- **info** – [out] pointer to rocblas\_int. Array of `batch_count` integers on the GPU.  
If `info_j` = 0, successful exit for matrix  $A_j$ . If `info_j` =  $i > 0$ , the algorithm did not converge.  $i$  elements of  $E_j$  did not converge to zero.
- **batch\_count** – [in] rocblas\_int. `batch_count`  $\geq 0$ .  
Number of matrices in the batch.

**roc solver\_<type>heev()**

rocblas\_status **roc solver\_zheev**(rocblas\_handle handle, const *rocblas\_evect* evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, double \*D, double \*E, rocblas\_int \*info)

rocblas\_status **roc solver\_cheev**(rocblas\_handle handle, const *rocblas\_evect* evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, float \*D, float \*E, rocblas\_int \*info)

HEEV computes the eigenvalues and optionally the eigenvectors of a Hermitian matrix A.

The eigenvalues are returned in ascending order. The eigenvectors are computed depending on the value of evect. The computed eigenvectors are orthonormal.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **evect** – [in] *rocblas\_evect* .  
Specifies whether the eigenvectors are to be computed. If evect is rocblas\_evect\_original, then the eigenvectors are computed. rocblas\_evect\_tridiagonal is not supported.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the Hermitian matrix A is stored. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- **n** – [in] rocblas\_int. n >= 0  
Number of rows and columns of matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension lda\*n.  
On entry, the matrix A. On exit, the eigenvectors of A if they were computed and the algorithm converged; otherwise contents of A are destroyed.
- **lda** – [in] rocblas\_int. lda >= n.  
Specifies the leading dimension of matrix A.
- **D** – [out] pointer to real type. Array on the GPU of dimension n.  
The eigenvalues of A in increasing order.
- **E** – [out] pointer to real type. Array on the GPU of dimension n.  
This array is used to work internally with the tridiagonal matrix T associated to A. On exit, if info > 0, it contains the unconverged off-diagonal elements of T (or properly speaking, a tridiagonal matrix equivalent to T). The diagonal elements of this matrix are in D; those that converged correspond to a subset of the eigenvalues of A (not necessarily ordered).
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If info = 0, successful exit. If info = i > 0, the algorithm did not converge. i elements of E did not converge to zero.

**roc solver\_<type>heev\_batched()**

rocblas\_status **roc solver\_zheev\_batched**(rocblas\_handle handle, const *rocblas\_evect* evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cheev\_batched**(rocblas\_handle handle, const *rocblas\_evect* evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

HEEV\_BATCHED computes the eigenvalues and optionally the eigenvectors of a batch of Hermitian matrices  $A_j$ .

The eigenvalues are returned in ascending order. The eigenvectors are computed depending on the value of evect. The computed eigenvectors are orthonormal.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **evect** – [in] *rocblas\_evect* .  
Specifies whether the eigenvectors are to be computed. If evect is rocblas\_evect\_original, then the eigenvectors are computed. rocblas\_evect\_tridiagonal is not supported.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the Hermitian matrices  $A_j$  is stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A_j$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$   
Number of rows and columns of matrices  $A_j$ .
- **A** – [inout] Array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda \cdot n$ .  
On entry, the matrices  $A_j$ . On exit, the eigenvectors of  $A_j$  if they were computed and the algorithm converged; otherwise contents of  $A_j$  are destroyed.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **D** – [out] pointer to real type. Array on the GPU (the side depends on the value of strideD).  
The eigenvalues of  $A_j$  in increasing order.
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  to the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $strideD \geq n$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideE).  
This array is used to work internally with the tridiagonal matrix  $T_j$  associated to  $A_j$ . On exit, if  $info > 0$ , it contains the unconverged off-diagonal elements of  $T_j$  (or properly speaking, a tridiagonal matrix equivalent to  $T_j$ ). The diagonal elements of this matrix are in  $D_j$ ; those that converged correspond to a subset of the eigenvalues of  $A_j$  (not necessarily ordered).

- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  to the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $\text{strideE} \geq n$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $\text{info}_j = 0$ , successful exit for matrix  $A_j$ . If  $\text{info}_j = i > 0$ , the algorithm did not converge.  $i$  elements of  $E_j$  did not converge to zero.
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>heev\_strided\_batched()

rocblas\_status **roc solver\_zheev\_strided\_batched**(rocblas\_handle handle, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cheev\_strided\_batched**(rocblas\_handle handle, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

HEEV\_STRIDED\_BATCHED computes the eigenvalues and optionally the eigenvectors of a batch of Hermitian matrices  $A_j$ .

The eigenvalues are returned in ascending order. The eigenvectors are computed depending on the value of evect. The computed eigenvectors are orthonormal.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **evect** – [in] rocblas\_evect .  
Specifies whether the eigenvectors are to be computed. If evect is rocblas\_evect\_original, then the eigenvectors are computed. rocblas\_evect\_tridiagonal is not supported.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower part of the Hermitian matrices  $A_j$  is stored. If uplo indicates lower (or upper), then the upper (or lower) part of  $A_j$  is not used.
- **n** – [in] rocblas\_int.  $n \geq 0$   
Number of rows and columns of matrices  $A_j$ .
- **A** – [inout] pointer to type. Array on the GPU (the side depends on the value of strideA).  
On entry, the matrices  $A_j$ . On exit, the eigenvectors of  $A_j$  if they were computed and the algorithm converged; otherwise contents of  $A_j$  are destroyed.
- **lda** – [in] rocblas\_int.  $\text{lda} \geq n$ .  
Specifies the leading dimension of matrices  $A_j$ .

- **strideA** – [in] rocblas\_stride.  
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  $\text{strideA} \geq \text{lda} * n$ .
- **D** – [out] pointer to real type. Array on the GPU (the side depends on the value of strideD).  
The eigenvalues of  $A_j$  in increasing order.
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_j$  to the next one  $D_{(j+1)}$ . There is no restriction for the value of strideD. Normal use case is  $\text{strideD} \geq n$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideE).  
This array is used to work internally with the tridiagonal matrix  $T_j$  associated to  $A_j$ . On exit, if  $\text{info} > 0$ , it contains the unconverged off-diagonal elements of  $T_j$  (or properly speaking, a tridiagonal matrix equivalent to  $T_j$ ). The diagonal elements of this matrix are in  $D_j$ ; those that converged correspond to a subset of the eigenvalues of  $A_j$  (not necessarily ordered).
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  to the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $\text{strideE} \geq n$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $\text{info}_j = 0$ , successful exit for matrix  $A_j$ . If  $\text{info}_j = i > 0$ , the algorithm did not converge.  $i$  elements of  $E_j$  did not converge to zero.
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>sygv()

rocblas\_status **roc solver\_dsylv**(rocblas\_handle handle, const rocblas\_iform itype, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*B, const rocblas\_int ldb, double \*D, double \*E, rocblas\_int \*info)

rocblas\_status **roc solver\_ssylv**(rocblas\_handle handle, const rocblas\_iform itype, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*B, const rocblas\_int ldb, float \*D, float \*E, rocblas\_int \*info)

SYGV computes the eigenvalues and (optionally) eigenvectors of a real generalized symmetric-definite eigenproblem.

The problem solved by this function is either of the form

$A * X = \text{lambda} * B * X \text{ (1st form), or}$ $A * B * X = \text{lambda} * X \text{ (2nd form), or}$ $B * A * X = \text{lambda} * X \text{ (3rd form),}$
---

depending on the value of itype.

#### Parameters

- **handle** – [in] rocblas\_handle.

- **itype** – [in] *rocblas\_iform* .  
Specifies the form of the generalized eigenproblem.
- **evect** – [in] *rocblas\_evect* .  
Specifies whether the eigenvectors are to be computed. If *evect* is *rocblas\_evect\_original*, then the eigenvectors are computed. *rocblas\_evect\_tridiagonal* is not supported.
- **uplo** – [in] *rocblas\_fill*.  
Specifies whether the upper or lower parts of the matrices A and B are stored. If *uplo* indicates lower (or upper), then the upper (or lower) parts of A and B are not used.
- **n** – [in] *rocblas\_int*.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the symmetric matrix A. On exit, if *evect* is original, the matrix Z of eigenvectors, normalized as follows:
  - a. If *itype* is *ax* or *abx*, as  $Z' * B * Z = I$ ;
  - b. If *itype* is *bax*, as  $Z' * \text{inv}(B) * Z = I$ . Otherwise, if *evect* is none, then the upper or lower triangular part of the matrix A (including the diagonal) is destroyed, depending on the value of *uplo*.
- **lda** – [in] *rocblas\_int*.  $lda \geq n$ .  
Specifies the leading dimension of A.
- **B** – [out] pointer to type. Array on the GPU of dimension  $ldb \times n$ .  
On entry, the symmetric positive definite matrix B. On exit, the triangular factor of B as returned by POTRF.
- **ldb** – [in] *rocblas\_int*.  $ldb \geq n$ .  
Specifies the leading dimension of B.
- **D** – [out] pointer to type. Array on the GPU of dimension  $n$ .  
On exit, the eigenvalues in increasing order.
- **E** – [out] pointer to type. Array on the GPU of dimension  $n$ .  
This array is used to work internally with the tridiagonal matrix T associated with the reduced eigenvalue problem. On exit, if  $0 < \text{info} \leq n$ , it contains the unconverged off-diagonal elements of T (or properly speaking, a tridiagonal matrix equivalent to T). The diagonal elements of this matrix are in D; those that converged correspond to a subset of the eigenvalues (not necessarily ordered).
- **info** – [out] pointer to a *rocblas\_int* on the GPU.  
If  $\text{info} = 0$ , successful exit. If  $\text{info} = i \leq n$ ,  $i$  off-diagonal elements of an intermediate tridiagonal form did not converge to zero. If  $\text{info} = n + i$ , the leading minor of order  $i$  of B is not positive definite.

**roc solver\_<type>sygv\_batched()**

rocblas\_status **roc solver\_dsylv\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const *rocblas\_evect* evect, const rocblas\_fill uplo, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*const B[], const rocblas\_int ldb, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_ssygv\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const *rocblas\_evect* evect, const rocblas\_fill uplo, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*const B[], const rocblas\_int ldb, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

SYGV\_BATCHED computes the eigenvalues and (optionally) eigenvectors of a batch of real generalized symmetric-definite eigenproblems.

The problem solved by this function is either of the form

$A_i * X_i = \lambda_i * B_i * X_i$  (1st form), **or**  
 $A_i * B_i * X_i = \lambda_i * X_i$  (2nd form), **or**  
 $B_i * A_i * X_i = \lambda_i * X_i$  (3rd form),

depending on the value of itype.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **itype** – [in] *rocblas\_iform* .  
Specifies the form of the generalized eigenproblems.
- **evect** – [in] *rocblas\_evect* .  
Specifies whether the eigenvectors are to be computed. If evect is rocblas\_evect\_original, then the eigenvectors are computed. rocblas\_evect\_tridiagonal is not supported.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower parts of the matrices  $A_i$  and  $B_i$  are stored. If uplo indicates lower (or upper), then the upper (or lower) parts of  $A_i$  and  $B_i$  are not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension  $lda * n$ .  
On entry, the symmetric matrices  $A_i$ . On exit, if evect is original, the matrix  $Z_i$  of eigenvectors, normalized as follows:
  - a. If itype is ax or abx, as  $Z_i' * B_i * Z_i = I$ ;
  - b. If itype is bax, as  $Z_i' * \text{inv}(B_i) * Z_i = I$ . Otherwise, if evect is none, then the upper or lower triangular part of the matrices  $A_i$  (including the diagonal) are destroyed, depending on the value of uplo.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of  $A_i$ .



- **B** – [out] array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldb \cdot n$ .

On entry, the symmetric positive definite matrices  $B_i$ . On exit, the triangular factor of  $B_i$  as returned by POTRF.

- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .

Specifies the leading dimension of  $B_i$ .

- **D** – [out] pointer to type. Array on the GPU (the size depends on the value of strideD).

On exit, the eigenvalues in increasing order.

- **strided** – [in] rocblas\_stride.

Stride from the start of one vector  $D_i$  to the next one  $D_{(i+1)}$ . There is no restriction for the value of strideD. Normal use is  $strided \geq n$ .

- **E** – [out] pointer to type. Array on the GPU (the size depends on the value of strideE).

This array is used to work internally with the tridiagonal matrix  $T_i$  associated with the  $i$ th reduced eigenvalue problem. On exit, if  $0 < info_i \leq n$ , it contains the unconverged off-diagonal elements of  $T_i$  (or properly speaking, a tridiagonal matrix equivalent to  $T_i$ ). The diagonal elements of this matrix are in  $D_i$ ; those that converged correspond to a subset of the eigenvalues (not necessarily ordered).

