AMD AOCL-Sparse API Guide

version 4.2.0.0 (21 Nov 2023)

Generated by Doxygen 1.9.1

1 The AMD AOCL-Sparse Library

1.1 Introduction

The AMD Optimized CPU Library AOCL-Sparse is a library that contains Basic Linear Algebra Subroutines for sparse matrices and vectors (Sparse BLAS) and is optimized for AMD EPYC and RYZEN family of CPU processors. It implements numerical algorithms in C++ while providing a public-facing C interface.

Functionality of AMD AOCL-Sparse library is organized in the following categories

1. **Sparse Level 1** functions perform vector operations such as dot product, vector additions on sparse vectors, and other similar operations.

2. **Sparse Level 2** functions describe the operations between a matrix in sparse format and a vector in dense format.
3. **Sparse Level 3** functions describe the operations between a matrix in sparse format and one or more dense/sparse matrices.
4. **Iterative sparse solvers** that solve sparse linear system of equations.
5. **Analysis and execute functionalities** for performing optimized operations.
6. Sparse format conversion functions for translating matrices in a variety of sparse storage formats.
7. Sparse auxiliary functions used to perform miscelaneous tasks adjacent to the ones described above.

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

3 File Documentation

3.1 aoclsparse_analysis.h File Reference

aoclsparse_analysis.h provides sparse format analysis subprograms

Functions

- DLL_PUBLIC aoclsparse_status aoclsparse_optimize (aoclsparse_matrix mat)

  Performs data allocations and restructuring operations related to sparse matrices.

- DLL_PUBLIC aoclsparse_status aoclsparse_set_mv_hint (aoclsparse_matrix mat, aoclsparse_operation trans, const aoclsparse_mat_descr descr, aoclsparse_int expected_no_of_calls)

  Provides any hints such as the type of routine, expected no of calls etc.

- DLL_PUBLIC aoclsparse_status aoclsparse_set_lu_smoother_hint (aoclsparse_matrix mat, aoclsparse_operation trans, const aoclsparse_mat_descr descr, aoclsparse_int expected_no_of_calls)

  Provides any hints such as the type of routine, expected no of calls etc.

- DLL_PUBLIC aoclsparse_status aoclsparse_set_sm_hint (aoclsparse_matrix A, aoclsparse_operation trans, const aoclsparse_mat_descr descr, const aoclsparse_order order, const aoclsparse_int dense_matrix_dim, const aoclsparse_int expected_no_of_calls)

  Store user-hints to accelerate the `aoclsparse_?trsm` *triangular-solvers.*

3.1.1 Detailed Description

aoclsparse_analysis.h provides sparse format analysis subprograms

3.1.2 Function Documentation

3.1.2.1 aoclsparse_optimize()

```
DLL_PUBLIC aoclsparse_status aoclsparse_optimize (
        aoclsparse_matrix mat )
```

Performs data allocations and restructuring operations related to sparse matrices.

`aoclsparse_optimize` Sparse matrices are restructured based on matrix analysis, into different storage formats to improve data access and thus performance.

Parameters

in	*mat*	sparse matrix in CSR format and sparse format information inside

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m` is invalid.
aoclsparse_status_invalid_pointer	
aoclsparse_status_internal_error	an internal error occurred.

3.1.2.2 aoclsparse_set_mv_hint()

```
DLL_PUBLIC aoclsparse_status aoclsparse_set_mv_hint (
            aoclsparse_matrix mat,
            aoclsparse_operation trans,
            const aoclsparse_mat_descr descr,
            aoclsparse_int expected_no_of_calls )
```

Provides any hints such as the type of routine, expected no of calls etc.

`aoclsparse_set_mv_hint` sets a hint id for analysis and execute phases of the program to analyse and perform ILU factorization and Solution

Parameters

in	*mat*	sparse matrix in CSR format and sparse format information inside
in	*trans*	Whether in transposed state or not. Transpose operation is not yet supported.
in	*descr*	descriptor of the sparse CSR matrix. Currently, only aoclsparse_matrix_type_general and aoclsparse_matrix_type_symmetric is supported.
in	*expected_no_of_calls*	unused parameter

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m` is invalid.
aoclsparse_status_invalid_pointer	
aoclsparse_status_internal_error	an internal error occurred.

3.1.2.3 aoclsparse_set_lu_smoother_hint()

```
DLL_PUBLIC aoclsparse_status aoclsparse_set_lu_↩
smoother_hint (
            aoclsparse_matrix mat,
            aoclsparse_operation trans,
            const aoclsparse_mat_descr descr,
            aoclsparse_int expected_no_of_calls )
```

Provides any hints such as the type of routine, expected no of calls etc.

`aoclsparse_set_lu_smoother_hint` sets a hint id for analysis and execute phases of the program to analyse and perform ILU factorization and Solution

Parameters

in	*mat*	A sparse matrix and ILU related information inside
in	*trans*	Whether in transposed state or not. Transpose operation is not yet supported.
in	*descr*	Descriptor of the sparse matrix.
in	*expected_no_of_calls*	unused parameter

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	indicates that `m` is invalid, expecting m>=0.
aoclsparse_status_invalid_pointer.	
aoclsparse_status_internal_error	Indicates that an internal error occurred.

3.1.2.4 aoclsparse_set_sm_hint()

```
DLL_PUBLIC aoclsparse_status aoclsparse_set_sm_hint (
    aoclsparse_matrix A,
    aoclsparse_operation trans,
    const aoclsparse_mat_descr descr,
    const aoclsparse_order order,
    const aoclsparse_int dense_matrix_dim,
    const aoclsparse_int expected_no_of_calls )
```

Store user-hints to accelerate the `aoclsparse_?trsm` triangular-solvers.

This function stores user-provided hints related to the structures of the matrices involved in a triangular linear system of equations and its solvers. The hints are for the problem

$$op(A) \cdot X = \alpha \cdot B,$$

where A is a sparse matrix, $op()$ is a linear operator, X and B are dense matrices, while $alpha$ is a scalar. The hints are used in order to perform certain optimizations over the input data that can potentially accelerate the solve operation. The hints include, expected number of calls to the API, matrix layout, dimension of dense right-hand-side matrix, etc.

Parameters

in	*A*	A sparse matrix A.
in	*trans*	Operation to perform on the sparse matrix A, valid options are aoclsparse_operation_none, aoclsparse_operation_transpose, and aoclsparse_operation_conjugate_transpose.
in	*descr*	Descriptor of the sparse matrix A.
in	*order*	Layout of the right-hand-side matrix B, valid options are aoclsparse_order_row and aoclsparse_order_column.
in	*dense_matrix_dim*	number of columns of the dense matrix B.
in	*expected_no_of_calls*	Hint on the potential number of calls to the solver API, e.g., calls to aoclsparse_strsm().

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n`, `nnz`, `ldb` or `ldx` is invalid. Expecting m>0, n>0, m==n, nnz>0, ldb>=n, ldx>=n
aoclsparse_status_invalid_value	Sparse matrix is not square, or `expected_no_of_calls` or `dense_matrix_dim` or `matrix` type are invalid.
aoclsparse_status_invalid_pointer	Pointers to sparse matrix A or dense matrices B or X or descriptor are null
aoclsparse_status_internal_error	Indicates that an internal error occurred.

3.2 aoclsparse_auxiliary.h File Reference

aoclsparse_auxiliary.h provides auxilary functions in aoclsparse

Functions

- DLL_PUBLIC const char ∗ aoclsparse_get_version ()

  Get AOCL-Sparse version.
- DLL_PUBLIC aoclsparse_status aoclsparse_create_mat_descr (aoclsparse_mat_descr ∗descr)

  Create a matrix descriptor.
- DLL_PUBLIC aoclsparse_status aoclsparse_copy_mat_descr (aoclsparse_mat_descr dest, const aoclsparse_mat_descr src)

  Copy a matrix descriptor.
- DLL_PUBLIC aoclsparse_status aoclsparse_destroy_mat_descr (aoclsparse_mat_descr descr)

  Destroy a matrix descriptor.
- DLL_PUBLIC aoclsparse_status aoclsparse_set_mat_index_base (aoclsparse_mat_descr descr, aoclsparse_index_base base)

  Specify the index base of a matrix descriptor.
- DLL_PUBLIC aoclsparse_index_base aoclsparse_get_mat_index_base (const aoclsparse_mat_descr descr)

  Get the index base of a matrix descriptor.
- DLL_PUBLIC aoclsparse_status aoclsparse_set_mat_type (aoclsparse_mat_descr descr, aoclsparse_matrix_type type)

  Specify the matrix type of a matrix descriptor.
- DLL_PUBLIC aoclsparse_matrix_type aoclsparse_get_mat_type (const aoclsparse_mat_descr descr)

  Get the matrix type of a matrix descriptor.
- DLL_PUBLIC aoclsparse_status aoclsparse_set_mat_fill_mode (aoclsparse_mat_descr descr, aoclsparse_fill_mode fill_mode)

  Specify the matrix fill mode of a matrix descriptor.
- DLL_PUBLIC aoclsparse_fill_mode aoclsparse_get_mat_fill_mode (const aoclsparse_mat_descr descr)

  Get the matrix fill mode of a matrix descriptor.
- DLL_PUBLIC aoclsparse_status aoclsparse_set_mat_diag_type (aoclsparse_mat_descr descr, aoclsparse_diag_type diag_type)

  Specify the matrix diagonal type of a matrix descriptor.
- DLL_PUBLIC aoclsparse_diag_type aoclsparse_get_mat_diag_type (const aoclsparse_mat_descr descr)

  Get the matrix diagonal type of a matrix descriptor.

- DLL_PUBLIC aoclsparse_status aoclsparse_create_scsr (aoclsparse_matrix *mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int nnz, aoclsparse_int *row_ptr, aoclsparse_int *col↩_idx, float *val)

  Creates a new `aoclsparse_matrix` *based on CSR (Compressed Sparse Row) format.*
- DLL_PUBLIC aoclsparse_status aoclsparse_create_dcsr (aoclsparse_matrix *mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int nnz, aoclsparse_int *row_ptr, aoclsparse_int *col↩_idx, double *val)

  Creates a new `aoclsparse_matrix` *based on CSR (Compressed Sparse Row) format.*
- DLL_PUBLIC aoclsparse_status aoclsparse_create_ccsr (aoclsparse_matrix *mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int nnz, aoclsparse_int *row_ptr, aoclsparse_int *col↩_idx, aoclsparse_float_complex *val)

  Creates a new `aoclsparse_matrix` *based on CSR (Compressed Sparse Row) format.*
- DLL_PUBLIC aoclsparse_status aoclsparse_create_zcsr (aoclsparse_matrix *mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int nnz, aoclsparse_int *row_ptr, aoclsparse_int *col↩_idx, aoclsparse_double_complex *val)

  Creates a new `aoclsparse_matrix` *based on CSR (Compressed Sparse Row) format.*

- DLL_PUBLIC aoclsparse_status aoclsparse_create_scoo (aoclsparse_matrix *mat, const aoclsparse_index_base base, const aoclsparse_int M, const aoclsparse_int N, const aoclsparse_int nnz, aoclsparse_int *row_ind, aoclsparse_int *col_ind, float *val)

  Creates a new `aoclsparse_matrix` *based on COO (Co-ordinate format).*
- DLL_PUBLIC aoclsparse_status aoclsparse_create_dcoo (aoclsparse_matrix *mat, const aoclsparse_index_base base, const aoclsparse_int M, const aoclsparse_int N, const aoclsparse_int nnz, aoclsparse_int *row_ind, aoclsparse_int *col_ind, double *val)

  Creates a new `aoclsparse_matrix` *based on COO (Co-ordinate format).*
- DLL_PUBLIC aoclsparse_status aoclsparse_create_ccoo (aoclsparse_matrix *mat, const aoclsparse_index_base base, const aoclsparse_int M, const aoclsparse_int N, const aoclsparse_int nnz, aoclsparse_int *row_ind, aoclsparse_int *col_ind, aoclsparse_float_complex *val)

  Creates a new `aoclsparse_matrix` *based on COO (Co-ordinate format).*
- DLL_PUBLIC aoclsparse_status aoclsparse_create_zcoo (aoclsparse_matrix *mat, const aoclsparse_index_base base, const aoclsparse_int M, const aoclsparse_int N, const aoclsparse_int nnz, aoclsparse_int *row_ind, aoclsparse_int *col_ind, aoclsparse_double_complex *val)

  Creates a new `aoclsparse_matrix` *based on COO (Co-ordinate format).*

- DLL_PUBLIC aoclsparse_status aoclsparse_export_scsr (const aoclsparse_matrix mat, aoclsparse_index_base *base, aoclsparse_int *m, aoclsparse_int *n, aoclsparse_int *nnz, aoclsparse_int **row_ptr, aoclsparse_int **col_ind, float **val)

  Export a `CSR` *matrix.*
- DLL_PUBLIC aoclsparse_status aoclsparse_export_dcsr (const aoclsparse_matrix mat, aoclsparse_index_base *base, aoclsparse_int *m, aoclsparse_int *n, aoclsparse_int *nnz, aoclsparse_int **row_ptr, aoclsparse_int **col_ind, double **val)

  Export a `CSR` *matrix.*
- DLL_PUBLIC aoclsparse_status aoclsparse_export_ccsr (const aoclsparse_matrix mat, aoclsparse_index_base *base, aoclsparse_int *m, aoclsparse_int *n, aoclsparse_int *nnz, aoclsparse_int **row_ptr, aoclsparse_int **col_ind, aoclsparse_float_complex **val)

  Export a `CSR` *matrix.*
- DLL_PUBLIC aoclsparse_status aoclsparse_export_zcsr (const aoclsparse_matrix mat, aoclsparse_index_base *base, aoclsparse_int *m, aoclsparse_int *n, aoclsparse_int *nnz, aoclsparse_int **row_ptr, aoclsparse_int **col_ind, aoclsparse_double_complex **val)

Export a `CSR` *matrix.*

- DLL_PUBLIC aoclsparse_status aoclsparse_destroy (aoclsparse_matrix *mat)

    Destroy a sparse matrix structure.