- **strideE** – [in] rocblas\_stride.

Stride from the start of one vector  $E_i$  to the next one  $E_{(i+1)}$ . There is no restriction for the value of strideE. Normal use is  $strideE \geq n$ .

- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.

If  $info_i = 0$ , successful exit of batch  $i$ . If  $info_i = j \leq n$ ,  $j$  off-diagonal elements of an intermediate tridiagonal form did not converge to zero. If  $info_i = n + j$ , the leading minor of order  $j$  of  $B_i$  is not positive definite.

- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .

Number of matrices in the batch.

## roc solver\_<type>sygv\_strided\_batched()

```
rocblas_status roc solver_dsygv_strided_batched(rocblas_handle handle, const rocblas_iform itype, const
rocblas_evect evect, const rocblas_fill uplo, const
rocblas_int n, double *A, const rocblas_int lda, const
rocblas_stride strideA, double *B, const rocblas_int ldb,
const rocblas_stride strideB, double *D, const rocblas_stride
strideD, double *E, const rocblas_stride strideE, rocblas_int
*info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_ssygv_strided_batched(rocblas_handle handle, const rocblas_iform itype, const
rocblas_evect evect, const rocblas_fill uplo, const
rocblas_int n, float *A, const rocblas_int lda, const
rocblas_stride strideA, float *B, const rocblas_int ldb, const
rocblas_stride strideB, float *D, const rocblas_stride
strideD, float *E, const rocblas_stride strideE, rocblas_int
*info, const rocblas_int batch_count)
```

SYGV\_STRIDED\_BATCHED computes the eigenvalues and (optionally) eigenvectors of a batch of real gener-

alized symmetric-definite eigenproblems.

The problem solved by this function is either of the form

$A_i * X_i = \lambda_i * B_i * X_i$  (1st form), **or**  
 $A_i * B_i * X_i = \lambda_i * X_i$  (2nd form), **or**  
 $B_i * A_i * X_i = \lambda_i * X_i$  (3rd form),

depending on the value of `itype`.

### Parameters

- **handle** – [in] `rocblas_handle`.
- **itype** – [in] `rocblas_iform`.  
Specifies the form of the generalized eigenproblems.
- **evect** – [in] `rocblas_evect`.  
Specifies whether the eigenvectors are to be computed. If `evect` is `rocblas_evect_original`, then the eigenvectors are computed. `rocblas_evect_tridiagonal` is not supported.
- **uplo** – [in] `rocblas_fill`.  
Specifies whether the upper or lower parts of the matrices  $A_i$  and  $B_i$  are stored. If `uplo` indicates lower (or upper), then the upper (or lower) parts of  $A_i$  and  $B_i$  are not used.
- **n** – [in] `rocblas_int`.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of `strideA`).  
On entry, the symmetric matrices  $A_i$ . On exit, if `evect` is `original`, the matrix  $Z_i$  of eigenvectors, normalized as follows:
  - a. If `itype` is `ax` or `abx`, as  $Z_i' * B_i * Z_i = I$ ;
  - b. If `itype` is `bax`, as  $Z_i' * \text{inv}(B_i) * Z_i = I$ . Otherwise, if `evect` is `none`, then the upper or lower triangular part of the matrices  $A_i$  (including the diagonal) are destroyed, depending on the value of `uplo`.
- **lda** – [in] `rocblas_int`.  $\text{lda} \geq n$ .  
Specifies the leading dimension of  $A_i$ .
- **strideA** – [in] `rocblas_stride`.  
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 is `strideA`  $\geq \text{lda} * n$ .
- **B** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideB`).  
On entry, the symmetric positive definite matrices  $B_i$ . On exit, the triangular factor of  $B_i$  as returned by POTRF.
- **ldb** – [in] `rocblas_int`.  $\text{ldb} \geq n$ .  
Specifies the leading dimension of  $B_i$ .
- **strideB** – [in] `rocblas_stride`.  
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 is `strideB`  $\geq \text{ldb} * n$ .

- **D** – [out] pointer to type. Array on the GPU (the size depends on the value of strideD).  
On exit, the eigenvalues in increasing order.
- **strided** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_i$  to the next one  $D_{(i+1)}$ . There is no restriction for the value of strideD. Normal use is strideD  $\geq n$ .
- **E** – [out] pointer to type. Array on the GPU (the size depends on the value of strideE).  
This array is used to work internally with the tridiagonal matrix  $T_i$  associated with the  $i$ th reduced eigenvalue problem. On exit, if  $0 < \text{info}_i \leq n$ , it contains the unconverged off-diagonal elements of  $T_i$  (or properly speaking, a tridiagonal matrix equivalent to  $T_i$ ). The diagonal elements of this matrix are in  $D_i$ ; those that converged correspond to a subset of the eigenvalues (not necessarily ordered).
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_i$  to the next one  $E_{(i+1)}$ . There is no restriction for the value of strideE. Normal use is strideE  $\geq n$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $\text{info}_i = 0$ , successful exit of batch  $i$ . If  $\text{info}_i = j \leq n$ ,  $j$  off-diagonal elements of an intermediate tridiagonal form did not converge to zero. If  $\text{info}_i = n + j$ , the leading minor of order  $j$  of  $B_i$  is not positive definite.
- **batch\_count** – [in] rocblas\_int. batch\_count  $\geq 0$ .  
Number of matrices in the batch.

## roc solver\_<type>hegv()

rocblas\_status **roc solver\_zhegv**(rocblas\_handle handle, const rocblas\_iform itype, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_double\_complex \*B, const rocblas\_int ldb, double \*D, double \*E, rocblas\_int \*info)

rocblas\_status **roc solver\_chev**(rocblas\_handle handle, const rocblas\_iform itype, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_float\_complex \*B, const rocblas\_int ldb, float \*D, float \*E, rocblas\_int \*info)

HEGV computes the eigenvalues and (optionally) eigenvectors of a complex generalized hermitian-definite eigenproblem.

The problem solved by this function is either of the form

$A * X = \text{lambda} * B * X \text{ (1st form), or}$ $A * B * X = \text{lambda} * X \text{ (2nd form), or}$ $B * A * X = \text{lambda} * X \text{ (3rd form),}$
---

depending on the value of itype.

### Parameters

- **handle** – [in] rocblas\_handle.
- **itype** – [in] rocblas\_iform .  
Specifies the form of the generalized eigenproblem.

- **evect** – [in] *rocblas\_evect* .  
Specifies whether the eigenvectors are to be computed. If *evect* is *rocblas\_evect\_original*, then the eigenvectors are computed. *rocblas\_evect\_tridiagonal* is not supported.
- **uplo** – [in] *rocblas\_fill*.  
Specifies whether the upper or lower parts of the matrices A and B are stored. If *uplo* indicates lower (or upper), then the upper (or lower) parts of A and B are not used.
- **n** – [in] *rocblas\_int*.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the hermitian matrix A. On exit, if *evect* is original, the matrix Z of eigenvectors, normalized as follows:
  - a. If *itype* is *ax* or *abx*, as  $Z' * B * Z = I$ ;
  - b. If *itype* is *bax*, as  $Z' * \text{inv}(B) * Z = I$ . Otherwise, if *evect* is *none*, then the upper or lower triangular part of the matrix A (including the diagonal) is destroyed, depending on the value of *uplo*.
- **lda** – [in] *rocblas\_int*.  $lda \geq n$ .  
Specifies the leading dimension of A.
- **B** – [out] pointer to type. Array on the GPU of dimension  $ldb \times n$ .  
On entry, the hermitian positive definite matrix B. On exit, the triangular factor of B as returned by POTRF.
- **ldb** – [in] *rocblas\_int*.  $ldb \geq n$ .  
Specifies the leading dimension of B.
- **D** – [out] pointer to real type. Array on the GPU of dimension  $n$ .  
On exit, the eigenvalues in increasing order.
- **E** – [out] pointer to real type. Array on the GPU of dimension  $n$ .  
This array is used to work internally with the tridiagonal matrix T associated with the reduced eigenvalue problem. On exit, if  $0 < \text{info} \leq n$ , it contains the unconverged off-diagonal elements of T (or properly speaking, a tridiagonal matrix equivalent to T). The diagonal elements of this matrix are in D; those that converged correspond to a subset of the eigenvalues (not necessarily ordered).
- **info** – [out] pointer to a *rocblas\_int* on the GPU.  
If *info* = 0, successful exit. If *info* =  $i \leq n$ ,  $i$  off-diagonal elements of an intermediate tridiagonal form did not converge to zero. If *info* =  $n + i$ , the leading minor of order  $i$  of B is not positive definite.

**roc solver\_<type>hegv\_batched()**

```
rocblas_status roc solver_zhegv_batched(rocblas_handle handle, const rocblas_iform itype, const rocblas_evect
    evect, const rocblas_fill uplo, const rocblas_int n,
    rocblas_double_complex *const A[], const rocblas_int lda,
    rocblas_double_complex *const B[], const rocblas_int ldb, double *D,
    const rocblas_stride strideD, double *E, const rocblas_stride strideE,
    rocblas_int *info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_chegv_batched(rocblas_handle handle, const rocblas_iform itype, const rocblas_evect
    evect, const rocblas_fill uplo, const rocblas_int n,
    rocblas_float_complex *const A[], const rocblas_int lda,
    rocblas_float_complex *const B[], const rocblas_int ldb, float *D,
    const rocblas_stride strideD, float *E, const rocblas_stride strideE,
    rocblas_int *info, const rocblas_int batch_count)
```

HEGV\_BATCHED computes the eigenvalues and (optionally) eigenvectors of a batch of complex generalized hermitian-definite eigenproblems.

The problem solved by this function is either of the form

```
A_i * X_i = lambda_i * B_i * X_i (1st form), or
A_i * B_i * X_i = lambda_i * X_i (2nd form), or
B_i * A_i * X_i = lambda_i * X_i (3rd form),
```

depending on the value of itype.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **itype** – [in] rocblas\_iform .  
Specifies the form of the generalized eigenproblems.
- **evect** – [in] rocblas\_evect .  
Specifies whether the eigenvectors are to be computed. If evect is rocblas\_evect\_original, then the eigenvectors are computed. rocblas\_evect\_tridiagonal is not supported.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower parts of the matrices A\_i and B\_i are stored. If uplo indicates lower (or upper), then the upper (or lower) parts of A\_i and B\_i are not used.
- **n** – [in] rocblas\_int. n >= 0.  
The matrix dimensions.
- **A** – [inout] array of pointers to type. Each pointer points to an array on the GPU of dimension lda\*n.  
On entry, the hermitian matrices A\_i. On exit, if evect is original, the matrix Z\_i of eigenvectors, normalized as follows:
  - a. If itype is ax or abx, as  $Z_i' * B_i * Z_i = I$ ;
  - b. If itype is bax, as  $Z_i' * \text{inv}(B_i) * Z_i = I$ . Otherwise, if evect is none, then the upper or lower triangular part of the matrices A\_i (including the diagonal) are destroyed, depending on the value of uplo.

- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of  $A_i$ .
- **B** – [out] array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldb \cdot n$ .  
On entry, the hermitian positive definite matrices  $B_i$ . On exit, the triangular factor of  $B_i$  as returned by POTRF.
- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .  
Specifies the leading dimension of  $B_i$ .
- **D** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideD).  
On exit, the eigenvalues in increasing order.
- **strided** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_i$  to the next one  $D_{(i+1)}$ . There is no restriction for the value of strideD. Normal use is  $strided \geq n$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideE).  
This array is used to work internally with the tridiagonal matrix  $T_i$  associated with the  $i$ th reduced eigenvalue problem. On exit, if  $0 < info_i \leq n$ , it contains the unconverged off-diagonal elements of  $T_i$  (or properly speaking, a tridiagonal matrix equivalent to  $T_i$ ). The diagonal elements of this matrix are in  $D_i$ ; those that converged correspond to a subset of the eigenvalues (not necessarily ordered).
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_i$  to the next one  $E_{(i+1)}$ . There is no restriction for the value of strideE. Normal use is  $strideE \geq n$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_i = 0$ , successful exit of batch  $i$ . If  $info_i = j \leq n$ ,  $j$  off-diagonal elements of an intermediate tridiagonal form did not converge to zero. If  $info_i = n + j$ , the leading minor of order  $j$  of  $B_i$  is not positive definite.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>hegv\_strided\_batched()

rocblas\_status roc solver\_zhegv\_strided\_batched(rocblas\_handle handle, const rocblas\_iform itype, const rocblas\_evect evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_double\_complex \*B, const rocblas\_int ldb, const rocblas\_stride strideB, double \*D, const rocblas\_stride strideD, double \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_chegv\_strided\_batched**(rocblas\_handle handle, const *rocblas\_iform* itype, const *rocblas\_evect* evect, const rocblas\_fill uplo, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_float\_complex \*B, const rocblas\_int ldb, const rocblas\_stride strideB, float \*D, const rocblas\_stride strideD, float \*E, const rocblas\_stride strideE, rocblas\_int \*info, const rocblas\_int batch\_count)

HEGV\_STRIDED\_BATCHED computes the eigenvalues and (optionally) eigenvectors of a batch of complex generalized hermitian-definite eigenproblems.

The problem solved by this function is either of the form

$A_i * X_i = \lambda_i * B_i * X_i$  (1st form), **or**  
 $A_i * B_i * X_i = \lambda_i * X_i$  (2nd form), **or**  
 $B_i * A_i * X_i = \lambda_i * X_i$  (3rd form),

depending on the value of itype.

### Parameters

- **handle** – [in] rocblas\_handle.
- **itype** – [in] *rocblas\_iform* .  
Specifies the form of the generalized eigenproblems.
- **evect** – [in] *rocblas\_evect* .  
Specifies whether the eigenvectors are to be computed. If evect is rocblas\_evect\_original, then the eigenvectors are computed. rocblas\_evect\_tridiagonal is not supported.
- **uplo** – [in] rocblas\_fill.  
Specifies whether the upper or lower parts of the matrices  $A_i$  and  $B_i$  are stored. If uplo indicates lower (or upper), then the upper (or lower) parts of  $A_i$  and  $B_i$  are not used.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The matrix dimensions.
- **A** – [inout] pointer to type. Array on the GPU (the size depends on the value of strideA).  
On entry, the hermitian matrices  $A_i$ . On exit, if evect is original, the matrix  $Z_i$  of eigenvectors, normalized as follows:
  - a. If itype is ax or abx, as  $Z_i' * B_i * Z_i = I$ ;
  - b. If itype is bax, as  $Z_i' * \text{inv}(B_i) * Z_i = I$ . Otherwise, if evect is none, then the upper or lower triangular part of the matrices  $A_i$  (including the diagonal) are destroyed, depending on the value of uplo.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of  $A_i$ .
- **strideA** – [in] rocblas\_stride.  
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 is  $\text{strideA} \geq lda * n$ .
- **B** – [out] pointer to type. Array on the GPU (the size depends on the value of strideB).  
On entry, the hermitian positive definite matrices  $B_i$ . On exit, the triangular factor of  $B_i$  as returned by POTRF.

- **ldb** – [in] rocblas\_int.  $ldb \geq n$ .  
Specifies the leading dimension of  $B_i$ .
- **strideB** – [in] rocblas\_stride.  
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 is  $strideB \geq ldb * n$ .
- **D** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideD).  
On exit, the eigenvalues in increasing order.
- **strideD** – [in] rocblas\_stride.  
Stride from the start of one vector  $D_i$  to the next one  $D_{(i+1)}$ . There is no restriction for the value of strideD. Normal use is  $strideD \geq n$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideE).  
This array is used to work internally with the tridiagonal matrix  $T_i$  associated with the  $i$ th reduced eigenvalue problem. On exit, if  $0 < info_i \leq n$ , it contains the unconverged off-diagonal elements of  $T_i$  (or properly speaking, a tridiagonal matrix equivalent to  $T_i$ ). The diagonal elements of this matrix are in  $D_i$ ; those that converged correspond to a subset of the eigenvalues (not necessarily ordered).
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_i$  to the next one  $E_{(i+1)}$ . There is no restriction for the value of strideE. Normal use is  $strideE \geq n$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_i = 0$ , successful exit of batch  $i$ . If  $info_i = j \leq n$ ,  $j$  off-diagonal elements of an intermediate tridiagonal form did not converge to zero. If  $info_i = n + j$ , the leading minor of order  $j$  of  $B_i$  is not positive definite.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### 2.7.4.7 Singular Value Decomposition

#### roc solver\_<type>gesvd()

rocblas\_status **roc solver\_zgesvd**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, double \*S, rocblas\_double\_complex \*U, const rocblas\_int ldu, rocblas\_double\_complex \*V, const rocblas\_int ldv, double \*E, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info)

rocblas\_status **roc solver\_cgesvd**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, float \*S, rocblas\_float\_complex \*U, const rocblas\_int ldu, rocblas\_float\_complex \*V, const rocblas\_int ldv, float \*E, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info)



rocblas\_status **rocblas\_dgesvd**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, double \*S, double \*U, const rocblas\_int ldu, double \*V, const rocblas\_int ldv, double \*E, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info)

rocblas\_status **rocblas\_sgesvd**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, float \*S, float \*U, const rocblas\_int ldu, float \*V, const rocblas\_int ldv, float \*E, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info)

GESVD computes the Singular Values and optionally the Singular Vectors of a general m-by-n matrix A (Singular Value Decomposition).

The SVD of matrix A is given by:

$$A = U * S * V'$$

where the m-by-n matrix S is zero except, possibly, for its min(m,n) diagonal elements, which are the singular values of A. U and V are orthogonal (unitary) matrices. The first min(m,n) columns of U and V are the left and right singular vectors of A, respectively.

The computation of the singular vectors is optional and it is controlled by the function arguments left\_svect and right\_svect as described below. When computed, this function returns the transpose (or transpose conjugate) of the right singular vectors, i.e. the rows of V'.

left\_svect and right\_svect are *rocblas\_svect* enums that can take the following values:

- *rocblas\_svect\_all*: the entire matrix U (or V') is computed,
- *rocblas\_svect\_singular*: only the singular vectors (first min(m,n) columns of U or rows of V') are computed,
- *rocblas\_svect\_overwrite*: the first columns (or rows) of A are overwritten with the singular vectors, or
- *rocblas\_svect\_none*: no columns (or rows) of U (or V') are computed, i.e. no singular vectors.

left\_svect and right\_svect cannot both be set to overwrite. When neither is set to overwrite, the contents of A are destroyed by the time the function returns.

---

**Note:** When  $m \gg n$  (or  $n \gg m$ ) the algorithm could be sped up by compressing the matrix A via a QR (or LQ) factorization, and working with the triangular factor afterwards (thin-SVD). If the singular vectors are also requested, its computation could be sped up as well via executing some intermediate operations out-of-place, and relying more on matrix multiplications (GEMMs); this will require, however, a larger memory workspace. The parameter fast\_alg controls whether the fast algorithm is executed or not. For more details see the sections “Tuning rocSOLVER performance” and “Memory model” on the User’s guide.

---

### Parameters

- **handle** – [in] rocblas\_handle.
- **left\_svect** – [in] *rocblas\_svect* .  
Specifies how the left singular vectors are computed.
- **right\_svect** – [in] *rocblas\_svect* .  
Specifies how the right singular vectors are computed.

- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry the matrix A. On exit, if left\_svect (or right\_svect) is equal to overwrite, the first columns (or rows) contain the left (or right) singular vectors; otherwise, contents of A are destroyed.
- **lda** – [in] rocblas\_int.  $lda \geq m$ .  
The leading dimension of A.
- **S** – [out] pointer to real type. Array on the GPU of dimension  $\min(m,n)$ .  
The singular values of A in decreasing order.
- **U** – [out] pointer to type. Array on the GPU of dimension  $ldu \times \min(m,n)$  if left\_svect is set to singular, or  $ldu \times m$  when left\_svect is equal to all.  
The matrix of left singular vectors stored as columns. Not referenced if left\_svect is set to overwrite or none.
- **ldu** – [in] rocblas\_int.  $ldu \geq m$  if left\_svect is all or singular;  $ldu \geq 1$  otherwise.  
The leading dimension of U.
- **V** – [out] pointer to type. Array on the GPU of dimension  $ldv \times n$ .  
The matrix of right singular vectors stored as rows (transposed / conjugate-transposed). Not referenced if right\_svect is set to overwrite or none.
- **ldv** – [in] rocblas\_int.  $ldv \geq n$  if right\_svect is all;  $ldv \geq \min(m,n)$  if right\_svect is set to singular; or  $ldv \geq 1$  otherwise.  
The leading dimension of V.
- **E** – [out] pointer to real type. Array on the GPU of dimension  $\min(m,n)-1$ .  
This array is used to work internally with the bidiagonal matrix B associated to A (using BDSQR). On exit, if  $info > 0$ , it contains the unconverged off-diagonal elements of B (or properly speaking, a bidiagonal matrix orthogonally equivalent to B). The diagonal elements of this matrix are in S; those that converged correspond to a subset of the singular values of A (not necessarily ordered).
- **fast\_alg** – [in] *rocblas\_workmode* .  
If set to rocblas\_outofplace, the function will execute the fast thin-SVD version of the algorithm when possible.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If  $info = 0$ , successful exit. If  $info = i > 0$ , BDSQR did not converge.  $i$  elements of E did not converge to zero.

**roc solver\_<type>gesvd\_batched()**

rocblas\_status **roc solver\_zgesvd\_batched**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, double \*S, const rocblas\_stride strideS, rocblas\_double\_complex \*U, const rocblas\_int ldu, const rocblas\_stride strideU, rocblas\_double\_complex \*V, const rocblas\_int ldv, const rocblas\_stride strideV, double \*E, const rocblas\_stride strideE, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgesvd\_batched**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, float \*S, const rocblas\_stride strideS, rocblas\_float\_complex \*U, const rocblas\_int ldu, const rocblas\_stride strideU, rocblas\_float\_complex \*V, const rocblas\_int ldv, const rocblas\_stride strideV, float \*E, const rocblas\_stride strideE, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgesvd\_batched**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, double \*const A[], const rocblas\_int lda, double \*S, const rocblas\_stride strideS, double \*U, const rocblas\_int ldu, const rocblas\_stride strideU, double \*V, const rocblas\_int ldv, const rocblas\_stride strideV, double \*E, const rocblas\_stride strideE, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgesvd\_batched**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, float \*S, const rocblas\_stride strideS, float \*U, const rocblas\_int ldu, const rocblas\_stride strideU, float \*V, const rocblas\_int ldv, const rocblas\_stride strideV, float \*E, const rocblas\_stride strideE, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info, const rocblas\_int batch\_count)

GESVD\_BATCHED computes the Singular Values and optionally the Singular Vectors of a batch of general m-by-n matrix A (Singular Value Decomposition).

The SVD of matrix A<sub>j</sub> is given by:

$$A_j = U_j * S_j * V_j'$$

where the m-by-n matrix S<sub>j</sub> is zero except, possibly, for its min(m,n) diagonal elements, which are the singular values of A<sub>j</sub>. U<sub>j</sub> and V<sub>j</sub> are orthogonal (unitary) matrices. The first min(m,n) columns of U<sub>j</sub> and V<sub>j</sub> are the left and right singular vectors of A<sub>j</sub>, respectively.

The computation of the singular vectors is optional and it is controlled by the function arguments left\_svect and right\_svect as described below. When computed, this function returns the transpose (or transpose conjugate) of the right singular vectors, i.e. the rows of V<sub>j</sub>'.

left\_svect and right\_svect are *rocblas\_svect* enums that can take the following values:

- `rocblas_svect_all`: the entire matrix  $U_j$  (or  $V_j'$ ) is computed,
- `rocblas_svect_singular`: only the singular vectors (first  $\min(m,n)$  columns of  $U_j$  or rows of  $V_j'$ ) are computed,
- `rocblas_svect_overwrite`: the first columns (or rows) of  $A_j$  are overwritten with the singular vectors, or
- `rocblas_svect_none`: no columns (or rows) of  $U_j$  (or  $V_j'$ ) are computed, i.e. no singular vectors.

`left_svect` and `right_svect` cannot both be set to overwrite. When neither is set to overwrite, the contents of  $A_j$  are destroyed by the time the function returns.

---

**Note:** When  $m \gg n$  (or  $n \gg m$ ) the algorithm could be sped up by compressing the matrix  $A_j$  via a QR (or LQ) factorization, and working with the triangular factor afterwards (thin-SVD). If the singular vectors are also requested, its computation could be sped up as well via executing some intermediate operations out-of-place, and relying more on matrix multiplications (GEMMs); this will require, however, a larger memory workspace. The parameter `fast_alg` controls whether the fast algorithm is executed or not. For more details see the sections “Tuning rocSOLVER performance” and “Memory model” on the User’s guide.