- DLL_PUBLIC aoclsparse_status aoclsparse_create_scsc (aoclsparse_matrix *mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int nnz, aoclsparse_int *col_ptr, aoclsparse_int *row←_idx, float *val)

    Creates a new `aoclsparse_matrix` *based on CSC (Compressed Sparse Column) format.*
- DLL_PUBLIC aoclsparse_status aoclsparse_create_dcsc (aoclsparse_matrix *mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int nnz, aoclsparse_int *col_ptr, aoclsparse_int *row←_idx, double *val)

    Creates a new `aoclsparse_matrix` *based on CSC (Compressed Sparse Column) format.*
- DLL_PUBLIC aoclsparse_status aoclsparse_create_ccsc (aoclsparse_matrix *mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int nnz, aoclsparse_int *col_ptr, aoclsparse_int *row←_idx, aoclsparse_float_complex *val)

    Creates a new `aoclsparse_matrix` *based on CSC (Compressed Sparse Column) format.*
- DLL_PUBLIC aoclsparse_status aoclsparse_create_zcsc (aoclsparse_matrix *mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int nnz, aoclsparse_int *col_ptr, aoclsparse_int *row←_idx, aoclsparse_double_complex *val)

    Creates a new `aoclsparse_matrix` *based on CSC (Compressed Sparse Column) format.*

- DLL_PUBLIC aoclsparse_status aoclsparse_copy (const aoclsparse_matrix src, const aoclsparse_mat_descr descr, aoclsparse_matrix *dest)

    Creates a copy of source `aoclsparse_matrix`*.*

- DLL_PUBLIC aoclsparse_status aoclsparse_order_mat (aoclsparse_matrix mat)

    Performs ordering of index array of the matrix.

- DLL_PUBLIC aoclsparse_status aoclsparse_export_scsc (const aoclsparse_matrix mat, aoclsparse_index_base *base, aoclsparse_int *m, aoclsparse_int *n, aoclsparse_int *nnz, aoclsparse_int **col_ptr, aoclsparse_int **row_ind, float **val)

    Export `CSC` *matrix.*
- DLL_PUBLIC aoclsparse_status aoclsparse_export_dcsc (const aoclsparse_matrix mat, aoclsparse_index_base *base, aoclsparse_int *m, aoclsparse_int *n, aoclsparse_int *nnz, aoclsparse_int **col_ptr, aoclsparse_int **row_ind, double **val)

    Export `CSC` *matrix.*
- DLL_PUBLIC aoclsparse_status aoclsparse_export_ccsc (const aoclsparse_matrix mat, aoclsparse_index_base *base, aoclsparse_int *m, aoclsparse_int *n, aoclsparse_int *nnz, aoclsparse_int **col_ptr, aoclsparse_int **row_ind, aoclsparse_float_complex **val)

    Export `CSC` *matrix.*
- DLL_PUBLIC aoclsparse_status aoclsparse_export_zcsc (const aoclsparse_matrix mat, aoclsparse_index_base *base, aoclsparse_int *m, aoclsparse_int *n, aoclsparse_int *nnz, aoclsparse_int **col_ptr, aoclsparse_int **row_ind, aoclsparse_double_complex **val)

    Export `CSC` *matrix.*

3.2.1 Detailed Description

aoclsparse_auxiliary.h provides auxilary functions in aoclsparse

3.2.2 Function Documentation

3.2.2.1 aoclsparse_get_version() `DLL_PUBLIC const char* aoclsparse_get_version ( )`

Get AOCL-Sparse version.

`aoclsparse_get_version` gets the aoclsparse library version number. in the format "AOCL-Sparse <major>.<minor>.<patch>"

Parameters

out	*version*	the version string of the aoclsparse library.

3.2.2.2 aoclsparse_create_mat_descr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_mat_descr (
        aoclsparse_mat_descr * descr )
```

Create a matrix descriptor.

`aoclsparse_create_mat_descr` creates a matrix descriptor. It initializes aoclsparse_matrix_type to aoclsparse_matrix_type_general and aoclsparse_index_base to aoclsparse_index_base_zero. It should be destroyed at the end using aoclsparse_destroy_mat_descr().

Parameters

out	*descr*	the pointer to the matrix descriptor.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`descr` pointer is invalid.

3.2.2.3 aoclsparse_copy_mat_descr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_copy_mat_descr (
        aoclsparse_mat_descr dest,
        const aoclsparse_mat_descr src )
```

Copy a matrix descriptor.

`aoclsparse_copy_mat_descr` copies a matrix descriptor. Both, source and destination matrix descriptors must be initialized prior to calling `aoclsparse_copy_mat_descr`.

Parameters

out	*dest*	the pointer to the destination matrix descriptor.
in	*src*	the pointer to the source matrix descriptor.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`src` or `dest` pointer is invalid.

3.2.2.4 aoclsparse_destroy_mat_descr() `DLL_PUBLIC aoclsparse_status aoclsparse_destroy_mat_descr ( aoclsparse_mat_descr descr )`

Destroy a matrix descriptor.

`aoclsparse_destroy_mat_descr` destroys a matrix descriptor and releases all resources used by the descriptor.

Parameters

in	*descr*	the matrix descriptor.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`descr` is invalid.

3.2.2.5 aoclsparse_set_mat_index_base() `DLL_PUBLIC aoclsparse_status aoclsparse_set_mat_index_base ( aoclsparse_mat_descr descr, aoclsparse_index_base base )`

Specify the index base of a matrix descriptor.

`aoclsparse_set_mat_index_base` sets the index base of a matrix descriptor. Valid options are aoclsparse_index_base_zero or aoclsparse_index_base_one.

Parameters

in,out	*descr*	the matrix descriptor.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`descr` pointer is invalid.
aoclsparse_status_invalid_value	`base` is invalid.

3.2.2.6 aoclsparse_get_mat_index_base() `DLL_PUBLIC aoclsparse_index_base aoclsparse_get_mat_index_base ( const aoclsparse_mat_descr descr )`

Get the index base of a matrix descriptor.

`aoclsparse_get_mat_index_base` returns the index base of a matrix descriptor.

Parameters

in	*descr*	the matrix descriptor.

Returns

aoclsparse_index_base_zero or aoclsparse_index_base_one.

3.2.2.7 aoclsparse_set_mat_type() `DLL_PUBLIC aoclsparse_status aoclsparse_set_mat_type ( aoclsparse_mat_descr descr, aoclsparse_matrix_type type )`

Specify the matrix type of a matrix descriptor.

`aoclsparse_set_mat_type` sets the matrix type of a matrix descriptor. Valid matrix types are aoclsparse_matrix_type_general, aoclsparse_matrix_type_symmetric, aoclsparse_matrix_type_hermitian or aoclsparse_matrix_type_triangular.

Parameters

in,out	*descr*	the matrix descriptor.
in	*type*	aoclsparse_matrix_type_general, aoclsparse_matrix_type_symmetric, aoclsparse_matrix_type_hermitian or aoclsparse_matrix_type_triangular.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`descr` pointer is invalid.
aoclsparse_status_invalid_value	`type` is invalid.

3.2.2.8 aoclsparse_get_mat_type() `DLL_PUBLIC aoclsparse_matrix_type aoclsparse_get_mat_type ( const aoclsparse_mat_descr descr )`

Get the matrix type of a matrix descriptor.

`aoclsparse_get_mat_type` returns the matrix type of a matrix descriptor.

Parameters

in	*descr*	the matrix descriptor.

Returns

aoclsparse_matrix_type_general, aoclsparse_matrix_type_symmetric, aoclsparse_matrix_type_hermitian or aoclsparse_matrix_type_triangular.

3.2.2.9 aoclsparse_set_mat_fill_mode() `DLL_PUBLIC aoclsparse_status aoclsparse_set_mat_fill_mode ( aoclsparse_mat_descr descr, aoclsparse_fill_mode fill_mode )`

Specify the matrix fill mode of a matrix descriptor.

`aoclsparse_set_mat_fill_mode` sets the matrix fill mode of a matrix descriptor. Valid fill modes are aoclsparse_fill_mode_lower or aoclsparse_fill_mode_upper.

Parameters

in,out	*descr*	the matrix descriptor.
in	*fill_mode*	aoclsparse_fill_mode_lower or aoclsparse_fill_mode_upper.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`descr` pointer is invalid.
aoclsparse_status_invalid_value	`fill_mode` is invalid.

3.2.2.10 aoclsparse_get_mat_fill_mode() `DLL_PUBLIC aoclsparse_fill_mode aoclsparse_get_mat_fill_mode ( const aoclsparse_mat_descr descr )`

Get the matrix fill mode of a matrix descriptor.

`aoclsparse_get_mat_fill_mode` returns the matrix fill mode of a matrix descriptor.

Parameters

in	*descr*	the matrix descriptor.

Returns

aoclsparse_fill_mode_lower or aoclsparse_fill_mode_upper.

3.2.2.11 aoclsparse_set_mat_diag_type() `DLL_PUBLIC aoclsparse_status aoclsparse_set_mat_diag_type ( aoclsparse_mat_descr descr, aoclsparse_diag_type diag_type )`

Specify the matrix diagonal type of a matrix descriptor.

`aoclsparse_set_mat_diag_type` sets the matrix diagonal type of a matrix descriptor. Valid diagonal types are aoclsparse_diag_type_unit, aoclsparse_diag_type_non_unit or aoclsparse_diag_type_zero.

Parameters

in,out	*descr*	the matrix descriptor.
in	*diag_type*	aoclsparse_diag_type_unit or aoclsparse_diag_type_non_unit or aoclsparse_diag_type_zero.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`descr` pointer is invalid.
aoclsparse_status_invalid_value	`diag_type` is invalid.

3.2.2.12 aoclsparse_get_mat_diag_type() `DLL_PUBLIC aoclsparse_diag_type aoclsparse_get_mat_diag_type ( const aoclsparse_mat_descr descr )`

Get the matrix diagonal type of a matrix descriptor.

`aoclsparse_get_mat_diag_type` returns the matrix diagonal type of a matrix descriptor.

Parameters

in	*descr*	the matrix descriptor.

Returns

aoclsparse_diag_type_unit or aoclsparse_diag_type_non_unit or aoclsparse_diag_type_zero.

3.2.2.13 aoclsparse_create_scsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_scsr (
        aoclsparse_matrix * mat,
        aoclsparse_index_base base,
        aoclsparse_int M,
        aoclsparse_int N,
        aoclsparse_int nnz,
        aoclsparse_int * row_ptr,
        aoclsparse_int * col_idx,
        float * val )
```

Creates a new `aoclsparse_matrix` based on CSR (Compressed Sparse Row) format.

`aoclsparse_create_(s/d/c/z)csr` creates `aoclsparse_matrix` and initializes it with input parameters passed. The input arrays are left unchanged except for the call to `aoclsparse_order_mat`, which performs ordering of column indices of the matrix. To avoid any changes to the input data, `aoclsparse_copy` can be used. To convert any other format to CSR, `aoclsparse_convert` can be used. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

out	*mat*	the pointer to the CSR sparse matrix allocated in the API.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
in	*M*	number of rows of the sparse CSR matrix.
in	*N*	number of columns of the sparse CSR matrix.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*col_idx*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
in	*val*	array of `nnz` elements of the sparse CSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	at least one of `row_ptr`, `col_idx` or `val` pointer is NULL.
aoclsparse_status_invalid_size	at least one of `M`, `N` or `nnz` has a negative value.
aoclsparse_status_invalid_index_value	any `col_idx` value is not within N.
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.14 aoclsparse_create_dcsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_dcsr (
        aoclsparse_matrix * mat,
        aoclsparse_index_base base,
        aoclsparse_int M,
        aoclsparse_int N,
        aoclsparse_int nnz,
        aoclsparse_int * row_ptr,
        aoclsparse_int * col_idx,
        double * val )
```

Creates a new `aoclsparse_matrix` based on CSR (Compressed Sparse Row) format.

`aoclsparse_create_(s/d/c/z)csr` creates `aoclsparse_matrix` and initializes it with input parameters passed. The input arrays are left unchanged except for the call to `aoclsparse_order_mat`, which performs ordering of column indices of the matrix. To avoid any changes to the input data, `aoclsparse_copy` can be used. To convert any other format to CSR, `aoclsparse_convert` can be used. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

out	*mat*	the pointer to the CSR sparse matrix allocated in the API.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
in	*M*	number of rows of the sparse CSR matrix.
in	*N*	number of columns of the sparse CSR matrix.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*col_idx*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
in	*val*	array of `nnz` elements of the sparse CSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	at least one of `row_ptr`, `col_idx` or `val` pointer is NULL.
aoclsparse_status_invalid_size	at least one of `M`, `N` or `nnz` has a negative value.
aoclsparse_status_invalid_index_value	any `col_idx` value is not within N.
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.15 aoclsparse_create_ccsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_ccsr (
        aoclsparse_matrix * mat,
        aoclsparse_index_base base,
        aoclsparse_int M,
        aoclsparse_int N,
        aoclsparse_int nnz,
        aoclsparse_int * row_ptr,
        aoclsparse_int * col_idx,
        aoclsparse_float_complex * val )
```

Creates a new `aoclsparse_matrix` based on CSR (Compressed Sparse Row) format.

`aoclsparse_create_(s/d/c/z)csr` creates `aoclsparse_matrix` and initializes it with input parameters passed. The input arrays are left unchanged except for the call to `aoclsparse_order_mat`, which performs ordering of column indices of the matrix. To avoid any changes to the input data, `aoclsparse_copy` can be used. To convert any other format to CSR, `aoclsparse_convert` can be used. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

out	*mat*	the pointer to the CSR sparse matrix allocated in the API.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
in	*M*	number of rows of the sparse CSR matrix.

Parameters

in	*N*	number of columns of the sparse CSR matrix.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*col_idx*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
in	*val*	array of `nnz` elements of the sparse CSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	at least one of `row_ptr`, `col_idx` or `val` pointer is NULL.
aoclsparse_status_invalid_size	at least one of `M`, `N` or `nnz` has a negative value.
aoclsparse_status_invalid_index_value	any `col_idx` value is not within N.
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.16 aoclsparse_create_zcsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_zcsr (
    aoclsparse_matrix * mat,
    aoclsparse_index_base base,
    aoclsparse_int M,
    aoclsparse_int N,
    aoclsparse_int nnz,
    aoclsparse_int * row_ptr,
    aoclsparse_int * col_idx,
    aoclsparse_double_complex * val )
```

Creates a new `aoclsparse_matrix` based on CSR (Compressed Sparse Row) format.

`aoclsparse_create_(s/d/c/z)csr` creates `aoclsparse_matrix` and initializes it with input parameters passed. The input arrays are left unchanged except for the call to `aoclsparse_order_mat`, which performs ordering of column indices of the matrix. To avoid any changes to the input data, `aoclsparse_copy` can be used. To convert any other format to CSR, `aoclsparse_convert` can be used. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

out	*mat*	the pointer to the CSR sparse matrix allocated in the API.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
in	*M*	number of rows of the sparse CSR matrix.
in	*N*	number of columns of the sparse CSR matrix.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*col_idx*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
in	*val*	array of `nnz` elements of the sparse CSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.

Return values

aoclsparse_status_invalid_pointer	at least one of `row_ptr`, `col_idx` or `val` pointer is NULL.
aoclsparse_status_invalid_size	at least one of `M`, `N` or `nnz` has a negative value.
aoclsparse_status_invalid_index_value	any `col_idx` value is not within N.
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.17 aoclsparse_create_scoo()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_scoo (
        aoclsparse_matrix * mat,
        const aoclsparse_index_base base,
        const aoclsparse_int M,
        const aoclsparse_int N,
        const aoclsparse_int nnz,
        aoclsparse_int * row_ind,
        aoclsparse_int * col_ind,
        float * val )
```

Creates a new `aoclsparse_matrix` based on COO (Co-ordinate format).

`aoclsparse_create_(s/d/c/z)coo` creates `aoclsparse_matrix` and initializes it with input parameters passed. Array data must not be modified by the user while matrix is alive as the pointers are copied, not the data. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

`in,out`	*mat*	the pointer to the COO sparse matrix.
`in`	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one depending on whether the index first element starts from 0 or 1.
`in`	*M*	total number of rows of the sparse COO matrix.
`in`	*N*	total number of columns of the sparse COO matrix.
`in`	*nnz*	number of non-zero entries of the sparse COO matrix.
`in`	*row_ind*	array of `nnz` elements that point to the row of the element in co-ordinate Format.
`in`	*col_ind*	array of `nnz` elements that point to the column of the element in co-ordinate Format.
`in`	*val*	array of `nnz` elements of the sparse COO matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	pointer given to API is invalid or nullptr.
aoclsparse_status_invalid_size	`coo` dimension of matrix or non-zero elements is invalid.
aoclsparse_status_invalid_index_value	index given for `coo` is out of matrix bounds depending on base given
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.18 aoclsparse_create_dcoo()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_dcoo (
        aoclsparse_matrix * mat,
```

```
const aoclsparse_index_base base,
const aoclsparse_int M,
const aoclsparse_int N,
const aoclsparse_int nnz,
aoclsparse_int * row_ind,
aoclsparse_int * col_ind,
double * val )
```

Creates a new `aoclsparse_matrix` based on COO (Co-ordinate format).

`aoclsparse_create_(s/d/c/z)coo` creates `aoclsparse_matrix` and initializes it with input parameters passed. Array data must not be modified by the user while matrix is alive as the pointers are copied, not the data. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

in,out	*mat*	the pointer to the COO sparse matrix.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one depending on whether the index first element starts from 0 or 1.
in	*M*	total number of rows of the sparse COO matrix.
in	*N*	total number of columns of the sparse COO matrix.
in	*nnz*	number of non-zero entries of the sparse COO matrix.
in	*row_ind*	array of `nnz` elements that point to the row of the element in co-ordinate Format.
in	*col_ind*	array of `nnz` elements that point to the column of the element in co-ordinate Format.
in	*val*	array of `nnz` elements of the sparse COO matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	pointer given to API is invalid or nullptr.
aoclsparse_status_invalid_size	`coo` dimension of matrix or non-zero elements is invalid.
aoclsparse_status_invalid_index_value	index given for `coo` is out of matrix bounds depending on base given
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.19 aoclsparse_create_ccoo() `DLL_PUBLIC aoclsparse_status aoclsparse_create_ccoo (`

```
aoclsparse_matrix * mat,
const aoclsparse_index_base base,
const aoclsparse_int M,
const aoclsparse_int N,
const aoclsparse_int nnz,
aoclsparse_int * row_ind,
aoclsparse_int * col_ind,
aoclsparse_float_complex * val )
```

Creates a new `aoclsparse_matrix` based on COO (Co-ordinate format).

`aoclsparse_create_(s/d/c/z)coo` creates `aoclsparse_matrix` and initializes it with input parameters passed. Array data must not be modified by the user while matrix is alive as the pointers are copied, not the data. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

in,out	*mat*	the pointer to the COO sparse matrix.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one depending on whether the index first element starts from 0 or 1.
in	*M*	total number of rows of the sparse COO matrix.
in	*N*	total number of columns of the sparse COO matrix.
in	*nnz*	number of non-zero entries of the sparse COO matrix.
in	*row_ind*	array of `nnz` elements that point to the row of the element in co-ordinate Format.
in	*col_ind*	array of `nnz` elements that point to the column of the element in co-ordinate Format.
in	*val*	array of `nnz` elements of the sparse COO matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	pointer given to API is invalid or nullptr.
aoclsparse_status_invalid_size	`coo` dimension of matrix or non-zero elements is invalid.
aoclsparse_status_invalid_index_value	index given for `coo` is out of matrix bounds depending on base given
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.20 aoclsparse_create_zcoo()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_zcoo (
        aoclsparse_matrix * mat,
        const aoclsparse_index_base base,
        const aoclsparse_int M,
        const aoclsparse_int N,
        const aoclsparse_int nnz,
        aoclsparse_int * row_ind,
        aoclsparse_int * col_ind,
        aoclsparse_double_complex * val )
```

Creates a new `aoclsparse_matrix` based on COO (Co-ordinate format).

`aoclsparse_create_(s/d/c/z)coo` creates `aoclsparse_matrix` and initializes it with input parameters passed. Array data must not be modified by the user while matrix is alive as the pointers are copied, not the data. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

in,out	*mat*	the pointer to the COO sparse matrix.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one depending on whether the index first element starts from 0 or 1.
in	*M*	total number of rows of the sparse COO matrix.
in	*N*	total number of columns of the sparse COO matrix.
in	*nnz*	number of non-zero entries of the sparse COO matrix.
in	*row_ind*	array of `nnz` elements that point to the row of the element in co-ordinate Format.
in	*col_ind*	array of `nnz` elements that point to the column of the element in co-ordinate Format.
in	*val*	array of `nnz` elements of the sparse COO matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	pointer given to API is invalid or nullptr.
aoclsparse_status_invalid_size	`coo` dimension of matrix or non-zero elements is invalid.
aoclsparse_status_invalid_index_value	index given for `coo` is out of matrix bounds depending on base given
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.21 aoclsparse_export_scsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_export_scsr (
        const aoclsparse_matrix mat,
        aoclsparse_index_base * base,
        aoclsparse_int * m,
        aoclsparse_int * n,
        aoclsparse_int * nnz,
        aoclsparse_int ** row_ptr,
        aoclsparse_int ** col_ind,
        float ** val )
```

Export a `CSR` matrix.

`aoclsparse_export_(s/d/c/z)csr` exposes the components defining the `CSR` matrix in `mat` structure by copying out the data pointers. No additional memory is allocated. User should not modify the arrays and once `aoclsparse_destroy()` is called to free `mat`, these arrays will become inaccessible. If the matrix is not in `CSR` format, an error is obtained. aoclsparse_convert_csr can be used to convert non-CSR format to `CSR` format.

Parameters

`in`	*mat*	the pointer to the CSR sparse matrix.
`out`	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
`out`	*m*	number of rows of the sparse CSR matrix.
`out`	*n*	number of columns of the sparse CSR matrix.
`out`	*nnz*	number of non-zero entries of the sparse CSR matrix.
`out`	*row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
`out`	*col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
`out`	*val*	array of `nnz` elements of the sparse CSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`mat` or any of the output arguments are NULL.
aoclsparse_status_invalid_value	`mat` is not in CSR format.
aoclsparse_status_wrong_type	data type of `mat` does not match the function.

3.2.2.22 aoclsparse_export_dcsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_export_dcsr (
        const aoclsparse_matrix mat,
```

```
aoclsparse_index_base * base,
aoclsparse_int * m,
aoclsparse_int * n,
aoclsparse_int * nnz,
aoclsparse_int ** row_ptr,
aoclsparse_int ** col_ind,
double ** val )
```

Export a CSR matrix.

aoclsparse_export_(s/d/c/z)csr exposes the components defining the CSR matrix in mat structure by copying out the data pointers. No additional memory is allocated. User should not modify the arrays and once aoclsparse_destroy() is called to free mat, these arrays will become inaccessible. If the matrix is not in CSR format, an error is obtained. aoclsparse_convert_csr can be used to convert non-CSR format to CSR format.

Parameters

in	*mat*	the pointer to the CSR sparse matrix.
out	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
out	*m*	number of rows of the sparse CSR matrix.
out	*n*	number of columns of the sparse CSR matrix.
out	*nnz*	number of non-zero entries of the sparse CSR matrix.
out	*row_ptr*	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
out	*col_ind*	array of nnz elements containing the column indices of the sparse CSR matrix.
out	*val*	array of nnz elements of the sparse CSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	mat or any of the output arguments are NULL.
aoclsparse_status_invalid_value	mat is not in CSR format.
aoclsparse_status_wrong_type	data type of mat does not match the function.

3.2.2.23 aoclsparse_export_ccsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_export_ccsr (
const aoclsparse_matrix mat,
aoclsparse_index_base * base,
aoclsparse_int * m,
aoclsparse_int * n,
aoclsparse_int * nnz,
aoclsparse_int ** row_ptr,
aoclsparse_int ** col_ind,
aoclsparse_float_complex ** val )
```

Export a CSR matrix.

aoclsparse_export_(s/d/c/z)csr exposes the components defining the CSR matrix in mat structure by copying out the data pointers. No additional memory is allocated. User should not modify the arrays and once aoclsparse_destroy() is called to free mat, these arrays will become inaccessible. If the matrix is not in CSR format, an error is obtained. aoclsparse_convert_csr can be used to convert non-CSR format to CSR format.

Parameters

in	*mat*	the pointer to the CSR sparse matrix.
out	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
out	*m*	number of rows of the sparse CSR matrix.
out	*n*	number of columns of the sparse CSR matrix.
out	*nnz*	number of non-zero entries of the sparse CSR matrix.
out	*row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
out	*col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
out	*val*	array of `nnz` elements of the sparse CSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`mat` or any of the output arguments are NULL.
aoclsparse_status_invalid_value	`mat` is not in CSR format.
aoclsparse_status_wrong_type	data type of `mat` does not match the function.

3.2.2.24 aoclsparse_export_zcsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_export_zcsr (
        const aoclsparse_matrix mat,
        aoclsparse_index_base * base,
        aoclsparse_int * m,
        aoclsparse_int * n,
        aoclsparse_int * nnz,
        aoclsparse_int ** row_ptr,
        aoclsparse_int ** col_ind,
        aoclsparse_double_complex ** val )
```

Export a `CSR` matrix.

`aoclsparse_export_(s/d/c/z)csr` exposes the components defining the `CSR` matrix in `mat` structure by copying out the data pointers. No additional memory is allocated. User should not modify the arrays and once `aoclsparse_destroy()` is called to free `mat`, these arrays will become inaccessible. If the matrix is not in `CSR` format, an error is obtained. aoclsparse_convert_csr can be used to convert non-CSR format to `CSR` format.

Parameters

in	*mat*	the pointer to the CSR sparse matrix.
out	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
out	*m*	number of rows of the sparse CSR matrix.
out	*n*	number of columns of the sparse CSR matrix.
out	*nnz*	number of non-zero entries of the sparse CSR matrix.
out	*row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
out	*col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
out	*val*	array of `nnz` elements of the sparse CSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`mat` or any of the output arguments are NULL.
aoclsparse_status_invalid_value	`mat` is not in CSR format.
aoclsparse_status_wrong_type	data type of `mat` does not match the function.

3.2.2.25 aoclsparse_destroy()

```
DLL_PUBLIC aoclsparse_status aoclsparse_destroy (
    aoclsparse_matrix * mat )
```

Destroy a sparse matrix structure.

`aoclsparse_destroy` destroys a structure that holds the matrix

Parameters

`in`	*mat*	the pointer to the sparse matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`matrix` structure pointer is invalid.

3.2.2.26 aoclsparse_create_scsc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_scsc (
    aoclsparse_matrix * mat,
    aoclsparse_index_base base,
    aoclsparse_int M,
    aoclsparse_int N,
    aoclsparse_int nnz,
    aoclsparse_int * col_ptr,
    aoclsparse_int * row_idx,
    float * val )
```

Creates a new `aoclsparse_matrix` based on CSC (Compressed Sparse Column) format.