---

### Parameters

- **handle** – [in] `rocblas_handle`.
- **left\_svect** – [in] `rocblas_svect` .  
Specifies how the left singular vectors are computed.
- **right\_svect** – [in] `rocblas_svect` .  
Specifies how the right singular vectors are computed.
- **m** – [in] `rocblas_int`.  $m \geq 0$ .  
The number of rows of all matrices  $A_j$  in the batch.
- **n** – [in] `rocblas_int`.  $n \geq 0$ .  
The number of columns of all matrices  $A_j$  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 matrices  $A_j$ . On exit, if `left_svect` (or `right_svect`) is equal to `overwrite`, the first columns (or rows) of  $A_j$  contain the left (or right) corresponding singular vectors; otherwise, contents of  $A_j$  are destroyed.
- **lda** – [in] `rocblas_int`.  $lda \geq m$ .  
The leading dimension of  $A_j$ .
- **S** – [out] pointer to real type. Array on the GPU (the size depends on the value of `strideS`).  
The singular values of  $A_j$  in decreasing order.
- **strideS** – [in] `rocblas_stride`.  
Stride from the start of one vector  $S_j$  to the next one  $S_{(j+1)}$ . There is no restriction for the value of `strideS`. Normal use case is  $strideS \geq \min(m,n)$ .
- **U** – [out] pointer to type. Array on the GPU (the side depends on the value of `strideU`).

The matrices  $U_j$  of left singular vectors stored as columns. Not referenced if `left_svect` is set to `overwrite` or `none`.

- **ldu** – [in] `rocblas_int`. `ldu`  $\geq m$  if `left_svect` is `all` or `singular`; `ldu`  $\geq 1$  otherwise.

The leading dimension of  $U_j$ .

- **strideU** – [in] `rocblas_stride`.

Stride from the start of one matrix  $U_j$  to the next one  $U_{(j+1)}$ . There is no restriction for the value of `strideU`. Normal use case is `strideU`  $\geq ldu * \min(m, n)$  if `left_svect` is set to `singular`, or `strideU`  $\geq ldu * m$  when `left_svect` is equal to `all`.

- **V** – [out] pointer to type. Array on the GPU (the size depends on the value of `strideV`).

The matrices  $V_j$  of right singular vectors stored as rows (transposed / conjugate-transposed). Not referenced if `right_svect` is set to `overwrite` or `none`.

- **ldv** – [in] `rocblas_int`. `ldv`  $\geq n$  if `right_svect` is `all`; `ldv`  $\geq \min(m, n)$  if `right_svect` is set to `singular`; or `ldv`  $\geq 1$  otherwise.

The leading dimension of  $V$ .

- **strideV** – [in] `rocblas_stride`.

Stride from the start of one matrix  $V_j$  to the next one  $V_{(j+1)}$ . There is no restriction for the value of `strideV`. Normal use case is `strideV`  $\geq ldv * n$ .

- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of `strideE`).

This array is used to work internally with the bidiagonal matrix  $B_j$  associated to  $A_j$  (using BDSQR). On exit, if `info`  $> 0$ , it contains the unconverged off-diagonal elements of  $B_j$  (or properly speaking, a bidiagonal matrix orthogonally equivalent to  $B_j$ ). The diagonal elements of this matrix are in  $S_j$ ; those that converged correspond to a subset of the singular values of  $A_j$  (not necessarily ordered).

- **strideE** – [in] `rocblas_stride`.

Stride from the start of one vector  $E_j$  to the next one  $E_{(j+1)}$ . There is no restriction for the value of `strideE`. Normal use case is `strideE`  $\geq \min(m, n) - 1$ .

- **fast\_alg** – [in] `rocblas_workmode`.

If set to `rocblas_outofplace`, the function will execute the fast thin-SVD version of the algorithm when possible.

- **info** – [out] pointer to a `rocblas_int` on the GPU.

If `info` = 0, successful exit. If `info` =  $i > 0$ , BDSQR did not converge.  $i$  elements of  $E$  did not converge to zero.

- **batch\_count** – [in] `rocblas_int`. `batch_count`  $\geq 0$ .

Number of matrices in the batch.

**roc solver\_<type>gesvd\_strided\_batched()**

rocblas\_status **roc solver\_zgesvd\_strided\_batched**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*S, const rocblas\_stride strideS, rocblas\_double\_complex \*U, const rocblas\_int ldu, const rocblas\_stride strideU, rocblas\_double\_complex \*V, const rocblas\_int ldv, const rocblas\_stride strideV, double \*E, const rocblas\_stride strideE, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgesvd\_strided\_batched**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*S, const rocblas\_stride strideS, rocblas\_float\_complex \*U, const rocblas\_int ldu, const rocblas\_stride strideU, rocblas\_float\_complex \*V, const rocblas\_int ldv, const rocblas\_stride strideV, float \*E, const rocblas\_stride strideE, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgesvd\_strided\_batched**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, double \*S, const rocblas\_stride strideS, double \*U, const rocblas\_int ldu, const rocblas\_stride strideU, double \*V, const rocblas\_int ldv, const rocblas\_stride strideV, double \*E, const rocblas\_stride strideE, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgesvd\_strided\_batched**(rocblas\_handle handle, const *rocblas\_svect* left\_svect, const *rocblas\_svect* right\_svect, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, float \*S, const rocblas\_stride strideS, float \*U, const rocblas\_int ldu, const rocblas\_stride strideU, float \*V, const rocblas\_int ldv, const rocblas\_stride strideV, float \*E, const rocblas\_stride strideE, const *rocblas\_workmode* fast\_alg, rocblas\_int \*info, const rocblas\_int batch\_count)

GESVD\_STRIDED\_BATCHED computes the Singular Values and optionally the Singular Vectors of a batch of general m-by-n matrix A (Singular Value Decomposition).

The SVD of matrix  $A_j$  is given by:

$$A_j = U_j * S_j * V_j'$$

where the m-by-n matrix  $S_j$  is zero except, possibly, for its  $\min(m,n)$  diagonal elements, which are the singular values of  $A_j$ .  $U_j$  and  $V_j$  are orthogonal (unitary) matrices. The first  $\min(m,n)$  columns of  $U_j$  and  $V_j$  are the left and right singular vectors of  $A_j$ , respectively.

The computation of the singular vectors is optional and it is controlled by the function arguments `left_svect` and `right_svect` as described below. When computed, this function returns the transpose (or transpose conjugate) of the right singular vectors, i.e. the rows of  $V_{-j}'$ .

`left_svect` and `right_svect` are *rocblas\_svect* enums that can take the following values:

- `rocblas_svect_all`: the entire matrix  $U_{-j}$  (or  $V_{-j}'$ ) is computed,
- `rocblas_svect_singular`: only the singular vectors (first  $\min(m,n)$  columns of  $U_{-j}$  or rows of  $V_{-j}'$ ) are computed,
- `rocblas_svect_overwrite`: the first columns (or rows) of  $A_{-j}$  are overwritten with the singular vectors, or
- `rocblas_svect_none`: no columns (or rows) of  $U_{-j}$  (or  $V_{-j}'$ ) are computed, i.e. no singular vectors.

`left_svect` and `right_svect` cannot both be set to overwrite. When neither is set to overwrite, the contents of  $A_{-j}$  are destroyed by the time the function returns.

---

**Note:** When  $m \gg n$  (or  $n \gg m$ ) the algorithm could be sped up by compressing the matrix  $A_{-j}$  via a QR (or LQ) factorization, and working with the triangular factor afterwards (thin-SVD). If the singular vectors are also requested, its computation could be sped up as well via executing some intermediate operations out-of-place, and relying more on matrix multiplications (GEMMs); this will require, however, a larger memory workspace. The parameter `fast_alg` controls whether the fast algorithm is executed or not. For more details see the sections “Tuning rocSOLVER performance” and “Memory model” on the User’s guide.

---

### Parameters

- **handle** – [in] `rocblas_handle`.
- **left\_svect** – [in] *rocblas\_svect* .  
Specifies how the left singular vectors are computed.
- **right\_svect** – [in] *rocblas\_svect* .  
Specifies how the right singular vectors are computed.
- **m** – [in] `rocblas_int`.  $m \geq 0$ .  
The number of rows of all matrices  $A_{-j}$  in the batch.
- **n** – [in] `rocblas_int`.  $n \geq 0$ .  
The number of columns of all matrices  $A_{-j}$  in the batch.
- **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, if `left_svect` (or `right_svect`) is equal to `overwrite`, the first columns (or rows) of  $A_{-j}$  contain the left (or right) corresponding singular vectors; otherwise, contents of  $A_{-j}$  are destroyed.
- **lda** – [in] `rocblas_int`.  $lda \geq m$ .  
The leading dimension of  $A_{-j}$ .
- **strideA** – [in] `rocblas_stride`.  
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  $strideA \geq lda * n$ .

- **S** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideS).  
The singular values of  $A_j$  in decreasing order.
- **strideS** – [in] rocblas\_stride.  
Stride from the start of one vector  $S_j$  to the next one  $S_{(j+1)}$ . There is no restriction for the value of strideS. Normal use case is  $\text{strideS} \geq \min(m,n)$ .
- **U** – [out] pointer to type. Array on the GPU (the side depends on the value of strideU).  
The matrices  $U_j$  of left singular vectors stored as columns. Not referenced if left\_svect is set to overwrite or none.
- **ldu** – [in] rocblas\_int.  $\text{ldu} \geq m$  if left\_svect is all or singular;  $\text{ldu} \geq 1$  otherwise.  
The leading dimension of  $U_j$ .
- **strideU** – [in] rocblas\_stride.  
Stride from the start of one matrix  $U_j$  to the next one  $U_{(j+1)}$ . There is no restriction for the value of strideU. Normal use case is  $\text{strideU} \geq \text{ldu} * \min(m,n)$  if left\_svect is set to singular, or  $\text{strideU} \geq \text{ldu} * m$  when left\_svect is equal to all.
- **V** – [out] pointer to type. Array on the GPU (the size depends on the value of strideV).  
The matrices  $V_j$  of right singular vectors stored as rows (transposed / conjugate-transposed). Not referenced if right\_svect is set to overwrite or none.
- **ldv** – [in] rocblas\_int.  $\text{ldv} \geq n$  if right\_svect is all;  $\text{ldv} \geq \min(m,n)$  if right\_svect is set to singular; or  $\text{ldv} \geq 1$  otherwise.  
The leading dimension of  $V$ .
- **strideV** – [in] rocblas\_stride.  
Stride from the start of one matrix  $V_j$  to the next one  $V_{(j+1)}$ . There is no restriction for the value of strideV. Normal use case is  $\text{strideV} \geq \text{ldv} * n$ .
- **E** – [out] pointer to real type. Array on the GPU (the size depends on the value of strideE).  
This array is used to work internally with the bidiagonal matrix  $B_j$  associated to  $A_j$  (using BDSQR). On exit, if  $\text{info} > 0$ , it contains the unconverged off-diagonal elements of  $B_j$  (or properly speaking, a bidiagonal matrix orthogonally equivalent to  $B_j$ ). The diagonal elements of this matrix are in  $S_j$ ; those that converged correspond to a subset of the singular values of  $A_j$  (not necessarily ordered).
- **strideE** – [in] rocblas\_stride.  
Stride from the start of one vector  $E_j$  to the next one  $E_{(j+1)}$ . There is no restriction for the value of strideE. Normal use case is  $\text{strideE} \geq \min(m,n)-1$ .
- **fast\_alg** – [in] *rocblas\_workmode* .  
If set to rocblas\_outofplace, the function will execute the fast thin-SVD version of the algorithm when possible.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If  $\text{info} = 0$ , successful exit. If  $\text{info} = i > 0$ , BDSQR did not converge.  $i$  elements of  $E$  did not converge to zero.
- **batch\_count** – [in] rocblas\_int.  $\text{batch\_count} \geq 0$ .  
Number of matrices in the batch.



## 2.7.5 Lapack-like Functions

Other Lapack-like routines provided by rocSOLVER.

### 2.7.5.1 Linear-systems solvers

#### roc solver\_<type>getri\_outofplace()

rocblas\_status **roc solver\_zgetri\_outofplace**(rocblas\_handle handle, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_double\_complex \*C, const rocblas\_int ldc, rocblas\_int \*info)

rocblas\_status **roc solver\_cgetri\_outofplace**(rocblas\_handle handle, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_int \*ipiv, rocblas\_float\_complex \*C, const rocblas\_int ldc, rocblas\_int \*info)

rocblas\_status **roc solver\_dgetri\_outofplace**(rocblas\_handle handle, const rocblas\_int n, double \*A, const rocblas\_int lda, rocblas\_int \*ipiv, double \*C, const rocblas\_int ldc, rocblas\_int \*info)

rocblas\_status **roc solver\_sgetri\_outofplace**(rocblas\_handle handle, const rocblas\_int n, float \*A, const rocblas\_int lda, rocblas\_int \*ipiv, float \*C, const rocblas\_int ldc, rocblas\_int \*info)

GETRI\_OUTOFPLACE computes the inverse  $C = \text{inv}(A)$  of a general n-by-n matrix A.