`aoclsparse_create_(s/d/c/z)csc` creates `aoclsparse_matrix` and initializes it with input parameters passed. The input arrays are left unchanged except for the call to `aoclsparse_order_mat`, which performs ordering of row indices of the matrix. To avoid any changes to the input data, `aoclsparse_copy` can be used. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

`in,out`	*mat*	the pointer to the CSC sparse matrix allocated in the API.
`in`	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
`in`	*M*	number of rows of the sparse CSC matrix.
`in`	*N*	number of columns of the sparse CSC matrix.

Parameters

in	*nnz*	number of non-zero entries of the sparse CSC matrix.
in	*col_ptr*	array of `n+1` elements that points to the start of every column in row_idx array of the sparse CSC matrix.
in	*row_idx*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
in	*val*	array of `nnz` elements of the sparse CSC matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`col_ptr`, `row_idx` or `val` pointer is NULL.
aoclsparse_status_invalid_size	`M`, `N` or `nnz` are negative values.
aoclsparse_status_invalid_index_value	any `row_idx` value is not within M.
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.27 aoclsparse_create_dcsc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_dcsc (
    aoclsparse_matrix * mat,
    aoclsparse_index_base base,
    aoclsparse_int M,
    aoclsparse_int N,
    aoclsparse_int nnz,
    aoclsparse_int * col_ptr,
    aoclsparse_int * row_idx,
    double * val )
```

Creates a new `aoclsparse_matrix` based on CSC (Compressed Sparse Column) format.

`aoclsparse_create_(s/d/c/z)csc` creates `aoclsparse_matrix` and initializes it with input parameters passed. The input arrays are left unchanged except for the call to `aoclsparse_order_mat`, which performs ordering of row indices of the matrix. To avoid any changes to the input data, `aoclsparse_copy` can be used. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

in,out	*mat*	the pointer to the CSC sparse matrix allocated in the API.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
in	*M*	number of rows of the sparse CSC matrix.
in	*N*	number of columns of the sparse CSC matrix.
in	*nnz*	number of non-zero entries of the sparse CSC matrix.
in	*col_ptr*	array of `n+1` elements that points to the start of every column in row_idx array of the sparse CSC matrix.
in	*row_idx*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
in	*val*	array of `nnz` elements of the sparse CSC matrix.

Return values

aoclsparse_status_success	the operation completed successfully.

Return values

aoclsparse_status_invalid_pointer	`col_ptr`, `row_idx` or `val` pointer is NULL.
aoclsparse_status_invalid_size	`M`, `N` or `nnz` are negative values.
aoclsparse_status_invalid_index_value	any `row_idx` value is not within M.
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.28 aoclsparse_create_ccsc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_ccsc (
        aoclsparse_matrix * mat,
        aoclsparse_index_base base,
        aoclsparse_int M,
        aoclsparse_int N,
        aoclsparse_int nnz,
        aoclsparse_int * col_ptr,
        aoclsparse_int * row_idx,
        aoclsparse_float_complex * val )
```

Creates a new `aoclsparse_matrix` based on CSC (Compressed Sparse Column) format.

`aoclsparse_create_(s/d/c/z)csc` creates `aoclsparse_matrix` and initializes it with input parameters passed. The input arrays are left unchanged except for the call to `aoclsparse_order_mat`, which performs ordering of row indices of the matrix. To avoid any changes to the input data, `aoclsparse_copy` can be used. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

`in,out`	*mat*	the pointer to the CSC sparse matrix allocated in the API.
`in`	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
`in`	*M*	number of rows of the sparse CSC matrix.
`in`	*N*	number of columns of the sparse CSC matrix.
`in`	*nnz*	number of non-zero entries of the sparse CSC matrix.
`in`	*col_ptr*	array of `n+1` elements that points to the start of every column in row_idx array of the sparse CSC matrix.
`in`	*row_idx*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
`in`	*val*	array of `nnz` elements of the sparse CSC matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`col_ptr`, `row_idx` or `val` pointer is NULL.
aoclsparse_status_invalid_size	`M`, `N` or `nnz` are negative values.
aoclsparse_status_invalid_index_value	any `row_idx` value is not within M.
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.29 aoclsparse_create_zcsc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_create_zcsc (
        aoclsparse_matrix * mat,
```

```
aoclsparse_index_base base,
aoclsparse_int M,
aoclsparse_int N,
aoclsparse_int nnz,
aoclsparse_int * col_ptr,
aoclsparse_int * row_idx,
aoclsparse_double_complex * val )
```

Creates a new aoclsparse_matrix based on CSC (Compressed Sparse Column) format.

aoclsparse_create_(s/d/c/z)csc creates aoclsparse_matrix and initializes it with input parameters passed. The input arrays are left unchanged except for the call to aoclsparse_order_mat, which performs ordering of row indices of the matrix. To avoid any changes to the input data, aoclsparse_copy can be used. Matrix should be destroyed at the end using aoclsparse_destroy.

Parameters

in,out	*mat*	the pointer to the CSC sparse matrix allocated in the API.
in	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
in	*M*	number of rows of the sparse CSC matrix.
in	*N*	number of columns of the sparse CSC matrix.
in	*nnz*	number of non-zero entries of the sparse CSC matrix.
in	*col_ptr*	array of n+1 elements that points to the start of every column in row_idx array of the sparse CSC matrix.
in	*row_idx*	array of nnz elements containing the row indices of the sparse CSC matrix.
in	*val*	array of nnz elements of the sparse CSC matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	col_ptr, row_idx or val pointer is NULL.
aoclsparse_status_invalid_size	M, N or nnz are negative values.
aoclsparse_status_invalid_index_value	any row_idx value is not within M.
aoclsparse_status_memory_error	memory allocation for matrix failed.

3.2.2.30 aoclsparse_copy()

```
DLL_PUBLIC aoclsparse_status aoclsparse_copy (
const aoclsparse_matrix src,
const aoclsparse_mat_descr descr,
aoclsparse_matrix * dest )
```

Creates a copy of source aoclsparse_matrix.

aoclsparse_copy creates a deep copy of source aoclsparse_matrix (hints and optimized data are not copied). Matrix should be destroyed using aoclsparse_destroy(). aoclsparse_convert_csr() can also be used to create a copy of the source matrix while converting it in CSR format.

Parameters

in	*src*	the source aoclsparse_matrix to copy.
in	*descr*	the source matrix descriptor, this argument is reserved for future releases and it will not be referenced.
out	*dest*	pointer to the newly allocated copied aoclsparse_matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`src`, `dest` or internal pointers are NULL or dest points to src.
aoclsparse_status_memory_error	memory allocation for matrix failed.
aoclsparse_status_invalid_value	`src` matrix type is invalid.
aoclsparse_status_wrong_type	`src` matrix data type is invalid.

3.2.2.31 aoclsparse_order_mat()

```
DLL_PUBLIC aoclsparse_status aoclsparse_order_mat (
        aoclsparse_matrix mat )
```

Performs ordering of index array of the matrix.

`aoclsparse_order` orders column indices within a row for matrix in CSR format and row indices within a column for CSC format. It also adjusts value array accordingly. Ordering is implemented only for CSR and CSC format. `aoclsparse_copy` can be used to get exact copy of data `aoclsparse_convert` can be used to convert any format to CSR. Matrix should be destroyed at the end using `aoclsparse_destroy`.

Parameters

in,out	*mat*	pointer to matrix in either CSR or CSC format

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`mat` pointer is NULL.
aoclsparse_status_memory_error	internal memory allocation failed.
aoclsparse_status_not_implemented	matrix is not in CSR format.

3.2.2.32 aoclsparse_export_scsc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_export_scsc (
        const aoclsparse_matrix mat,
        aoclsparse_index_base * base,
        aoclsparse_int * m,
        aoclsparse_int * n,
        aoclsparse_int * nnz,
        aoclsparse_int ** col_ptr,
        aoclsparse_int ** row_ind,
        float ** val )
```

Export `CSC` matrix.

`aoclsparse_export_(s/d/c/z)csc` exposes the components defining the `CSC` matrix in `mat` structure by copying out the data pointers. No additional memory is allocated. User should not modify the arrays and once `aoclsparse_destroy()` is called to free `mat`, these arrays will become inaccessible. If the matrix is not in `CSC` format, an error is obtained.

Parameters

in	*mat*	the pointer to the CSC sparse matrix.
out	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
out	*m*	number of rows of the sparse CSC matrix.
out	*n*	number of columns of the sparse CSC matrix.
out	*nnz*	number of non-zero entries of the sparse CSC matrix.
out	*col_ptr*	array of `n+1` elements that point to the start of every col of the sparse CSC matrix.
out	*row_ind*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
out	*val*	array of `nnz` elements of the sparse CSC matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`mat` or any of the output arguments are NULL.
aoclsparse_status_invalid_value	`mat` is not in CSC format.
aoclsparse_status_wrong_type	data type of `mat` does not match the function.

3.2.2.33 aoclsparse_export_dcsc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_export_dcsc (
        const aoclsparse_matrix mat,
        aoclsparse_index_base * base,
        aoclsparse_int * m,
        aoclsparse_int * n,
        aoclsparse_int * nnz,
        aoclsparse_int ** col_ptr,
        aoclsparse_int ** row_ind,
        double ** val )
```

Export `CSC` matrix.

`aoclsparse_export_(s/d/c/z)csc` exposes the components defining the `CSC` matrix in `mat` structure by copying out the data pointers. No additional memory is allocated. User should not modify the arrays and once `aoclsparse_destroy()` is called to free `mat`, these arrays will become inaccessible. If the matrix is not in `CSC` format, an error is obtained.

Parameters

in	*mat*	the pointer to the CSC sparse matrix.
out	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
out	*m*	number of rows of the sparse CSC matrix.
out	*n*	number of columns of the sparse CSC matrix.
out	*nnz*	number of non-zero entries of the sparse CSC matrix.
out	*col_ptr*	array of `n+1` elements that point to the start of every col of the sparse CSC matrix.
out	*row_ind*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
out	*val*	array of `nnz` elements of the sparse CSC matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`mat` or any of the output arguments are NULL.
aoclsparse_status_invalid_value	`mat` is not in CSC format.
aoclsparse_status_wrong_type	data type of `mat` does not match the function.

3.2.2.34 aoclsparse_export_ccsc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_export_ccsc (
        const aoclsparse_matrix mat,
        aoclsparse_index_base * base,
        aoclsparse_int * m,
        aoclsparse_int * n,
        aoclsparse_int * nnz,
        aoclsparse_int ** col_ptr,
        aoclsparse_int ** row_ind,
        aoclsparse_float_complex ** val )
```

Export `CSC` matrix.

`aoclsparse_export_(s/d/c/z)csc` exposes the components defining the `CSC` matrix in `mat` structure by copying out the data pointers. No additional memory is allocated. User should not modify the arrays and once `aoclsparse_destroy()` is called to free `mat`, these arrays will become inaccessible. If the matrix is not in `CSC` format, an error is obtained.

Parameters

`in`	*mat*	the pointer to the CSC sparse matrix.
`out`	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
`out`	*m*	number of rows of the sparse CSC matrix.
`out`	*n*	number of columns of the sparse CSC matrix.
`out`	*nnz*	number of non-zero entries of the sparse CSC matrix.
`out`	*col_ptr*	array of `n+1` elements that point to the start of every col of the sparse CSC matrix.
`out`	*row_ind*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
`out`	*val*	array of `nnz` elements of the sparse CSC matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`mat` or any of the output arguments are NULL.
aoclsparse_status_invalid_value	`mat` is not in CSC format.
aoclsparse_status_wrong_type	data type of `mat` does not match the function.

3.2.2.35 aoclsparse_export_zcsc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_export_zcsc (
        const aoclsparse_matrix mat,
        aoclsparse_index_base * base,
```

```
aoclsparse_int * m,
aoclsparse_int * n,
aoclsparse_int * nnz,
aoclsparse_int ** col_ptr,
aoclsparse_int ** row_ind,
aoclsparse_double_complex ** val )
```

Export `CSC` matrix.

`aoclsparse_export_(s/d/c/z)csc` exposes the components defining the `CSC` matrix in `mat` structure by copying out the data pointers. No additional memory is allocated. User should not modify the arrays and once `aoclsparse_destroy()` is called to free `mat`, these arrays will become inaccessible. If the matrix is not in `CSC` format, an error is obtained.

Parameters

in	*mat*	the pointer to the CSC sparse matrix.
out	*base*	aoclsparse_index_base_zero or aoclsparse_index_base_one.
out	*m*	number of rows of the sparse CSC matrix.
out	*n*	number of columns of the sparse CSC matrix.
out	*nnz*	number of non-zero entries of the sparse CSC matrix.
out	*col_ptr*	array of `n+1` elements that point to the start of every col of the sparse CSC matrix.
out	*row_ind*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
out	*val*	array of `nnz` elements of the sparse CSC matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`mat` or any of the output arguments are NULL.
aoclsparse_status_invalid_value	`mat` is not in CSC format.
aoclsparse_status_wrong_type	data type of `mat` does not match the function.

3.3 aoclsparse_convert.h File Reference

aoclsparse_convert.h provides sparse format conversion subprograms

Functions

- DLL_PUBLIC aoclsparse_status aoclsparse_csr2ell_width (aoclsparse_int m, aoclsparse_int nnz, const aoclsparse_int ∗csr_row_ptr, aoclsparse_int ∗ell_width)

  Convert a sparse CSR matrix into a sparse ELL matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_csr2dia_ndiag (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, aoclsparse_int nnz, const aoclsparse_int ∗csr_row_ptr, const aoclsparse_int ∗csr_col_ind, aoclsparse_int ∗dia_num_diag)

  Convert a sparse CSR matrix into a sparse DIA matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_csr2bsr_nnz (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, const aoclsparse_int ∗csr_row_ptr, const aoclsparse_int ∗csr_col_ind, aoclsparse_int block_dim, aoclsparse_int ∗bsr_row_ptr, aoclsparse_int ∗bsr_nnz)

  aoclsparse_csr2bsr_nnz computes the number of nonzero block columns per row and the total number of nonzero blocks in a sparse BSR matrix given a sparse CSR matrix as input.

- DLL_PUBLIC aoclsparse_status aoclsparse_convert_csr (const aoclsparse_matrix src_mat, const aoclsparse_operation op, aoclsparse_matrix *dest_mat)

  Convert internal representation of matrix into a sparse CSR matrix.

- DLL_PUBLIC aoclsparse_status aoclsparse_scsr2ell (aoclsparse_int m, const aoclsparse_mat_descr descr, const aoclsparse_int *csr_row_ptr, const aoclsparse_int *csr_col_ind, const float *csr_val, aoclsparse_int *ell_col_ind, float *ell_val, aoclsparse_int ell_width)

  Convert a sparse CSR matrix into a sparse ELLPACK matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2ell (aoclsparse_int m, const aoclsparse_mat_descr descr, const aoclsparse_int *csr_row_ptr, const aoclsparse_int *csr_col_ind, const double *csr_val, aoclsparse_int *ell_col_ind, double *ell_val, aoclsparse_int ell_width)

  Convert a sparse CSR matrix into a sparse ELLPACK matrix.

- DLL_PUBLIC aoclsparse_status aoclsparse_scsr2dia (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, const aoclsparse_int *csr_row_ptr, const aoclsparse_int *csr_col_ind, const float *csr_val, aoclsparse_int dia_num_diag, aoclsparse_int *dia_offset, float *dia_val)

  Convert a sparse CSR matrix into a sparse DIA matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2dia (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, const aoclsparse_int *csr_row_ptr, const aoclsparse_int *csr_col_ind, const double *csr_val, aoclsparse_int dia_num_diag, aoclsparse_int *dia_offset, double *dia_val)

  Convert a sparse CSR matrix into a sparse DIA matrix.

- DLL_PUBLIC aoclsparse_status aoclsparse_scsr2bsr (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, const float *csr_val, const aoclsparse_int *csr_row_ptr, const aoclsparse_int *csr_col_ind, aoclsparse_int block_dim, float *bsr_val, aoclsparse_int *bsr_row_ptr, aoclsparse_int *bsr_col_ind)

  Convert a sparse CSR matrix into a sparse BSR matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2bsr (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, const double *csr_val, const aoclsparse_int *csr_row_ptr, const aoclsparse_int *csr_col_ind, aoclsparse_int block_dim, double *bsr_val, aoclsparse_int *bsr_row_ptr, aoclsparse_int *bsr_col_ind)

  Convert a sparse CSR matrix into a sparse BSR matrix.

- DLL_PUBLIC aoclsparse_status aoclsparse_scsr2csc (aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const aoclsparse_mat_descr descr, aoclsparse_index_base baseCSC, const aoclsparse_int *csr_row_ptr, const aoclsparse_int *csr_col_ind, const float *csr_val, aoclsparse_int *csc_row_ind, aoclsparse_int *csc_col_ptr, float *csc_val)

  Convert a sparse CSR matrix into a sparse CSC matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2csc (aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const aoclsparse_mat_descr descr, aoclsparse_index_base baseCSC, const aoclsparse_int *csr_row_ptr, const aoclsparse_int *csr_col_ind, const double *csr_val, aoclsparse_int *csc_row_ind, aoclsparse_int *csc_col_ptr, double *csc_val)

  Convert a sparse CSR matrix into a sparse CSC matrix.

- DLL_PUBLIC aoclsparse_status aoclsparse_ccsr2csc (aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const aoclsparse_mat_descr descr, aoclsparse_index_base baseCSC, const aoclsparse_int ∗csr_↩ row_ptr, const aoclsparse_int ∗csr_col_ind, const aoclsparse_float_complex ∗csr_val, aoclsparse_int ∗csc↩ _row_ind, aoclsparse_int ∗csc_col_ptr, aoclsparse_float_complex ∗csc_val)

  Convert a sparse CSR matrix into a sparse CSC matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_zcsr2csc (aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const aoclsparse_mat_descr descr, aoclsparse_index_base baseCSC, const aoclsparse_int ∗csr_row↩ _ptr, const aoclsparse_int ∗csr_col_ind, const aoclsparse_double_complex ∗csr_val, aoclsparse_int ∗csc_↩ row_ind, aoclsparse_int ∗csc_col_ptr, aoclsparse_double_complex ∗csc_val)

  Convert a sparse CSR matrix into a sparse CSC matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_scsr2dense (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, const float ∗csr_val, const aoclsparse_int ∗csr_row_ptr, const aoclsparse_int ∗csr_col_ind, float ∗A, aoclsparse_int ld, aoclsparse_order order)

  This function converts the sparse matrix in CSR format into a dense matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2dense (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, const double ∗csr_val, const aoclsparse_int ∗csr_row_ptr, const aoclsparse_int ∗csr_col_ind, double ∗A, aoclsparse_int ld, aoclsparse_order order)

  This function converts the sparse matrix in CSR format into a dense matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_ccsr2dense (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, const aoclsparse_float_complex ∗csr_val, const aoclsparse_int ∗csr_row_ptr, const aoclsparse_int ∗csr_col_ind, aoclsparse_float_complex ∗A, aoclsparse_int ld, aoclsparse_order order)

  This function converts the sparse matrix in CSR format into a dense matrix.
- DLL_PUBLIC aoclsparse_status aoclsparse_zcsr2dense (aoclsparse_int m, aoclsparse_int n, const aoclsparse_mat_descr descr, const aoclsparse_double_complex ∗csr_val, const aoclsparse_int ∗csr_row_↩ ptr, const aoclsparse_int ∗csr_col_ind, aoclsparse_double_complex ∗A, aoclsparse_int ld, aoclsparse_order order)

  This function converts the sparse matrix in CSR format into a dense matrix.

3.3.1 Detailed Description

aoclsparse_convert.h provides sparse format conversion subprograms

3.3.2 Function Documentation

3.3.2.1 aoclsparse_csr2ell_width()

```
DLL_PUBLIC aoclsparse_status aoclsparse_csr2ell_width (
        aoclsparse_int m,
        aoclsparse_int nnz,
        const aoclsparse_int * csr_row_ptr,
        aoclsparse_int * ell_width )
```

Convert a sparse CSR matrix into a sparse ELL matrix.

`aoclsparse_csr2ell_width` computes the maximum of the per row non-zero elements over all rows, the ELL `width`, for a given CSR matrix.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
out	*ell_width*	pointer to the number of non-zero elements per row in ELL storage format.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m` is invalid.
aoclsparse_status_invalid_pointer	`csr_row_ptr`, or `ell_width` pointer is invalid.
aoclsparse_status_internal_error	an internal error occurred.

3.3.2.2 aoclsparse_scsr2ell()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsr2ell (
        aoclsparse_int m,
        const aoclsparse_mat_descr descr,
        const aoclsparse_int * csr_row_ptr,
        const aoclsparse_int * csr_col_ind,
        const float * csr_val,
        aoclsparse_int * ell_col_ind,
        float * ell_val,
        aoclsparse_int ell_width )
```

Convert a sparse CSR matrix into a sparse ELLPACK matrix.

`aoclsparse_csr2ell` converts a CSR matrix into an ELL matrix. It is assumed, that `ell_val` and `ell_col_ind` are allocated. Allocation size is computed by the number of rows times the number of ELL non-zero elements per row, such that $\text{nnz}_{\text{ELL}} = m \cdot \text{ell\_width}$. The number of ELL non-zero elements per row is obtained by aoclsparse_csr2ell_width(). The index base is preserved during the conversion.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix. Only the base index is used in the conversion process, the remaining descriptor elements are ignored.
in	*csr_val*	array containing the values of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array containing the column indices of the sparse CSR matrix.
in	*ell_width*	number of non-zero elements per row in ELL storage format.
out	*ell_val*	array of `m` times `ell_width` elements of the sparse ELL matrix.
out	*ell_col_ind*	array of `m` times `ell_width` elements containing the column indices of the sparse ELL matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_handle	the library context was not initialized.
aoclsparse_status_invalid_size	`m` or `ell_width` is invalid.

Return values

aoclsparse_status_invalid_pointer	`csr_val`, `csr_row_ptr`, `csr_col_ind`, `ell_val` or `ell_col_ind` pointer is invalid.

3.3.2.3 aoclsparse_dcsr2ell()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2ell (
    aoclsparse_int m,
    const aoclsparse_mat_descr descr,
    const aoclsparse_int * csr_row_ptr,
    const aoclsparse_int * csr_col_ind,
    const double * csr_val,
    aoclsparse_int * ell_col_ind,
    double * ell_val,
    aoclsparse_int ell_width )
```

Convert a sparse CSR matrix into a sparse ELLPACK matrix.

`aoclsparse_csr2ell` converts a CSR matrix into an ELL matrix. It is assumed, that `ell_val` and `ell_col_ind` are allocated. Allocation size is computed by the number of rows times the number of ELL non-zero elements per row, such that $\text{nnz}_{\text{ELL}} = m \cdot \text{ell\_width}$. The number of ELL non-zero elements per row is obtained by aoclsparse_csr2ell_width(). The index base is preserved during the conversion.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix. Only the base index is used in the conversion process, the remaining descriptor elements are ignored.
in	*csr_val*	array containing the values of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array containing the column indices of the sparse CSR matrix.
in	*ell_width*	number of non-zero elements per row in ELL storage format.
out	*ell_val*	array of `m` times `ell_width` elements of the sparse ELL matrix.
out	*ell_col_ind*	array of `m` times `ell_width` elements containing the column indices of the sparse ELL matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_handle	the library context was not initialized.
aoclsparse_status_invalid_size	`m` or `ell_width` is invalid.
aoclsparse_status_invalid_pointer	`csr_val`, `csr_row_ptr`, `csr_col_ind`, `ell_val` or `ell_col_ind` pointer is invalid.

3.3.2.4 aoclsparse_csr2dia_ndiag()

```
DLL_PUBLIC aoclsparse_status aoclsparse_csr2dia_ndiag (
    aoclsparse_int m,
    aoclsparse_int n,
```

```
const aoclsparse_mat_descr descr,
aoclsparse_int nnz,
const aoclsparse_int * csr_row_ptr,
const aoclsparse_int * csr_col_ind,
aoclsparse_int * dia_num_diag )
```

Convert a sparse CSR matrix into a sparse DIA matrix.

`aoclsparse_csr2dia_ndiag` computes the number of the diagonals for a given CSR matrix.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*n*	number of cols of the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix. Only the base index is used in computing the diagonals, the remaining descriptor elements are ignored.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array containing the column indices of the sparse CSR matrix.
out	*dia_num_diag*	pointer to the number of diagonals with non-zeroes in DIA storage format.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m` is invalid.
aoclsparse_status_invalid_pointer	`csr_row_ptr`, or `ell_width` pointer is invalid.
aoclsparse_status_internal_error	an internal error occurred.

3.3.2.5 aoclsparse_scsr2dia()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsr2dia (
aoclsparse_int m,
aoclsparse_int n,
const aoclsparse_mat_descr descr,
const aoclsparse_int * csr_row_ptr,
const aoclsparse_int * csr_col_ind,
const float * csr_val,
aoclsparse_int dia_num_diag,
aoclsparse_int * dia_offset,
float * dia_val )
```

Convert a sparse CSR matrix into a sparse DIA matrix.

`aoclsparse_csr2dia` converts a CSR matrix into an DIA matrix. It is assumed, that `dia_val` and `dia_offset` are allocated. Allocation size is computed by the number of rows times the number of diagonals. The number of DIA diagonals is obtained by aoclsparse_csr2dia_ndiag().The index base is preserved during the conversion.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*n*	number of cols of the sparse CSR matrix.

Parameters

in	*descr*	descriptor of the input sparse CSR matrix. Only the base index is used in the conversion process, the remaining descriptor elements are ignored.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array containing the column indices of the sparse CSR matrix.
in	*csr_val*	array containing the values of the sparse CSR matrix.
in	*dia_num_diag*	number of diagoanls in ELL storage format.
out	*dia_offset*	array of `dia_num_diag` elements containing the diagonal offsets from main diagonal.
out	*dia_val*	array of `m` times `dia_num_diag` elements of the sparse DIA matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_handle	the library context was not initialized.
aoclsparse_status_invalid_size	`m` or `ell_width` is invalid.
aoclsparse_status_invalid_pointer	`csr_val`, `csr_row_ptr`, `csr_col_ind`, `ell_val` or `ell_col_ind` pointer is invalid.