The inverse is computed by solving the linear system

$$A * C = I$$

where I is the identity matrix, and A is factorized as  $A = P * L * U$  as given by GETRF.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of the matrix A.
- **A** – [in] pointer to type. Array on the GPU of dimension  $\text{lda} * n$ .  
The factors L and U of the factorization  $A = P * L * U$  returned by GETRF.
- **lda** – [in] rocblas\_int.  $\text{lda} \geq n$ .  
Specifies the leading dimension of A.
- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU of dimension n.  
The pivot indices returned by GETRF.
- **C** – [out] pointer to type. Array on the GPU of dimension  $\text{ldc} * n$ .  
If  $\text{info} = 0$ , the inverse of A. Otherwise, undefined.

- **ldc** – [in] rocblas\_int. ldc >= n.  
Specifies the leading dimension of C.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If info = 0, successful exit. If info = i > 0, U is singular. U(i,i) is the first zero pivot.

### roc solver\_<type>getri\_outofplace\_batched()

rocblas\_status **roc solver\_zgetri\_outofplace\_batched**(rocblas\_handle handle, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_double\_complex \*const C[], const rocblas\_int ldc, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgetri\_outofplace\_batched**(rocblas\_handle handle, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, rocblas\_float\_complex \*const C[], const rocblas\_int ldc, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgetri\_outofplace\_batched**(rocblas\_handle handle, const rocblas\_int n, double \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, double \*const C[], const rocblas\_int ldc, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgetri\_outofplace\_batched**(rocblas\_handle handle, const rocblas\_int n, float \*const A[], const rocblas\_int lda, rocblas\_int \*ipiv, const rocblas\_stride strideP, float \*const C[], const rocblas\_int ldc, rocblas\_int \*info, const rocblas\_int batch\_count)

GETRI\_OUTOFPPLACE\_BATCHED computes the inverse  $C_j = \text{inv}(A_j)$  of a batch of general n-by-n matrices  $A_j$ .

The inverse is computed by solving the linear system

$$A_j * C_j = I$$

where I is the identity matrix, and  $A_j$  is factorized as  $A_j = P_j * L_j * U_j$  as given by GETRF\_BATCHED.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **n** – [in] rocblas\_int. n >= 0.  
The number of rows and columns of all matrices  $A_j$  in the batch.
- **A** – [in] array of pointers to type. Each pointer points to an array on the GPU of dimension lda\*n.  
The factors  $L_j$  and  $U_j$  of the factorization  $A_j = P_j * L_j * U_j$  returned by GETRF\_BATCHED.

- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU (the size depends on the value of strideP).  
The pivot indices returned by GETRF\_BATCHED.
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $ipiv_j$  to the next one  $ipiv_{(i+j)}$ . There is no restriction for the value of strideP. Normal use case is  $strideP \geq n$ .
- **C** – [out] array of pointers to type. Each pointer points to an array on the GPU of dimension  $ldc*n$ .  
If  $info = 0$ , the inverse of matrices  $A_j$ . Otherwise, undefined.
- **ldc** – [in] rocblas\_int.  $ldc \geq n$ .  
Specifies the leading dimension of  $C_j$ .
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If  $info_j = 0$ , successful exit for inversion of  $A_j$ . If  $info_j = i > 0$ ,  $U_j$  is singular.  $U_j(i,i)$  is the first zero pivot.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>getri\_outofplace\_strided\_batched()

```
rocblas_status roc solver_zgetri_outofplace_strided_batched(rocblas_handle handle, const rocblas_int n,
    rocblas_double_complex *A, const rocblas_int lda, const rocblas_stride strideA,
    rocblas_int *ipiv, const rocblas_stride strideP, rocblas_double_complex *C, const
    rocblas_int ldc, const rocblas_stride strideC, rocblas_int *info, const rocblas_int
    batch_count)
```

```
rocblas_status roc solver_cgetri_outofplace_strided_batched(rocblas_handle handle, const rocblas_int n,
    rocblas_float_complex *A, const rocblas_int lda, const rocblas_stride strideA, rocblas_int
    *ipiv, const rocblas_stride strideP,
    rocblas_float_complex *C, const rocblas_int ldc, const rocblas_stride strideC, rocblas_int
    *info, const rocblas_int batch_count)
```

```
rocblas_status rocsolver_dgetri_outofplace_strided_batched(rocblas_handle handle, const rocblas_int n,
double *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_int *ipiv,
const rocblas_stride strideP, double *C,
const rocblas_int ldc, const rocblas_stride
strideC, rocblas_int *info, const rocblas_int
batch_count)
```

```
rocblas_status rocsolver_sgetri_outofplace_strided_batched(rocblas_handle handle, const rocblas_int n,
float *A, const rocblas_int lda, const
rocblas_stride strideA, rocblas_int *ipiv,
const rocblas_stride strideP, float *C, const
rocblas_int ldc, const rocblas_stride strideC,
rocblas_int *info, const rocblas_int
batch_count)
```

GETRI\_OUTOFPLACE\_STRIDED\_BATCHED computes the inverse  $C_j = \text{inv}(A_j)$  of a batch of general  $n$ -by- $n$  matrices  $A_j$ .

The inverse is computed by solving the linear system

$$A_j * C_j = I$$

where  $I$  is the identity matrix, and  $A_j$  is factorized as  $A_j = P_j * L_j * U_j$  as given by GETRF\_STRIDED\_BATCHED.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of rows and columns of all matrices  $A_i$  in the batch.
- **A** – [in] pointer to type. Array on the GPU (the size depends on the value of strideA).  
The factors  $L_j$  and  $U_j$  of the factorization  $A_j = P_j * L_j * U_j$  returned by GETRF\_STRIDED\_BATCHED.
- **lda** – [in] rocblas\_int.  $lda \geq n$ .  
Specifies the leading dimension of matrices  $A_j$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_j$  and the next one  $A_{(j+1)}$ . There is no restriction for the value of strideA. Normal use case is  $\text{strideA} \geq lda * n$ .
- **ipiv** – [in] pointer to rocblas\_int. Array on the GPU (the size depends on the value of strideP).  
The pivot indices returned by GETRF\_STRIDED\_BATCHED.
- **strideP** – [in] rocblas\_stride.  
Stride from the start of one vector  $\text{ipiv}_j$  to the next one  $\text{ipiv}_{(j+1)}$ . There is no restriction for the value of strideP. Normal use case is  $\text{strideP} \geq n$ .
- **C** – [out] pointer to type. Array on the GPU (the size depends on the value of strideC).  
If  $\text{info} = 0$ , the inverse of matrices  $A_j$ . Otherwise, undefined.

- **ldc** – [in] rocblas\_int. ldc >= n.  
Specifies the leading dimension of C<sub>j</sub>.
- **strideC** – [in] rocblas\_stride.  
Stride from the start of one matrix C<sub>j</sub> and the next one C<sub>(j+1)</sub>. There is no restriction for the value of strideC. Normal use case is strideC >= ldc\*n
- **info** – [out] pointer to rocblas\_int. Array of batch\_count integers on the GPU.  
If info<sub>j</sub> = 0, successful exit for inversion of A<sub>j</sub>. If info<sub>j</sub> = i > 0, U<sub>j</sub> is singular. U<sub>j</sub>(i,i) is the first zero pivot.
- **batch\_count** – [in] rocblas\_int. batch\_count >= 0.  
Number of matrices in the batch.

### 2.7.5.2 Triangular Factorizations

#### roc solver\_<type>getf2\_npvt()

rocblas\_status **roc solver\_zgetf2\_npvt** (rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_cgetf2\_npvt** (rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_dgetf2\_npvt** (rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_sgetf2\_npvt** (rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, rocblas\_int \*info)

GETF2\_NPVT computes the LU factorization of a general m-by-n matrix A without partial pivoting.

(This is the unblocked Level-2-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with small and mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

The factorization has the form

$$A = L * U$$

where L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

Note: Although this routine can offer better performance, Gaussian elimination without pivoting is not backward stable. If numerical accuracy is compromised, use the legacy-LAPACK-like API GETF2 routines instead.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int. m >= 0.  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int. n >= 0.  
The number of columns of the matrix A.

- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the  $m$ -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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of  $A$ .
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If  $info = 0$ , successful exit. If  $info = i > 0$ ,  $U$  is singular.  $U(i,i)$  is the first zero element in the diagonal. The factorization from this point might be incomplete.

### roc solver\_<type>getf2\_npvt\_batched()

rocblas\_status **roc solver\_zgetf2\_npvt\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgetf2\_npvt\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgetf2\_npvt\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgetf2\_npvt\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*const A[], const rocblas\_int lda, rocblas\_int \*info, const rocblas\_int batch\_count)

GETF2\_NPVT\_BATCHED computes the LU factorization of a batch of general  $m$ -by- $n$  matrices without partial pivoting.

(This is the unblocked Level-2-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with small and mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

The factorization of matrix  $A_i$  in the batch has the form

$$A_i = L_i * U_i$$

where  $L_i$  is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and  $U_i$  is upper triangular (upper trapezoidal if  $m < n$ ).

Note: Although this routine can offer better performance, Gaussian elimination without pivoting is not backward stable. If numerical accuracy is compromised, use the legacy-LAPACK-like API GETF2 routines instead.

#### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns 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  $lda \cdot n$ .

On entry, the  $m$ -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]** rocblas\_int.  $lda \geq m$ .

Specifies the leading dimension of matrices  $A_i$ .

- **info** – **[out]** pointer to rocblas\_int. Array of batch\_count 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 element in the diagonal. The factorization from this point might be incomplete.

- **batch\_count** – **[in]** rocblas\_int.  $batch\_count \geq 0$ .

Number of matrices in the batch.

### roc solver\_<type>getf2\_npvt\_strided\_batched()

rocblas\_status **roc solver\_zgetf2\_npvt\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_cgetf2\_npvt\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_dgetf2\_npvt\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

rocblas\_status **roc solver\_sgetf2\_npvt\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

GETF2\_NPVT\_STRIDED\_BATCHED computes the LU factorization of a batch of general  $m$ -by- $n$  matrices without partial pivoting.

(This is the unblocked Level-2-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with small and mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

The factorization of matrix  $A_i$  in the batch has the form

$$A_i = L_i * U_i$$

where  $L_i$  is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and  $U_i$  is upper triangular (upper trapezoidal if  $m < n$ ).

Note: Although this routine can offer better performance, Gaussian elimination without pivoting is not backward stable. If numerical accuracy is compromised, use the legacy-LAPACK-like API GETF2 routines instead.

**Parameters**

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns 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 m-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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_i$  and the next one  $A_{(i+1)}$ . There is no restriction for the value of strideA. Normal use case is  $strideA \geq lda * n$
- **info** – [out] pointer to rocblas\_int. Array of batch\_count 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 element in the diagonal. The factorization from this point might be incomplete.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

**roc solver\_<type>getrf\_npvt()**

rocblas\_status **roc solver\_zgetrf\_npvt**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_double\_complex \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_cgetrf\_npvt**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, rocblas\_float\_complex \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_dgetrf\_npvt**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, double \*A, const rocblas\_int lda, rocblas\_int \*info)

rocblas\_status **roc solver\_sgetrf\_npvt**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, rocblas\_int \*info)

GETRF\_NPVT computes the LU factorization of a general m-by-n matrix A without partial pivoting.

(This is the blocked Level-3-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

The factorization has the form

$$A = L * U$$



where L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

Note: Although this routine can offer better performance, Gaussian elimination without pivoting is not backward stable. If numerical accuracy is compromised, use the legacy-LAPACK-like API GETRF routines instead.

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of the matrix A.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns of the matrix A.
- **A** – [inout] pointer to type. Array on the GPU of dimension  $lda \times n$ .  
On entry, the m-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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of A.
- **info** – [out] pointer to a rocblas\_int on the GPU.  
If  $info = 0$ , successful exit. If  $info = i > 0$ , U is singular.  $U(i,i)$  is the first zero element in the diagonal. The factorization from this point might be incomplete.