3.3.2.6 aoclsparse_dcsr2dia()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2dia (
        aoclsparse_int m,
        aoclsparse_int n,
        const aoclsparse_mat_descr descr,
        const aoclsparse_int * csr_row_ptr,
        const aoclsparse_int * csr_col_ind,
        const double * csr_val,
        aoclsparse_int dia_num_diag,
        aoclsparse_int * dia_offset,
        double * dia_val )
```

Convert a sparse CSR matrix into a sparse DIA matrix.

`aoclsparse_csr2dia` converts a CSR matrix into an DIA matrix. It is assumed, that `dia_val` and `dia_offset` are allocated. Allocation size is computed by the number of rows times the number of diagonals. The number of DIA diagonals is obtained by aoclsparse_csr2dia_ndiag().The index base is preserved during the conversion.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*n*	number of cols of the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix. Only the base index is used in the conversion process, the remaining descriptor elements are ignored.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array containing the column indices of the sparse CSR matrix.
in	*csr_val*	array containing the values of the sparse CSR matrix.
in	*dia_num_diag*	number of diagoanls in ELL storage format.
out	*dia_offset*	array of `dia_num_diag` elements containing the diagonal offsets from main diagonal.
out	*dia_val*	array of `m` times `dia_num_diag` elements of the sparse DIA matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_handle	the library context was not initialized.
aoclsparse_status_invalid_size	`m` or `ell_width` is invalid.
aoclsparse_status_invalid_pointer	`csr_val`, `csr_row_ptr`, `csr_col_ind`, `ell_val` or `ell_col_ind` pointer is invalid.

3.3.2.7 aoclsparse_csr2bsr_nnz()

```
DLL_PUBLIC aoclsparse_status aoclsparse_csr2bsr_nnz (
        aoclsparse_int m,
        aoclsparse_int n,
        const aoclsparse_mat_descr descr,
        const aoclsparse_int * csr_row_ptr,
        const aoclsparse_int * csr_col_ind,
        aoclsparse_int block_dim,
        aoclsparse_int * bsr_row_ptr,
        aoclsparse_int * bsr_nnz )
```

aoclsparse_csr2bsr_nnz computes the number of nonzero block columns per row and the total number of nonzero blocks in a sparse BSR matrix given a sparse CSR matrix as input.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*n*	number of columns of the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix. Only the base index is used in computing the nnz blocks, the remaining descriptor elements are ignored.
in	*csr_row_ptr*	integer array containing `m+1` elements that point to the start of each row of the CSR matrix
in	*csr_col_ind*	integer array of the column indices for each non-zero element in the CSR matrix
in	*block_dim*	the block dimension of the BSR matrix. Between 1 and min(m, n)
out	*bsr_row_ptr*	integer array containing `mb+1` elements that point to the start of each block row of the BSR matrix
out	*bsr_nnz*	total number of nonzero elements in device or host memory.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m` or `n` or `block_dim` is invalid.
aoclsparse_status_invalid_pointer	`csr_row_ptr` or `csr_col_ind` or `bsr_row_ptr` or `bsr_nnz` pointer is invalid.

3.3.2.8 aoclsparse_scsr2bsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsr2bsr (
        aoclsparse_int m,
        aoclsparse_int n,
        const aoclsparse_mat_descr descr,
```

```
const float * csr_val,
const aoclsparse_int * csr_row_ptr,
const aoclsparse_int * csr_col_ind,
aoclsparse_int block_dim,
float * bsr_val,
aoclsparse_int * bsr_row_ptr,
aoclsparse_int * bsr_col_ind )
```

Convert a sparse CSR matrix into a sparse BSR matrix.

`aoclsparse_csr2bsr` converts a CSR matrix into a BSR matrix. It is assumed, that `bsr_val`, `bsr_col↩_ind` and `bsr_row_ptr` are allocated. Allocation size for `bsr_row_ptr` is computed as `mb+1` where `mb` is the number of block rows in the BSR matrix. Allocation size for `bsr_val` and `bsr_col_ind` is computed using `csr2bsr_nnz()` which also fills in `bsr_row_ptr`. The index base is preserved during the conversion.

Parameters

in	*m*	number of rows in the sparse CSR matrix.
in	*n*	number of columns in the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix. Only the base index is used in the conversion process, the remaining descriptor elements are ignored.
in	*csr_val*	array of `nnz` elements containing the values of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
in	*block_dim*	size of the blocks in the sparse BSR matrix.
out	*bsr_val*	array of `nnzb*block_dim*block_dim` containing the values of the sparse BSR matrix.
out	*bsr_row_ptr*	array of `mb+1` elements that point to the start of every block row of the sparse BSR matrix.
out	*bsr_col_ind*	array of `nnzb` elements containing the block column indices of the sparse BSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m` or `n` or `block_dim` is invalid.
aoclsparse_status_invalid_pointer	`bsr_val`, `bsr_row_ptr`, `bsr_col_ind`, `csr_val`, `csr_row_ptr` or `csr_col_ind` pointer is invalid.

3.3.2.9 aoclsparse_dcsr2bsr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2bsr (
aoclsparse_int m,
aoclsparse_int n,
const aoclsparse_mat_descr descr,
const double * csr_val,
const aoclsparse_int * csr_row_ptr,
const aoclsparse_int * csr_col_ind,
aoclsparse_int block_dim,
double * bsr_val,
aoclsparse_int * bsr_row_ptr,
aoclsparse_int * bsr_col_ind )
```

Convert a sparse CSR matrix into a sparse BSR matrix.

aoclsparse_csr2bsr converts a CSR matrix into a BSR matrix. It is assumed, that bsr_val, bsr_col←
_ind and bsr_row_ptr are allocated. Allocation size for bsr_row_ptr is computed as mb+1 where mb is the number of block rows in the BSR matrix. Allocation size for bsr_val and bsr_col_ind is computed using csr2bsr_nnz() which also fills in bsr_row_ptr. The index base is preserved during the conversion.

Parameters

in	*m*	number of rows in the sparse CSR matrix.
in	*n*	number of columns in the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix. Only the base index is used in the conversion process, the remaining descriptor elements are ignored.
in	*csr_val*	array of nnz elements containing the values of the sparse CSR matrix.
in	*csr_row_ptr*	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array of nnz elements containing the column indices of the sparse CSR matrix.
in	*block_dim*	size of the blocks in the sparse BSR matrix.
out	*bsr_val*	array of nnzb*block_dim*block_dim containing the values of the sparse BSR matrix.
out	*bsr_row_ptr*	array of mb+1 elements that point to the start of every block row of the sparse BSR matrix.
out	*bsr_col_ind*	array of nnzb elements containing the block column indices of the sparse BSR matrix.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	m or n or block_dim is invalid.
aoclsparse_status_invalid_pointer	bsr_val, bsr_row_ptr, bsr_col_ind, csr_val, csr_row_ptr or csr_col_ind pointer is invalid.

3.3.2.10 aoclsparse_scsr2csc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsr2csc (
        aoclsparse_int m,
        aoclsparse_int n,
        aoclsparse_int nnz,
        const aoclsparse_mat_descr descr,
        aoclsparse_index_base baseCSC,
        const aoclsparse_int * csr_row_ptr,
        const aoclsparse_int * csr_col_ind,
        const float * csr_val,
        aoclsparse_int * csc_row_ind,
        aoclsparse_int * csc_col_ptr,
        float * csc_val )
```

Convert a sparse CSR matrix into a sparse CSC matrix.

aoclsparse_csr2csc converts a CSR matrix into a CSC matrix. aoclsparse_csr2csc can also be used to convert a CSC matrix into a CSR matrix. The index base can be modified during the conversion.

Note

The resulting matrix can also be seen as the transpose of the input matrix.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*n*	number of columns of the sparse CSR matrix.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix.Only the base index is used in the conversion process, the remaining descriptor elements are ignored.
in	*baseCSC*	the desired index base (zero or one) for the converted matrix.
in	*csr_val*	array of `nnz` elements of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
out	*csc_val*	array of `nnz` elements of the sparse CSC matrix.
out	*csc_row_ind*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
out	*csc_col_ptr*	array of `n+1` elements that point to the start of every column of the sparse CSC matrix. aoclsparse_csr2csc_buffer_size().

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_pointer	`csr_val`, `csr_row_ptr`, `csr_col_ind`, `csc_val`, `csc_row_ind`, `csc_col_ptr` is invalid.

3.3.2.11 aoclsparse_dcsr2csc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2csc (
        aoclsparse_int m,
        aoclsparse_int n,
        aoclsparse_int nnz,
        const aoclsparse_mat_descr descr,
        aoclsparse_index_base baseCSC,
        const aoclsparse_int * csr_row_ptr,
        const aoclsparse_int * csr_col_ind,
        const double * csr_val,
        aoclsparse_int * csc_row_ind,
        aoclsparse_int * csc_col_ptr,
        double * csc_val )
```

Convert a sparse CSR matrix into a sparse CSC matrix.

`aoclsparse_csr2csc` converts a CSR matrix into a CSC matrix. `aoclsparse_csr2csc` can also be used to convert a CSC matrix into a CSR matrix. The index base can be modified during the conversion.

Note

The resulting matrix can also be seen as the transpose of the input matrix.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*n*	number of columns of the sparse CSR matrix.

Parameters

in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix.Only the base index is used in the conversion process, the remaining descriptor elements are ignored.
in	*baseCSC*	the desired index base (zero or one) for the converted matrix.
in	*csr_val*	array of `nnz` elements of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
out	*csc_val*	array of `nnz` elements of the sparse CSC matrix.
out	*csc_row_ind*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
out	*csc_col_ptr*	array of `n+1` elements that point to the start of every column of the sparse CSC matrix. aoclsparse_csr2csc_buffer_size().

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_pointer	`csr_val`, `csr_row_ptr`, `csr_col_ind`, `csc_val`, `csc_row_ind`, `csc_col_ptr` is invalid.

3.3.2.12 aoclsparse_ccsr2csc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ccsr2csc (
        aoclsparse_int m,
        aoclsparse_int n,
        aoclsparse_int nnz,
        const aoclsparse_mat_descr descr,
        aoclsparse_index_base baseCSC,
        const aoclsparse_int * csr_row_ptr,
        const aoclsparse_int * csr_col_ind,
        const aoclsparse_float_complex * csr_val,
        aoclsparse_int * csc_row_ind,
        aoclsparse_int * csc_col_ptr,
        aoclsparse_float_complex * csc_val )
```

Convert a sparse CSR matrix into a sparse CSC matrix.

`aoclsparse_csr2csc` converts a CSR matrix into a CSC matrix. `aoclsparse_csr2csc` can also be used to convert a CSC matrix into a CSR matrix. The index base can be modified during the conversion.

Note

The resulting matrix can also be seen as the transpose of the input matrix.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*n*	number of columns of the sparse CSR matrix.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix.Only the base index is used in the conversion process, the remaining descriptor elements are ignored.

Parameters

in	*baseCSC*	the desired index base (zero or one) for the converted matrix.
in	*csr_val*	array of `nnz` elements of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
out	*csc_val*	array of `nnz` elements of the sparse CSC matrix.
out	*csc_row_ind*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
out	*csc_col_ptr*	array of `n+1` elements that point to the start of every column of the sparse CSC matrix. aoclsparse_csr2csc_buffer_size().

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_pointer	`csr_val`, `csr_row_ptr`, `csr_col_ind`, `csc_val`, `csc_row_ind`, `csc_col_ptr` is invalid.

3.3.2.13 aoclsparse_zcsr2csc()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zcsr2csc (
    aoclsparse_int m,
    aoclsparse_int n,
    aoclsparse_int nnz,
    const aoclsparse_mat_descr descr,
    aoclsparse_index_base baseCSC,
    const aoclsparse_int * csr_row_ptr,
    const aoclsparse_int * csr_col_ind,
    const aoclsparse_double_complex * csr_val,
    aoclsparse_int * csc_row_ind,
    aoclsparse_int * csc_col_ptr,
    aoclsparse_double_complex * csc_val )
```

Convert a sparse CSR matrix into a sparse CSC matrix.

`aoclsparse_csr2csc` converts a CSR matrix into a CSC matrix. `aoclsparse_csr2csc` can also be used to convert a CSC matrix into a CSR matrix. The index base can be modified during the conversion.

Note

The resulting matrix can also be seen as the transpose of the input matrix.

Parameters

in	*m*	number of rows of the sparse CSR matrix.
in	*n*	number of columns of the sparse CSR matrix.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*descr*	descriptor of the input sparse CSR matrix.Only the base index is used in the conversion process, the remaining descriptor elements are ignored.
in	*baseCSC*	the desired index base (zero or one) for the converted matrix.
in	*csr_val*	array of `nnz` elements of the sparse CSR matrix.

Parameters

in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
out	*csc_val*	array of `nnz` elements of the sparse CSC matrix.
out	*csc_row_ind*	array of `nnz` elements containing the row indices of the sparse CSC matrix.
out	*csc_col_ptr*	array of `n+1` elements that point to the start of every column of the sparse CSC matrix. aoclsparse_csr2csc_buffer_size().

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_pointer	`csr_val`, `csr_row_ptr`, `csr_col_ind`, `csc_val`, `csc_row_ind`, `csc_col_ptr` is invalid.

3.3.2.14 aoclsparse_scsr2dense()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsr2dense (
        aoclsparse_int m,
        aoclsparse_int n,
        const aoclsparse_mat_descr descr,
        const float * csr_val,
        const aoclsparse_int * csr_row_ptr,
        const aoclsparse_int * csr_col_ind,
        float * A,
        aoclsparse_int ld,
        aoclsparse_order order )
```

This function converts the sparse matrix in CSR format into a dense matrix.

Parameters

in	*m*	number of rows of the dense matrix `A`.
in	*n*	number of columns of the dense matrix `A`.
in	*descr*	the descriptor of the dense matrix `A`, the supported matrix type is aoclsparse_matrix_type_general. Base index from the descriptor is used in the conversion process.
in	*csr_val*	array of nnz (= `csr_row_ptr`[m] - `csr_row_ptr`[0]) nonzero elements of matrix `A`.
in	*csr_row_ptr*	integer array of m+1 elements that contains the start of every row and the end of the last row plus one.
in	*csr_col_ind*	integer array of nnz (= `csr_row_ptr`[m] - csr_row_ptr[0]) column indices of the non-zero elements of matrix `A`.
out	*A*	array of dimensions (`ld`, `n`)
in	*ld*	leading dimension of dense array `A`.
in	*order*	memory layout of a dense matrix `A`.

Return values

aoclsparse_status_success	the operation completed successfully.

Return values

aoclsparse_status_invalid_size	m or n or ld is invalid.
aoclsparse_status_invalid_pointer	A or csr_val csr_row_ptr or csr_col_ind pointer is invalid.

3.3.2.15 aoclsparse_dcsr2dense()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2dense (
        aoclsparse_int m,
        aoclsparse_int n,
        const aoclsparse_mat_descr descr,
        const double * csr_val,
        const aoclsparse_int * csr_row_ptr,
        const aoclsparse_int * csr_col_ind,
        double * A,
        aoclsparse_int ld,
        aoclsparse_order order )
```

This function converts the sparse matrix in CSR format into a dense matrix.

Parameters

in	*m*	number of rows of the dense matrix A.
in	*n*	number of columns of the dense matrix A.
in	*descr*	the descriptor of the dense matrix A, the supported matrix type is aoclsparse_matrix_type_general. Base index from the descriptor is used in the conversion process.
in	*csr_val*	array of nnz (= csr_row_ptr[m] - csr_row_ptr[0]) nonzero elements of matrix A.
in	*csr_row_ptr*	integer array of m+1 elements that contains the start of every row and the end of the last row plus one.
in	*csr_col_ind*	integer array of nnz (= csr_row_ptr[m] - csr_row_ptr[0]) column indices of the non-zero elements of matrix A.
out	*A*	array of dimensions (ld, n)
in	*ld*	leading dimension of dense array A.
in	*order*	memory layout of a dense matrix A.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	m or n or ld is invalid.
aoclsparse_status_invalid_pointer	A or csr_val csr_row_ptr or csr_col_ind pointer is invalid.

3.3.2.16 aoclsparse_ccsr2dense()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ccsr2dense (
        aoclsparse_int m,
        aoclsparse_int n,
        const aoclsparse_mat_descr descr,
        const aoclsparse_float_complex * csr_val,
```

```
const aoclsparse_int * csr_row_ptr,
const aoclsparse_int * csr_col_ind,
aoclsparse_float_complex * A,
aoclsparse_int ld,
aoclsparse_order order )
```

This function converts the sparse matrix in CSR format into a dense matrix.

Parameters

in	*m*	number of rows of the dense matrix A.
in	*n*	number of columns of the dense matrix A.
in	*descr*	the descriptor of the dense matrix A, the supported matrix type is aoclsparse_matrix_type_general. Base index from the descriptor is used in the conversion process.
in	*csr_val*	array of nnz (= csr_row_ptr[m] - csr_row_ptr[0]) nonzero elements of matrix A.
in	*csr_row_ptr*	integer array of m+1 elements that contains the start of every row and the end of the last row plus one.
in	*csr_col_ind*	integer array of nnz (= csr_row_ptr[m] - csr_row_ptr[0]) column indices of the non-zero elements of matrix A.
out	*A*	array of dimensions (ld, n)
in	*ld*	leading dimension of dense array A.
in	*order*	memory layout of a dense matrix A.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	m or n or ld is invalid.
aoclsparse_status_invalid_pointer	A or csr_val csr_row_ptr or csr_col_ind pointer is invalid.

3.3.2.17 aoclsparse_zcsr2dense()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zcsr2dense (
aoclsparse_int m,
aoclsparse_int n,
const aoclsparse_mat_descr descr,
const aoclsparse_double_complex * csr_val,
const aoclsparse_int * csr_row_ptr,
const aoclsparse_int * csr_col_ind,
aoclsparse_double_complex * A,
aoclsparse_int ld,
aoclsparse_order order )
```

This function converts the sparse matrix in CSR format into a dense matrix.

Parameters

in	*m*	number of rows of the dense matrix A.
in	*n*	number of columns of the dense matrix A.
in	*descr*	the descriptor of the dense matrix A, the supported matrix type is aoclsparse_matrix_type_general. Base index from the descriptor is used in the conversion process.

Parameters

in	*csr_val*	array of nnz (= csr_row_ptr[m] - csr_row_ptr[0]) nonzero elements of matrix A.
in	*csr_row_ptr*	integer array of m+1 elements that contains the start of every row and the end of the last row plus one.
in	*csr_col_ind*	integer array of nnz (= csr_row_ptr[m] - csr_row_ptr[0]) column indices of the non-zero elements of matrix A.
out	*A*	array of dimensions (ld, n)
in	*ld*	leading dimension of dense array A.
in	*order*	memory layout of a dense matrix A.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	m or n or ld is invalid.
aoclsparse_status_invalid_pointer	A or csr_val csr_row_ptr or csr_col_ind pointer is invalid.

3.3.2.18 aoclsparse_convert_csr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_convert_csr (
        const aoclsparse_matrix src_mat,
        const aoclsparse_operation op,
        aoclsparse_matrix * dest_mat )
```

Convert internal representation of matrix into a sparse CSR matrix.

aoclsparse_convert_csr converts any supported matrix format into a CSR format matrix and returns it as a new aoclsparse_matrix. The new matrix can also be transposed or conjugate transposed during the conversion. It should be freed by calling aoclsparse_destroy. The source matrix needs to be initalized using aoclsparse_create_(d/s/c/z)(coo/csc/csr) and it is not modified here.

Parameters

in	*src_mat*	source matrix used for conversion.
in	*op*	operation to be performed on destination matrix
out	*dest_mat*	destination matrix output in CSR Format of the src_mat.

Return values

aoclsparse_status_success	the operation completed successfully
aoclsparse_status_invalid_size	matrix dimension are invalid
aoclsparse_status_invalid_pointer	pointers in src_mat or dest_mat are invalid
aoclsparse_status_not_implemented	conversion of the src_mat format given is not implemented
aoclsparse_status_memory_error	memory allocation for destination matrix failed

3.4 aoclsparse_functions.h File Reference

aoclsparse_functions.h provides AMD CPU hardware optimized level 1, 2, and 3 Sparse Linear Algebra Subprograms (Sparse BLAS)

Functions

- DLL_PUBLIC aoclsparse_status aoclsparse_saxpyi (const aoclsparse_int nnz, const float a, const float ∗x, const aoclsparse_int ∗indx, float ∗y)

  A variant of sparse vector-vector addition between a compressed sparse vector and a dense vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_daxpyi (const aoclsparse_int nnz, const double a, const double ∗x, const aoclsparse_int ∗indx, double ∗y)

  A variant of sparse vector-vector addition between a compressed sparse vector and a dense vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_caxpyi (const aoclsparse_int nnz, const void ∗a, const void ∗x, const aoclsparse_int ∗indx, void ∗y)

  A variant of sparse vector-vector addition between a compressed sparse vector and a dense vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_zaxpyi (const aoclsparse_int nnz, const void ∗a, const void ∗x, const aoclsparse_int ∗indx, void ∗y)

  A variant of sparse vector-vector addition between a compressed sparse vector and a dense vector.

- DLL_PUBLIC aoclsparse_status aoclsparse_cdotci (const aoclsparse_int nnz, const void ∗x, const aoclsparse_int ∗indx, const void ∗y, void ∗dot)

  Sparse conjugate dot product for single and double data precision complex types.
- DLL_PUBLIC aoclsparse_status aoclsparse_zdotci (const aoclsparse_int nnz, const void ∗x, const aoclsparse_int ∗indx, const void ∗y, void ∗dot)

  Sparse conjugate dot product for single and double data precision complex types.

- DLL_PUBLIC aoclsparse_status aoclsparse_cdotui (const aoclsparse_int nnz, const void ∗x, const aoclsparse_int ∗indx, const void ∗y, void ∗dot)

  Sparse dot product for single and double data precision complex types.
- DLL_PUBLIC aoclsparse_status aoclsparse_zdotui (const aoclsparse_int nnz, const void ∗x, const aoclsparse_int ∗indx, const void ∗y, void ∗dot)

  Sparse dot product for single and double data precision complex types.

- DLL_PUBLIC float aoclsparse_sdoti (const aoclsparse_int nnz, const float ∗x, const aoclsparse_int ∗indx, const float ∗y)

  Sparse dot product for single and double data precision real types.
- DLL_PUBLIC double aoclsparse_ddoti (const aoclsparse_int nnz, const double ∗x, const aoclsparse_int ∗indx, const double ∗y)

  Sparse dot product for single and double data precision real types.

- DLL_PUBLIC aoclsparse_status aoclsparse_ssctr (const aoclsparse_int nnz, const float ∗x, const aoclsparse_int ∗indx, float ∗y)

    Sparse scatter for single and double precision real and complex types.
- DLL_PUBLIC aoclsparse_status aoclsparse_dsctr (const aoclsparse_int nnz, const double ∗x, const aoclsparse_int ∗indx, double ∗y)

    Sparse scatter for single and double precision real and complex types.
- DLL_PUBLIC aoclsparse_status aoclsparse_csctr (const aoclsparse_int nnz, const void ∗x, const aoclsparse_int ∗indx, void ∗y)

    Sparse scatter for single and double precision real and complex types.
- DLL_PUBLIC aoclsparse_status aoclsparse_zsctr (const aoclsparse_int nnz, const void ∗x, const aoclsparse_int ∗indx, void ∗y)

    Sparse scatter for single and double precision real and complex types.

- DLL_PUBLIC aoclsparse_status aoclsparse_ssctrs (const aoclsparse_int nnz, const float ∗x, aoclsparse_int stride, float ∗y)

    Sparse scatter with stride for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_dsctrs (const aoclsparse_int nnz, const double ∗x, aoclsparse_int stride, double ∗y)

    Sparse scatter with stride for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_csctrs (const aoclsparse_int nnz, const void ∗x, aoclsparse_int stride, void ∗y)

    Sparse scatter with stride for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_zsctrs (const aoclsparse_int nnz, const void ∗x, aoclsparse_int stride, void ∗y)

    Sparse scatter with stride for real/complex single and double data precisions.

- DLL_PUBLIC aoclsparse_status aoclsparse_sroti (const aoclsparse_int nnz, float ∗x, const aoclsparse_int ∗indx, float ∗y, const float c, const float s)

    Applies Givens rotations to single and double precision real vectors.
- DLL_PUBLIC aoclsparse_status aoclsparse_droti (const aoclsparse_int nnz, double ∗x, const aoclsparse_int ∗indx, double ∗y, const double c, const double s)

    Applies Givens rotations to single and double precision real vectors.

- DLL_PUBLIC aoclsparse_status aoclsparse_sgthr (aoclsparse_int nnz, const float ∗y, float ∗x, const aoclsparse_int ∗indx)

    Gather elements from a dense vector and store them into a sparse vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_dgthr (aoclsparse_int nnz, const double ∗y, double ∗x, const aoclsparse_int ∗indx)

    Gather elements from a dense vector and store them into a sparse vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_cgthr (aoclsparse_int nnz, const void ∗y, void ∗x, const aoclsparse_int ∗indx)

    Gather elements from a dense vector and store them into a sparse vector.

- DLL_PUBLIC aoclsparse_status aoclsparse_zgthr (aoclsparse_int nnz, const void *y, void *x, const aoclsparse_int *indx)

    Gather elements from a dense vector and store them into a sparse vector.

- DLL_PUBLIC aoclsparse_status aoclsparse_sgthrz (aoclsparse_int nnz, float *y, float *x, const aoclsparse_int *indx)

    Gather and zero out elements from a dense vector and store them into a sparse vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_dgthrz (aoclsparse_int nnz, double *y, double *x, const aoclsparse_int *indx)

    Gather and zero out elements from a dense vector and store them into a sparse vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_cgthrz (aoclsparse_int nnz, void *y, void *x, const aoclsparse_int *indx)

    Gather and zero out elements from a dense vector and store them into a sparse vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_zgthrz (aoclsparse_int nnz, void *y, void *x, const aoclsparse_int *indx)

    Gather and zero out elements from a dense vector and store them into a sparse vector.

- DLL_PUBLIC aoclsparse_status aoclsparse_sgthrs (aoclsparse_int nnz, const float *y, float *x, aoclsparse_int stride)

    Gather elements from a dense vector using a stride and store them into a sparse vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_dgthrs (aoclsparse_int nnz, const double *y, double *x, aoclsparse_int stride)

    Gather elements from a dense vector using a stride and store them into a sparse vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_cgthrs (aoclsparse_int nnz, const void *y, void *x, aoclsparse_int stride)

    Gather elements from a dense vector using a stride and store them into a sparse vector.
- DLL_PUBLIC aoclsparse_status aoclsparse_zgthrs (aoclsparse_int nnz, const void *y, void *x, aoclsparse_int stride)

    Gather elements from a dense vector using a stride and store them into a sparse vector.

- DLL_PUBLIC aoclsparse_status aoclsparse_scsrmv (aoclsparse_operation trans, const float *alpha, aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const float *csr_val, const aoclsparse_int *csr_col_ind, const aoclsparse_int *csr_row_ptr, const aoclsparse_mat_descr descr, const float *x, const float *beta, float *y)

    Single and double precision sparse matrix vector multiplication using CSR storage format.
- DLL_PUBLIC aoclsparse_status aoclsparse_dcsrmv (aoclsparse_operation trans, const double *alpha, aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const double *csr_val, const aoclsparse_int *csr_col_ind, const aoclsparse_int *csr_row_ptr, const aoclsparse_mat_descr descr, const double *x, const double *beta, double *y)

    Single and double precision sparse matrix vector multiplication using CSR storage format.

- DLL_PUBLIC aoclsparse_status aoclsparse_sellmv (aoclsparse_operation trans, const float *alpha, aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const float *ell_val, const aoclsparse_int *ell↩ _col_ind, aoclsparse_int ell_width, const aoclsparse_mat_descr descr, const float *x, const float *beta, float *y)

  Single & Double precision sparse matrix vector multiplication using ELL storage format.
- DLL_PUBLIC aoclsparse_status aoclsparse_dellmv (aoclsparse_operation trans, const double *alpha, aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const double *ell_val, const aoclsparse_int *ell_↩ col_ind, aoclsparse_int ell_width, const aoclsparse_mat_descr descr, const double *x, const double *beta, double *y)

  Single & Double precision sparse matrix vector multiplication using ELL storage format.