### roc solver\_<type>getrf\_npvt\_batched()

```
rocblas_status roc solver_zgetrf_npvt_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int
n, rocblas_double_complex *const A[], const rocblas_int lda,
rocblas_int *info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_cgetrf_npvt_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int
n, rocblas_float_complex *const A[], const rocblas_int lda,
rocblas_int *info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_dgetrf_npvt_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int
n, double *const A[], const rocblas_int lda, rocblas_int *info,
const rocblas_int batch_count)
```

```
rocblas_status roc solver_sgetrf_npvt_batched(rocblas_handle handle, const rocblas_int m, const rocblas_int
n, float *const A[], const rocblas_int lda, rocblas_int *info,
const rocblas_int batch_count)
```

GETRF\_NPVT\_BATCHED computes the LU factorization of a batch of general m-by-n matrices without partial pivoting.

(This is the blocked Level-3-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

The factorization of matrix  $A_i$  in the batch has the form

$$A_i = L_i * U_i$$

where  $L_i$  is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and  $U_i$  is upper triangular (upper trapezoidal if  $m < n$ ).

Note: Although this routine can offer better performance, Gaussian elimination without pivoting is not backward stable. If numerical accuracy is compromised, use the legacy-LAPACK-like API GETRF routines instead.

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns 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  $lda * n$ .  
On entry, the  $m$ -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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **info** – [out] pointer to rocblas\_int. Array of `batch_count` 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 element in the diagonal. The factorization from this point might be incomplete.
- **batch\_count** – [in] rocblas\_int. `batch_count`  $\geq 0$ .  
Number of matrices in the batch.

### roc solver\_<type>getrf\_npvt\_strided\_batched()

```
rocblas_status roc solver_zgetrf_npvt_strided_batched(rocblas_handle handle, const rocblas_int m, const
                                                    rocblas_int n, rocblas_double_complex *A, const
                                                    rocblas_int lda, const rocblas_stride strideA,
                                                    rocblas_int *info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_cgetrf_npvt_strided_batched(rocblas_handle handle, const rocblas_int m, const
                                                    rocblas_int n, rocblas_float_complex *A, const
                                                    rocblas_int lda, const rocblas_stride strideA,
                                                    rocblas_int *info, const rocblas_int batch_count)
```

```
rocblas_status roc solver_dgetrf_npvt_strided_batched(rocblas_handle handle, const rocblas_int m, const
                                                    rocblas_int n, double *A, const rocblas_int lda,
                                                    const rocblas_stride strideA, rocblas_int *info, const
                                                    rocblas_int batch_count)
```

rocblas\_status **roc solver\_sgetrf\_npvt\_strided\_batched**(rocblas\_handle handle, const rocblas\_int m, const rocblas\_int n, float \*A, const rocblas\_int lda, const rocblas\_stride strideA, rocblas\_int \*info, const rocblas\_int batch\_count)

GETRF\_NPVT\_STRIDED\_BATCHED computes the LU factorization of a batch of general m-by-n matrices without partial pivoting.

(This is the blocked Level-3-BLAS version of the algorithm. An optimized internal implementation without rocBLAS calls could be executed with mid-size matrices if optimizations are enabled (default option). For more details see the section “tuning rocSOLVER performance” on the User’s guide).

The factorization of matrix  $A_i$  in the batch has the form

$$A_i = L_i * U_i$$

where  $L_i$  is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and  $U_i$  is upper triangular (upper trapezoidal if  $m < n$ ).

Note: Although this routine can offer better performance, Gaussian elimination without pivoting is not backward stable. If numerical accuracy is compromised, use the legacy-LAPACK-like API GETRF routines instead.

### Parameters

- **handle** – [in] rocblas\_handle.
- **m** – [in] rocblas\_int.  $m \geq 0$ .  
The number of rows of all matrices  $A_i$  in the batch.
- **n** – [in] rocblas\_int.  $n \geq 0$ .  
The number of columns 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 m-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] rocblas\_int.  $lda \geq m$ .  
Specifies the leading dimension of matrices  $A_i$ .
- **strideA** – [in] rocblas\_stride.  
Stride from the start of one matrix  $A_i$  and the next one  $A_{(i+1)}$ . There is no restriction for the value of strideA. Normal use case is  $strideA \geq lda * n$
- **info** – [out] pointer to rocblas\_int. Array of batch\_count 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 element in the diagonal. The factorization from this point might be incomplete.
- **batch\_count** – [in] rocblas\_int.  $batch\_count \geq 0$ .  
Number of matrices in the batch.

## 2.7.6 Deprecated

### 2.7.6.1 Types

See the [rocBLAS types](#) documentation for information on the suggested replacements.

#### **rocsolver\_int**

typedef rocblas\_int **rocsolver\_int**

*Deprecated:*

Use rocblas\_int.

Deprecated since version 3.5: Use rocblas\_int.

#### **rocsolver\_handle**

typedef rocblas\_handle **rocsolver\_handle**

*Deprecated:*

Use rocblas\_handle.

Deprecated since version 3.5: Use rocblas\_handle.

#### **rocsolver\_direction**

typedef *rocblas\_direct* **rocsolver\_direction**

*Deprecated:*

Use rocblas\_direct

Deprecated since version 3.5: Use *rocblas\_direct*.

#### **rocsolver\_storev**

typedef *rocblas\_storev* **rocsolver\_storev**

*Deprecated:*

Use rocblas\_storev.

Deprecated since version 3.5: Use *rocblas\_storev*.

### **roc solver\_operation**

typedef rocblas\_operation **roc solver\_operation**

*Deprecated:*

Use rocblas\_operation.

Deprecated since version 3.5: Use rocblas\_operation.

### **roc solver\_fill**

typedef rocblas\_fill **roc solver\_fill**

*Deprecated:*

Use rocblas\_fill.

Deprecated since version 3.5: Use rocblas\_fill.

### **roc solver\_diagonal**

typedef rocblas\_diagonal **roc solver\_diagonal**

*Deprecated:*

Use rocblas\_diagonal.

Deprecated since version 3.5: Use rocblas\_diagonal.

### **roc solver\_side**

typedef rocblas\_side **roc solver\_side**

*Deprecated:*

Use rocblas\_side.

Deprecated since version 3.5: Use rocblas\_side.

### **roc solver\_status**

typedef rocblas\_status **roc solver\_status**

*Deprecated:*

Use rocblas\_status.

Deprecated since version 3.5: Use rocblas\_status.

### 2.7.6.2 Auxiliary Functions

See the [rocBLAS auxiliary functions](#) documentation for information on the suggested replacements.

#### **roc solver\_create\_handle()**

*roc solver\_status* **roc solver\_create\_handle**(*roc solver\_handle* \*handle)  
Creates a handle and sets the pointer mode to `rocblas_pointer_mode_device`.

*Deprecated:*  
Use `rocblas_create_handle`.

Deprecated since version 3.5: Use `rocblas_create_handle()`.

#### **roc solver\_destroy\_handle()**

*roc solver\_status* **roc solver\_destroy\_handle**(*roc solver\_handle* handle)

*Deprecated:*  
Use `rocblas_destroy_handle`.

Deprecated since version 3.5: Use `rocblas_destroy_handle()`.

#### **roc solver\_set\_stream()**

*roc solver\_status* **roc solver\_set\_stream**(*roc solver\_handle* handle, `hipStream_t` stream)

*Deprecated:*  
Use `rocblas_set_stream`.

Deprecated since version 3.5: Use `rocblas_set_stream()`.

#### **roc solver\_get\_stream()**

*roc solver\_status* **roc solver\_get\_stream**(*roc solver\_handle* handle, `hipStream_t` \*stream)

*Deprecated:*  
Use `rocblas_get_stream`.

Deprecated since version 3.5: Use `rocblas_get_stream()`.

**roc solver\_set\_vector()**

*roc solver\_status* **roc solver\_set\_vector**(*roc solver\_int* n, *roc solver\_int* elem\_size, const void \*x, *roc solver\_int* incx, void \*y, *roc solver\_int* incy)

*Deprecated:*

Use rocblas\_set\_vector.

Deprecated since version 3.5: Use rocblas\_set\_vector().

**roc solver\_get\_vector()**

*roc solver\_status* **roc solver\_get\_vector**(*roc solver\_int* n, *roc solver\_int* elem\_size, const void \*x, *roc solver\_int* incx, void \*y, *roc solver\_int* incy)

*Deprecated:*

Use rocblas\_get\_vector.

Deprecated since version 3.5: Use rocblas\_get\_vector().

**roc solver\_set\_matrix()**

*roc solver\_status* **roc solver\_set\_matrix**(*roc solver\_int* rows, *roc solver\_int* cols, *roc solver\_int* elem\_size, const void \*a, *roc solver\_int* lda, void \*b, *roc solver\_int* ldb)

*Deprecated:*

Use rocblas\_set\_matrix.

Deprecated since version 3.5: Use rocblas\_set\_matrix().

**roc solver\_get\_matrix()**

*roc solver\_status* **roc solver\_get\_matrix**(*roc solver\_int* rows, *roc solver\_int* cols, *roc solver\_int* elem\_size, const void \*a, *roc solver\_int* lda, void \*b, *roc solver\_int* ldb)

*Deprecated:*

Use rocblas\_get\_matrix.

Deprecated since version 3.5: Use rocblas\_get\_matrix().





## LIBRARY DESIGN DOCUMENTATION

### 3.1 Introduction

More to come later...

### 3.2 Batch rocSOLVER

More to come later...

### 3.3 Clients

rocSOLVER has a basic/preliminary infrastructure for testing and benchmarking similar to that of rocBLAS.

On a normal installation, client binaries `rocsolver-test` and `rocsolver-bench` should be located in the directory `<rocsolverDIR>/build/clients/staging`.

#### 3.3.1 Testing rocSOLVER

`rocsolver-test` executes a suite of [Google tests](#) (*gtest*) that verifies the correct functioning of the library; the results computed by rocSOLVER, for random input data, are compared with the results computed by [NETLib LAPACK](#) on the CPU.

Calling the rocSOLVER `gtest` client with the `-help` flag

```
./rocsolver-test --help
```

returns information on different flags that control the behavior of the `gtests`.

### 3.3.2 Benchmarking rocSOLVER

`rocsolver-bench` runs any rocSOLVER function with random data of the specified dimensions; it compares the computed results, and provides basic performance information (as for now, execution times).

Similarly,

```
./rocsolver-bench --help
```

returns information on how to use the rocSOLVER benchmark client.

## R

- rocblas\_direct (*C enum*), 18
- rocblas\_direct.rocblas\_backward\_direction (*C enumerator*), 18
- rocblas\_direct.rocblas\_forward\_direction (*C enumerator*), 18
- rocblas\_eform (*C enum*), 19
- rocblas\_eform.rocblas\_eform\_abx (*C enumerator*), 19
- rocblas\_eform.rocblas\_eform\_ax (*C enumerator*), 19
- rocblas\_eform.rocblas\_eform\_bax (*C enumerator*), 19
- rocblas\_evect (*C enum*), 19
- rocblas\_evect.rocblas\_evect\_none (*C enumerator*), 19
- rocblas\_evect.rocblas\_evect\_original (*C enumerator*), 19
- rocblas\_evect.rocblas\_evect\_tridiagonal (*C enumerator*), 19
- rocblas\_storev (*C enum*), 18
- rocblas\_storev.rocblas\_column\_wise (*C enumerator*), 18
- rocblas\_storev.rocblas\_row\_wise (*C enumerator*), 18
- rocblas\_svect (*C enum*), 18
- rocblas\_svect.rocblas\_svect\_all (*C enumerator*), 18
- rocblas\_svect.rocblas\_svect\_none (*C enumerator*), 18
- rocblas\_svect.rocblas\_svect\_overwrite (*C enumerator*), 18
- rocblas\_svect.rocblas\_svect\_singular (*C enumerator*), 18
- rocblas\_workmode (*C enum*), 19
- rocblas\_workmode.rocblas\_inplace (*C enumerator*), 19
- rocblas\_workmode.rocblas\_outofplace (*C enumerator*), 19
- roc solver\_cbdsq (C function), 29
- roc solver\_cgebd2 (C function), 104
- roc solver\_cgebd2\_batched (C function), 105
- roc solver\_cgebd2\_strided\_batched (C function), 107
- roc solver\_cgebrd (C function), 109
- roc solver\_cgebrd\_batched (C function), 111
- roc solver\_cgebrd\_strided\_batched (C function), 113
- roc solver\_cgels (C function), 96
- roc solver\_cgels\_batched (C function), 97
- roc solver\_cgels\_strided\_batched (C function), 99
- roc solver\_cgels\_batched (C function), 100
- roc solver\_cgels\_batched (C function), 101
- roc solver\_cgels\_strided\_batched (C function), 102
- roc solver\_cgels (C function), 158
- roc solver\_cgels\_batched (C function), 160
- roc solver\_cgels\_strided\_batched (C function), 161
- roc solver\_cgeql2 (C function), 89
- roc solver\_cgeql2\_batched (C function), 90
- roc solver\_cgeql2\_strided\_batched (C function), 91
- roc solver\_cgeqlf (C function), 93
- roc solver\_cgeqlf\_batched (C function), 94
- roc solver\_cgeqlf\_strided\_batched (C function), 95
- roc solver\_cgeqr2 (C function), 81
- roc solver\_cgeqr2\_batched (C function), 82
- roc solver\_cgeqr2\_strided\_batched (C function), 84
- roc solver\_cgeqrf (C function), 85
- roc solver\_cgeqrf\_batched (C function), 86
- roc solver\_cgeqrf\_strided\_batched (C function), 87
- roc solver\_cgesvd (C function), 180
- roc solver\_cgesvd\_batched (C function), 183
- roc solver\_cgesvd\_strided\_batched (C function), 186
- roc solver\_cgetf2 (C function), 74
- roc solver\_cgetf2\_batched (C function), 75
- roc solver\_cgetf2\_npvt (C function), 193
- roc solver\_cgetf2\_npvt\_batched (C function), 194

- roc solver\_cgetf2\_npvt\_strided\_batched (*C function*), 195
- roc solver\_cgetf2\_strided\_batched (*C function*), 76
- roc solver\_cgetrf (*C function*), 77
- roc solver\_cgetrf\_batched (*C function*), 78
- roc solver\_cgetrf\_npvt (*C function*), 196
- roc solver\_cgetrf\_npvt\_batched (*C function*), 197
- roc solver\_cgetrf\_npvt\_strided\_batched (*C function*), 198
- roc solver\_cgetrf\_strided\_batched (*C function*), 80
- roc solver\_cgetri (*C function*), 151
- roc solver\_cgetri\_batched (*C function*), 152
- roc solver\_cgetri\_outofplace (*C function*), 189
- roc solver\_cgetri\_outofplace\_batched (*C function*), 190
- roc solver\_cgetri\_outofplace\_strided\_batched (*C function*), 191
- roc solver\_cgetri\_strided\_batched (*C function*), 153
- roc solver\_cgetrs (*C function*), 154
- roc solver\_cgetrs\_batched (*C function*), 155
- roc solver\_cgetrs\_strided\_batched (*C function*), 156
- roc solver\_cheev (*C function*), 167
- roc solver\_cheev\_batched (*C function*), 168
- roc solver\_cheev\_strided\_batched (*C function*), 169
- roc solver\_chegs2 (*C function*), 136
- roc solver\_chegs2\_batched (*C function*), 137
- roc solver\_chegs2\_strided\_batched (*C function*), 138
- roc solver\_chegst (*C function*), 144
- roc solver\_chegst\_batched (*C function*), 145
- roc solver\_chegst\_strided\_batched (*C function*), 146
- roc solver\_chegv (*C function*), 175
- roc solver\_chegv\_batched (*C function*), 177
- roc solver\_chegv\_strided\_batched (*C function*), 178
- roc solver\_chetd2 (*C function*), 119
- roc solver\_chetd2\_batched (*C function*), 120
- roc solver\_chetd2\_strided\_batched (*C function*), 122
- roc solver\_chetrd (*C function*), 128
- roc solver\_chetrd\_batched (*C function*), 129
- roc solver\_chetrd\_strided\_batched (*C function*), 130
- roc solver\_clabrd (*C function*), 27
- roc solver\_clacgv (*C function*), 20
- roc solver\_clarf (*C function*), 24
- roc solver\_clarfb (*C function*), 25
- roc solver\_clarfg (*C function*), 22
- roc solver\_clarft (*C function*), 23
- roc solver\_claswp (*C function*), 21
- roc solver\_clatrd (*C function*), 31
- roc solver\_cpotf2 (*C function*), 67
- roc solver\_cpotf2\_batched (*C function*), 68
- roc solver\_cpotf2\_strided\_batched (*C function*), 69
- roc solver\_cpotrff (*C function*), 70
- roc solver\_cpotrff\_batched (*C function*), 71
- roc solver\_cpotrff\_strided\_batched (*C function*), 72
- roc solver\_create\_handle (*C function*), 202
- roc solver\_csteqr (*C function*), 33
- roc solver\_ctrtri (*C function*), 148
- roc solver\_ctrtri\_batched (*C function*), 149
- roc solver\_ctrtri\_strided\_batched (*C function*), 150
- roc solver\_cung2l (*C function*), 54
- roc solver\_cung2r (*C function*), 50
- roc solver\_cungbr (*C function*), 55
- roc solver\_cungl2 (*C function*), 52
- roc solver\_cunglq (*C function*), 53
- roc solver\_cungql (*C function*), 54
- roc solver\_cungqr (*C function*), 51
- roc solver\_cungtr (*C function*), 56
- roc solver\_cum2l (*C function*), 62
- roc solver\_cum2r (*C function*), 57
- roc solver\_cumbr (*C function*), 64
- roc solver\_cumml2 (*C function*), 60
- roc solver\_cummlq (*C function*), 61
- roc solver\_cumml (*C function*), 63
- roc solver\_cumqr (*C function*), 58
- roc solver\_cumtr (*C function*), 66
- roc solver\_dbdsqr (*C function*), 29
- roc solver\_destroy\_handle (*C function*), 202
- roc solver\_dgebd2 (*C function*), 104
- roc solver\_dgebd2\_batched (*C function*), 105
- roc solver\_dgebd2\_strided\_batched (*C function*), 107
- roc solver\_dgebrd (*C function*), 109
- roc solver\_dgebrd\_batched (*C function*), 111
- roc solver\_dgebrd\_strided\_batched (*C function*), 113
- roc solver\_dgelq2 (*C function*), 96
- roc solver\_dgelq2\_batched (*C function*), 97
- roc solver\_dgelq2\_strided\_batched (*C function*), 99
- roc solver\_dgelqf (*C function*), 100
- roc solver\_dgelqf\_batched (*C function*), 101
- roc solver\_dgelqf\_strided\_batched (*C function*), 102
- roc solver\_dgels (*C function*), 158
- roc solver\_dgels\_batched (*C function*), 160

- roc solver\_dgels\_strided\_batched (*C function*), 161
- roc solver\_dgeql2 (*C function*), 89
- roc solver\_dgeql2\_batched (*C function*), 90
- roc solver\_dgeql2\_strided\_batched (*C function*), 91
- roc solver\_dgeqlf (*C function*), 93
- roc solver\_dgeqlf\_batched (*C function*), 94
- roc solver\_dgeqlf\_strided\_batched (*C function*), 95
- roc solver\_dgeqr2 (*C function*), 81
- roc solver\_dgeqr2\_batched (*C function*), 82
- roc solver\_dgeqr2\_strided\_batched (*C function*), 84
- roc solver\_dgeqrf (*C function*), 85
- roc solver\_dgeqrf\_batched (*C function*), 86
- roc solver\_dgeqrf\_strided\_batched (*C function*), 87
- roc solver\_dgesvd (*C function*), 180
- roc solver\_dgesvd\_batched (*C function*), 183
- roc solver\_dgesvd\_strided\_batched (*C function*), 186
- roc solver\_dgetf2 (*C function*), 74
- roc solver\_dgetf2\_batched (*C function*), 75
- roc solver\_dgetf2\_npvt (*C function*), 193
- roc solver\_dgetf2\_npvt\_batched (*C function*), 194
- roc solver\_dgetf2\_npvt\_strided\_batched (*C function*), 195
- roc solver\_dgetf2\_strided\_batched (*C function*), 76
- roc solver\_dgetrf (*C function*), 77
- roc solver\_dgetrf\_batched (*C function*), 79
- roc solver\_dgetrf\_npvt (*C function*), 196
- roc solver\_dgetrf\_npvt\_batched (*C function*), 197
- roc solver\_dgetrf\_npvt\_strided\_batched (*C function*), 198
- roc solver\_dgetrf\_strided\_batched (*C function*), 80
- roc solver\_dgetri (*C function*), 151
- roc solver\_dgetri\_batched (*C function*), 152
- roc solver\_dgetri\_outofplace (*C function*), 189
- roc solver\_dgetri\_outofplace\_batched (*C function*), 190
- roc solver\_dgetri\_outofplace\_strided\_batched (*C function*), 191
- roc solver\_dgetri\_strided\_batched (*C function*), 153
- roc solver\_dgetrs (*C function*), 154
- roc solver\_dgetrs\_batched (*C function*), 155
- roc solver\_dgetrs\_strided\_batched (*C function*), 157
- roc solver\_diagonal (*C type*), 201
- roc solver\_direction (*C type*), 200
- roc solver\_dlabrd (*C function*), 27
- roc solver\_dlarf (*C function*), 24
- roc solver\_dlarfb (*C function*), 25
- roc solver\_dlarfg (*C function*), 22
- roc solver\_dlarft (*C function*), 23
- roc solver\_dlaswp (*C function*), 21
- roc solver\_dlatrd (*C function*), 31
- roc solver\_dorg2l (*C function*), 37
- roc solver\_dorg2r (*C function*), 34
- roc solver\_dorgbr (*C function*), 39
- roc solver\_dorgl2 (*C function*), 36
- roc solver\_dorglq (*C function*), 36
- roc solver\_dorgql (*C function*), 38
- roc solver\_dorgqr (*C function*), 35
- roc solver\_dorgtr (*C function*), 40
- roc solver\_dorm2l (*C function*), 45
- roc solver\_dorm2r (*C function*), 41
- roc solver\_dormbr (*C function*), 48
- roc solver\_dorml2 (*C function*), 43
- roc solver\_dormlq (*C function*), 44
- roc solver\_dormql (*C function*), 46
- roc solver\_dormqr (*C function*), 42
- roc solver\_dormtr (*C function*), 49
- roc solver\_dpotf2 (*C function*), 67
- roc solver\_dpotf2\_batched (*C function*), 68
- roc solver\_dpotf2\_strided\_batched (*C function*), 69
- roc solver\_dpotrf (*C function*), 70
- roc solver\_dpotrf\_batched (*C function*), 71
- roc solver\_dpotrf\_strided\_batched (*C function*), 72
- roc solver\_dsteqr (*C function*), 33
- roc solver\_dsterf (*C function*), 32
- roc solver\_dsyev (*C function*), 163
- roc solver\_dsyev\_batched (*C function*), 164
- roc solver\_dsyev\_strided\_batched (*C function*), 165
- roc solver\_dsygs2 (*C function*), 132
- roc solver\_dsygs2\_batched (*C function*), 133
- roc solver\_dsygs2\_strided\_batched (*C function*), 134
- roc solver\_dsygst (*C function*), 140
- roc solver\_dsygst\_batched (*C function*), 141
- roc solver\_dsygst\_strided\_batched (*C function*), 142
- roc solver\_dsygv (*C function*), 170
- roc solver\_dsygv\_batched (*C function*), 172
- roc solver\_dsygv\_strided\_batched (*C function*), 173
- roc solver\_dsytd2 (*C function*), 115
- roc solver\_dsytd2\_batched (*C function*), 116
- roc solver\_dsytd2\_strided\_batched (*C function*), 118
- roc solver\_dsytrd (*C function*), 123
- roc solver\_dsytrd\_batched (*C function*), 125

- roc solver\_dsytrd\_strided\_batched (*C function*), 126
- roc solver\_dtrtri (*C function*), 148
- roc solver\_dtrtri\_batched (*C function*), 149
- roc solver\_dtrtri\_strided\_batched (*C function*), 150
- roc solver\_fill (*C type*), 201
- roc solver\_get\_matrix (*C function*), 203
- roc solver\_get\_stream (*C function*), 202
- roc solver\_get\_vector (*C function*), 203
- roc solver\_handle (*C type*), 200
- roc solver\_int (*C type*), 200
- roc solver\_log\_begin (*C function*), 20
- roc solver\_log\_end (*C function*), 20
- roc solver\_log\_flush\_profile (*C function*), 20
- roc solver\_log\_restore\_defaults (*C function*), 20
- roc solver\_log\_set\_layer\_mode (*C function*), 20
- roc solver\_log\_set\_max\_levels (*C function*), 20
- roc solver\_log\_write\_profile (*C function*), 20
- roc solver\_operation (*C type*), 201
- roc solver\_sbdsqr (*C function*), 29
- roc solver\_set\_matrix (*C function*), 203
- roc solver\_set\_stream (*C function*), 202
- roc solver\_set\_vector (*C function*), 203
- roc solver\_sgebd2 (*C function*), 104
- roc solver\_sgebd2\_batched (*C function*), 105
- roc solver\_sgebd2\_strided\_batched (*C function*), 108
- roc solver\_sgebrd (*C function*), 110
- roc solver\_sgebrd\_batched (*C function*), 111
- roc solver\_sgebrd\_strided\_batched (*C function*), 113
- roc solver\_sgelq2 (*C function*), 96
- roc solver\_sgelq2\_batched (*C function*), 97
- roc solver\_sgelq2\_strided\_batched (*C function*), 99
- roc solver\_sgelqf (*C function*), 100
- roc solver\_sgelqf\_batched (*C function*), 101
- roc solver\_sgelqf\_strided\_batched (*C function*), 103
- roc solver\_sgels (*C function*), 158
- roc solver\_sgels\_batched (*C function*), 160
- roc solver\_sgels\_strided\_batched (*C function*), 162
- roc solver\_sgeql2 (*C function*), 89
- roc solver\_sgeql2\_batched (*C function*), 90
- roc solver\_sgeql2\_strided\_batched (*C function*), 91
- roc solver\_sgeqlf (*C function*), 93
- roc solver\_sgeqlf\_batched (*C function*), 94
- roc solver\_sgeqlf\_strided\_batched (*C function*), 95
- roc solver\_sgeqr2 (*C function*), 81
- roc solver\_sgeqr2\_batched (*C function*), 83
- roc solver\_sgeqr2\_strided\_batched (*C function*), 84
- roc solver\_sgeqrf (*C function*), 85
- roc solver\_sgeqrf\_batched (*C function*), 86
- roc solver\_sgeqrf\_strided\_batched (*C function*), 88
- roc solver\_sgesvd (*C function*), 181
- roc solver\_sgesvd\_batched (*C function*), 183
- roc solver\_sgesvd\_strided\_batched (*C function*), 186
- roc solver\_sgetf2 (*C function*), 74
- roc solver\_sgetf2\_batched (*C function*), 75
- roc solver\_sgetf2\_npvt (*C function*), 193
- roc solver\_sgetf2\_npvt\_batched (*C function*), 194
- roc solver\_sgetf2\_npvt\_strided\_batched (*C function*), 195
- roc solver\_sgetf2\_strided\_batched (*C function*), 76
- roc solver\_sgetrf (*C function*), 78
- roc solver\_sgetrf\_batched (*C function*), 79
- roc solver\_sgetrf\_npvt (*C function*), 196
- roc solver\_sgetrf\_npvt\_batched (*C function*), 197
- roc solver\_sgetrf\_npvt\_strided\_batched (*C function*), 198
- roc solver\_sgetrf\_strided\_batched (*C function*), 80
- roc solver\_sgetri (*C function*), 151
- roc solver\_sgetri\_batched (*C function*), 152
- roc solver\_sgetri\_outofplace (*C function*), 189
- roc solver\_sgetri\_outofplace\_batched (*C function*), 190
- roc solver\_sgetri\_outofplace\_strided\_batched (*C function*), 192
- roc solver\_sgetri\_strided\_batched (*C function*), 153
- roc solver\_sgetrs (*C function*), 154
- roc solver\_sgetrs\_batched (*C function*), 155
- roc solver\_sgetrs\_strided\_batched (*C function*), 157
- roc solver\_side (*C type*), 201
- roc solver\_slabrd (*C function*), 27
- roc solver\_slarf (*C function*), 24
- roc solver\_slarfb (*C function*), 25
- roc solver\_slarfg (*C function*), 22
- roc solver\_slarft (*C function*), 23
- roc solver\_slaswp (*C function*), 21
- roc solver\_slatrd (*C function*), 31
- roc solver\_sorg2l (*C function*), 37
- roc solver\_sorg2r (*C function*), 34
- roc solver\_sorgbr (*C function*), 39
- roc solver\_sorgl2 (*C function*), 36
- roc solver\_sorglq (*C function*), 36
- roc solver\_sorgql (*C function*), 38
- roc solver\_sorgqr (*C function*), 35

- rocsolver\_sorgtr (*C function*), 40  
 rocsolver\_sorm2l (*C function*), 45  
 rocsolver\_sorm2r (*C function*), 41  
 rocsolver\_sormbr (*C function*), 48  
 rocsolver\_sorml2 (*C function*), 43  
 rocsolver\_sormlq (*C function*), 44  
 rocsolver\_sormql (*C function*), 46  
 rocsolver\_sormqr (*C function*), 42  
 rocsolver\_sormtr (*C function*), 49  
 rocsolver\_spotf2 (*C function*), 67  
 rocsolver\_spotf2\_batched (*C function*), 68  
 rocsolver\_spotf2\_strided\_batched (*C function*), 69  
 rocsolver\_spotrf (*C function*), 71  
 rocsolver\_spotrf\_batched (*C function*), 71  
 rocsolver\_spotrf\_strided\_batched (*C function*), 73  
 rocsolver\_ssteqr (*C function*), 33  
 rocsolver\_ssterf (*C function*), 32  
 rocsolver\_ssyev (*C function*), 163  
 rocsolver\_ssyev\_batched (*C function*), 164  
 rocsolver\_ssyev\_strided\_batched (*C function*), 165  
 rocsolver\_ssygs2 (*C function*), 132  
 rocsolver\_ssygs2\_batched (*C function*), 133  
 rocsolver\_ssygs2\_strided\_batched (*C function*), 134  
 rocsolver\_ssygst (*C function*), 140  
 rocsolver\_ssygst\_batched (*C function*), 141  
 rocsolver\_ssygst\_strided\_batched (*C function*), 142  
 rocsolver\_ssygv (*C function*), 170  
 rocsolver\_ssygv\_batched (*C function*), 172  
 rocsolver\_ssygv\_strided\_batched (*C function*), 173  
 rocsolver\_ssytd2 (*C function*), 115  
 rocsolver\_ssytd2\_batched (*C function*), 116  
 rocsolver\_ssytd2\_strided\_batched (*C function*), 118  
 rocsolver\_ssytrd (*C function*), 123  
 rocsolver\_ssytrd\_batched (*C function*), 125  
 rocsolver\_ssytrd\_strided\_batched (*C function*), 126  
 rocsolver\_status (*C type*), 201  
 rocsolver\_storev (*C type*), 200  
 rocsolver\_strtri (*C function*), 148  
 rocsolver\_strtri\_batched (*C function*), 149  
 rocsolver\_strtri\_strided\_batched (*C function*), 150  
 rocsolver\_zbdsqr (*C function*), 29  
 rocsolver\_zgebd2 (*C function*), 104  
 rocsolver\_zgebd2\_batched (*C function*), 105  
 rocsolver\_zgebd2\_strided\_batched (*C function*), 107  
 rocsolver\_zgebrd (*C function*), 109  
 rocsolver\_zgebrd\_batched (*C function*), 111  
 rocsolver\_zgebrd\_strided\_batched (*C function*), 113  
 rocsolver\_zgelq2 (*C function*), 96  
 rocsolver\_zgelq2\_batched (*C function*), 97  
 rocsolver\_zgelq2\_strided\_batched (*C function*), 99  
 rocsolver\_zgelqf (*C function*), 100  
 rocsolver\_zgelqf\_batched (*C function*), 101  
 rocsolver\_zgelqf\_strided\_batched (*C function*), 102  
 rocsolver\_zgels (*C function*), 158  
 rocsolver\_zgels\_batched (*C function*), 160  
 rocsolver\_zgels\_strided\_batched (*C function*), 161  
 rocsolver\_zgeql2 (*C function*), 89  
 rocsolver\_zgeql2\_batched (*C function*), 90  
 rocsolver\_zgeql2\_strided\_batched (*C function*), 91  
 rocsolver\_zgeqlf (*C function*), 93  
 rocsolver\_zgeqlf\_batched (*C function*), 94  
 rocsolver\_zgeqlf\_strided\_batched (*C function*), 95  
 rocsolver\_zgeqr2 (*C function*), 81  
 rocsolver\_zgeqr2\_batched (*C function*), 82  
 rocsolver\_zgeqr2\_strided\_batched (*C function*), 84  
 rocsolver\_zgeqrf (*C function*), 85  
 rocsolver\_zgeqrf\_batched (*C function*), 86  
 rocsolver\_zgeqrf\_strided\_batched (*C function*), 87  
 rocsolver\_zgesvd (*C function*), 180  
 rocsolver\_zgesvd\_batched (*C function*), 183  
 rocsolver\_zgesvd\_strided\_batched (*C function*), 186  
 rocsolver\_zgetf2 (*C function*), 74  
 rocsolver\_zgetf2\_batched (*C function*), 75  
 rocsolver\_zgetf2\_npvt (*C function*), 193  
 rocsolver\_zgetf2\_npvt\_batched (*C function*), 194  
 rocsolver\_zgetf2\_npvt\_strided\_batched (*C function*), 195  
 rocsolver\_zgetf2\_strided\_batched (*C function*), 76  
 rocsolver\_zgetrf (*C function*), 77  
 rocsolver\_zgetrf\_batched (*C function*), 78  
 rocsolver\_zgetrf\_npvt (*C function*), 196  
 rocsolver\_zgetrf\_npvt\_batched (*C function*), 197  
 rocsolver\_zgetrf\_npvt\_strided\_batched (*C function*), 198  
 rocsolver\_zgetrf\_strided\_batched (*C function*), 80  
 rocsolver\_zgetri (*C function*), 151  
 rocsolver\_zgetri\_batched (*C function*), 152

roc solver\_zgetri\_outofplace (C function), 189  
roc solver\_zgetri\_outofplace\_batched (C function), 190  
roc solver\_zgetri\_outofplace\_strided\_batched (C function), 191  
roc solver\_zgetri\_strided\_batched (C function), 153  
roc solver\_zgetrs (C function), 154  
roc solver\_zgetrs\_batched (C function), 155  
roc solver\_zgetrs\_strided\_batched (C function), 156  
roc solver\_zheev (C function), 167  
roc solver\_zheev\_batched (C function), 168  
roc solver\_zheev\_strided\_batched (C function), 169  
roc solver\_zhegs2 (C function), 136  
roc solver\_zhegs2\_batched (C function), 137  
roc solver\_zhegs2\_strided\_batched (C function), 138  
roc solver\_zhegst (C function), 144  
roc solver\_zhegst\_batched (C function), 145  
roc solver\_zhegst\_strided\_batched (C function), 146  
roc solver\_zhegv (C function), 175  
roc solver\_zhegv\_batched (C function), 177  
roc solver\_zhegv\_strided\_batched (C function), 178  
roc solver\_zhetd2 (C function), 119  
roc solver\_zhetd2\_batched (C function), 120  
roc solver\_zhetd2\_strided\_batched (C function), 122  
roc solver\_zhetrd (C function), 128  
roc solver\_zhetrd\_batched (C function), 129  
roc solver\_zhetrd\_strided\_batched (C function), 130  
roc solver\_zlabrd (C function), 27  
roc solver\_zlacgv (C function), 20  
roc solver\_zlarf (C function), 24  
roc solver\_zlarfb (C function), 25  
roc solver\_zlarfg (C function), 22  
roc solver\_zlarft (C function), 23  
roc solver\_zlaswp (C function), 21  
roc solver\_zlatrd (C function), 31  
roc solver\_zpotf2 (C function), 67  
roc solver\_zpotf2\_batched (C function), 68  
roc solver\_zpotf2\_strided\_batched (C function), 69  
roc solver\_zpotrf (C function), 70  
roc solver\_zpotrf\_batched (C function), 71  
roc solver\_zpotrf\_strided\_batched (C function), 72  
roc solver\_zsteqr (C function), 33  
roc solver\_ztrtri (C function), 148  
roc solver\_ztrtri\_batched (C function), 149  
roc solver\_ztrtri\_strided\_batched (C function), 150  
roc solver\_zung2l (C function), 54  
roc solver\_zung2r (C function), 50  
roc solver\_zungbr (C function), 55  
roc solver\_zungl2 (C function), 52  
roc solver\_zunglq (C function), 53  
roc solver\_zungql (C function), 54  
roc solver\_zungqr (C function), 51  
roc solver\_zungtr (C function), 56  
roc solver\_zunm2l (C function), 62  
roc solver\_zunm2r (C function), 57  
roc solver\_zunmbr (C function), 64  
roc solver\_zunml2 (C function), 60  
roc solver\_zunmlq (C function), 61  
roc solver\_zunmql (C function), 63  
roc solver\_zunmqr (C function), 58  
roc solver\_zunmtr (C function), 66