- DLL_PUBLIC aoclsparse_status aoclsparse_sdiamv (aoclsparse_operation trans, const float *alpha, aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const float *dia_val, const aoclsparse_int *dia_↩ offset, aoclsparse_int dia_num_diag, const aoclsparse_mat_descr descr, const float *x, const float *beta, float *y)

  Single & Double precision sparse matrix vector multiplication using DIA storage format.
- DLL_PUBLIC aoclsparse_status aoclsparse_ddiamv (aoclsparse_operation trans, const double *alpha, aoclsparse_int m, aoclsparse_int n, aoclsparse_int nnz, const double *dia_val, const aoclsparse_int *dia↩ _offset, aoclsparse_int dia_num_diag, const aoclsparse_mat_descr descr, const double *x, const double *beta, double *y)

  Single & Double precision sparse matrix vector multiplication using DIA storage format.

- DLL_PUBLIC aoclsparse_status aoclsparse_sbsrmv (aoclsparse_operation trans, const float *alpha, aoclsparse_int mb, aoclsparse_int nb, aoclsparse_int bsr_dim, const float *bsr_val, const aoclsparse_int *bsr_col_ind, const aoclsparse_int *bsr_row_ptr, const aoclsparse_mat_descr descr, const float *x, const float *beta, float *y)

  Single & Double precision Sparse matrix vector multiplication using BSR storage format.
- DLL_PUBLIC aoclsparse_status aoclsparse_dbsrmv (aoclsparse_operation trans, const double *alpha, aoclsparse_int mb, aoclsparse_int nb, aoclsparse_int bsr_dim, const double *bsr_val, const aoclsparse_int *bsr_col_ind, const aoclsparse_int *bsr_row_ptr, const aoclsparse_mat_descr descr, const double *x, const double *beta, double *y)

  Single & Double precision Sparse matrix vector multiplication using BSR storage format.

- DLL_PUBLIC aoclsparse_status aoclsparse_smv (aoclsparse_operation op, const float *alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const float *x, const float *beta, float *y)

  Computes sparse matrix vector multiplication for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_dmv (aoclsparse_operation op, const double *alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const double *x, const double *beta, double *y)

  Computes sparse matrix vector multiplication for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_cmv (aoclsparse_operation op, const aoclsparse_float_complex *alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const aoclsparse_float_complex *x, const aoclsparse_float_complex *beta, aoclsparse_float_complex *y)

  Computes sparse matrix vector multiplication for real/complex single and double data precisions.

- DLL_PUBLIC aoclsparse_status aoclsparse_zmv (aoclsparse_operation op, const aoclsparse_double_complex *alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const aoclsparse_double_complex *x, const aoclsparse_double_complex *beta, aoclsparse_double_complex *y)

  Computes sparse matrix vector multiplication for real/complex single and double data precisions.

- DLL_PUBLIC aoclsparse_status aoclsparse_scsrsv (aoclsparse_operation trans, const float *alpha, aoclsparse_int m, const float *csr_val, const aoclsparse_int *csr_col_ind, const aoclsparse_int *csr_row_ptr, const aoclsparse_mat_descr descr, const float *x, float *y)

  Sparse triangular solve using CSR storage format for single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_dcsrsv (aoclsparse_operation trans, const double *alpha, aoclsparse_int m, const double *csr_val, const aoclsparse_int *csr_col_ind, const aoclsparse_int *csr_row_ptr, const aoclsparse_mat_descr descr, const double *x, double *y)

  Sparse triangular solve using CSR storage format for single and double data precisions.

- DLL_PUBLIC aoclsparse_status aoclsparse_strsv (aoclsparse_operation trans, const float alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const float *b, float *x)

  Sparse triangular solver for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_dtrsv (aoclsparse_operation trans, const double alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const double *b, double *x)

  Sparse triangular solver for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_ctrsv (aoclsparse_operation trans, const aoclsparse_float_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const aoclsparse_float_complex *b, aoclsparse_float_complex *x)

  Sparse triangular solver for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_ztrsv (aoclsparse_operation trans, const aoclsparse_double_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const aoclsparse_double_complex *b, aoclsparse_double_complex *x)

  Sparse triangular solver for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_strsv_kid (aoclsparse_operation trans, const float alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const float *b, float *x, const aoclsparse_int kid)

  Sparse triangular solver for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_dtrsv_kid (aoclsparse_operation trans, const double alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const double *b, double *x, const aoclsparse_int kid)

  Sparse triangular solver for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_ctrsv_kid (aoclsparse_operation trans, const aoclsparse_float_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const aoclsparse_float_complex *b, aoclsparse_float_complex *x, const aoclsparse_int kid)

  Sparse triangular solver for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_ztrsv_kid (aoclsparse_operation trans, const aoclsparse_double_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const aoclsparse_double_complex *b, aoclsparse_double_complex *x, const aoclsparse_int kid)

  Sparse triangular solver for real/complex single and double data precisions.

- DLL_PUBLIC aoclsparse_status aoclsparse_sdotmv (const aoclsparse_operation op, const float alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const float ∗x, const float beta, float ∗y, float ∗d)

  Performs sparse matrix-vector multiplication followed by vector-vector multiplication.
- DLL_PUBLIC aoclsparse_status aoclsparse_ddotmv (const aoclsparse_operation op, const double alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const double ∗x, const double beta, double ∗y, double ∗d)

  Performs sparse matrix-vector multiplication followed by vector-vector multiplication.
- DLL_PUBLIC aoclsparse_status aoclsparse_cdotmv (const aoclsparse_operation op, const aoclsparse_float_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const aoclsparse_float_complex ∗x, const aoclsparse_float_complex beta, aoclsparse_float_complex ∗y, aoclsparse_float_complex ∗d)

  Performs sparse matrix-vector multiplication followed by vector-vector multiplication.
- DLL_PUBLIC aoclsparse_status aoclsparse_zdotmv (const aoclsparse_operation op, const aoclsparse_double_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, const aoclsparse_double_complex ∗x, const aoclsparse_double_complex beta, aoclsparse_double_complex ∗y, aoclsparse_double_complex ∗d)

  Performs sparse matrix-vector multiplication followed by vector-vector multiplication.

- DLL_PUBLIC aoclsparse_status aoclsparse_strsm (const aoclsparse_operation trans, const float alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const float ∗B, aoclsparse_int n, aoclsparse_int ldb, float ∗X, aoclsparse_int ldx)

  Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_dtrsm (const aoclsparse_operation trans, const double alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const double ∗B, aoclsparse_int n, aoclsparse_int ldb, double ∗X, aoclsparse_int ldx)

  Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_ctrsm (aoclsparse_operation trans, const aoclsparse_float_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const aoclsparse_float_complex ∗B, aoclsparse_int n, aoclsparse_int ldb, aoclsparse_float_complex ∗X, aoclsparse_int ldx)

  Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_ztrsm (aoclsparse_operation trans, const aoclsparse_double_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const aoclsparse_double_complex ∗B, aoclsparse_int n, aoclsparse_int ldb, aoclsparse_double_complex ∗X, aoclsparse_int ldx)

  Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_strsm_kid (const aoclsparse_operation trans, const float alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const float ∗B, aoclsparse_int n, aoclsparse_int ldb, float ∗X, aoclsparse_int ldx, const aoclsparse_int kid)

  Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_dtrsm_kid (const aoclsparse_operation trans, const double alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const double ∗B, aoclsparse_int n, aoclsparse_int ldb, double ∗X, aoclsparse_int ldx, const aoclsparse_int kid)

  Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

- DLL_PUBLIC aoclsparse_status aoclsparse_ctrsm_kid (aoclsparse_operation trans, const aoclsparse_↩ float_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const aoclsparse_float_complex ∗B, aoclsparse_int n, aoclsparse_int ldb, aoclsparse_float_complex ∗X, aoclsparse_int ldx, const aoclsparse_int kid)

    Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.
- DLL_PUBLIC aoclsparse_status aoclsparse_ztrsm_kid (aoclsparse_operation trans, const aoclsparse_↩ double_complex alpha, aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const aoclsparse_double_complex ∗B, aoclsparse_int n, aoclsparse_int ldb, aoclsparse_double_complex ∗X, aoclsparse_int ldx, const aoclsparse_int kid)

    Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

- DLL_PUBLIC aoclsparse_status aoclsparse_sp2m (aoclsparse_operation opA, const aoclsparse_mat_descr descrA, const aoclsparse_matrix A, aoclsparse_operation opB, const aoclsparse_mat_descr descrB, const aoclsparse_matrix B, const aoclsparse_request request, aoclsparse_matrix ∗C)

    Sparse matrix Sparse matrix multiplication for real and complex datatypes.

- DLL_PUBLIC aoclsparse_status aoclsparse_spmm (aoclsparse_operation opA, const aoclsparse_matrix A, const aoclsparse_matrix B, aoclsparse_matrix ∗C)

    Sparse matrix Sparse matrix multiplication for real and complex datatypes.

- DLL_PUBLIC aoclsparse_status aoclsparse_scsrmm (aoclsparse_operation op, const float alpha, const aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const float ∗B, aoclsparse_int n, aoclsparse_int ldb, const float beta, float ∗C, aoclsparse_int ldc)

    Sparse matrix dense matrix multiplication using CSR storage format.
- DLL_PUBLIC aoclsparse_status aoclsparse_dcsrmm (aoclsparse_operation op, const double alpha, const aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const double ∗B, aoclsparse_int n, aoclsparse_int ldb, const double beta, double ∗C, aoclsparse_int ldc)

    Sparse matrix dense matrix multiplication using CSR storage format.
- DLL_PUBLIC aoclsparse_status aoclsparse_ccsrmm (aoclsparse_operation op, const aoclsparse_float_↩ complex alpha, const aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const aoclsparse_float_complex ∗B, aoclsparse_int n, aoclsparse_int ldb, const aoclsparse_float_complex beta, aoclsparse_float_complex ∗C, aoclsparse_int ldc)

    Sparse matrix dense matrix multiplication using CSR storage format.
- DLL_PUBLIC aoclsparse_status aoclsparse_zcsrmm (aoclsparse_operation op, const aoclsparse_double↩ _complex alpha, const aoclsparse_matrix A, const aoclsparse_mat_descr descr, aoclsparse_order order, const aoclsparse_double_complex ∗B, aoclsparse_int n, aoclsparse_int ldb, const aoclsparse_double_↩ complex beta, aoclsparse_double_complex ∗C, aoclsparse_int ldc)

    Sparse matrix dense matrix multiplication using CSR storage format.

- DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2m (aoclsparse_operation trans_A, const aoclsparse_mat_descr descrA, const aoclsparse_matrix csrA, aoclsparse_operation trans_B, const aoclsparse_mat_descr descrB, const aoclsparse_matrix csrB, const aoclsparse_request request, aoclsparse_matrix ∗csrC)

Sparse matrix Sparse matrix multiplication using CSR storage format for single and double precision datatypes.

- DLL_PUBLIC aoclsparse_status aoclsparse_scsr2m (aoclsparse_operation trans_A, const aoclsparse_mat_descr descrA, const aoclsparse_matrix csrA, aoclsparse_operation trans_B, const aoclsparse_mat_descr descrB, const aoclsparse_matrix csrB, const aoclsparse_request request, aoclsparse_matrix *csrC)

  Sparse matrix Sparse matrix multiplication using CSR storage format for single and double precision datatypes.

- DLL_PUBLIC aoclsparse_status aoclsparse_dilu_smoother (aoclsparse_operation op, aoclsparse_matrix A, const aoclsparse_mat_descr descr, double **precond_csr_val, const double *approx_inv_diag, double *x, const double *b)

  Sparse Iterative solver algorithms for single and double precision datatypes.

- DLL_PUBLIC aoclsparse_status aoclsparse_silu_smoother (aoclsparse_operation op, aoclsparse_matrix A, const aoclsparse_mat_descr descr, float **precond_csr_val, const float *approx_inv_diag, float *x, const float *b)

  Sparse Iterative solver algorithms for single and double precision datatypes.

- DLL_PUBLIC aoclsparse_status aoclsparse_sadd (const aoclsparse_operation op, const aoclsparse_matrix A, const float alpha, const aoclsparse_matrix B, aoclsparse_matrix *C)

  Addition of two sparse matrices.

- DLL_PUBLIC aoclsparse_status aoclsparse_dadd (const aoclsparse_operation op, const aoclsparse_matrix A, const double alpha, const aoclsparse_matrix B, aoclsparse_matrix *C)

  Addition of two sparse matrices.

- DLL_PUBLIC aoclsparse_status aoclsparse_cadd (const aoclsparse_operation op, const aoclsparse_matrix A, const aoclsparse_float_complex alpha, const aoclsparse_matrix B, aoclsparse_matrix *C)

  Addition of two sparse matrices.

- DLL_PUBLIC aoclsparse_status aoclsparse_zadd (const aoclsparse_operation op, const aoclsparse_matrix A, const aoclsparse_double_complex alpha, const aoclsparse_matrix B, aoclsparse_matrix *C)

  Addition of two sparse matrices.

3.4.1 Detailed Description

aoclsparse_functions.h provides AMD CPU hardware optimized level 1, 2, and 3 Sparse Linear Algebra Subprograms (Sparse BLAS)

3.4.2 Function Documentation

3.4.2.1 aoclsparse_saxpyi()

```
DLL_PUBLIC aoclsparse_status aoclsparse_saxpyi (
        const aoclsparse_int nnz,
        const float a,
        const float * x,
        const aoclsparse_int * indx,
        float * y )
```

A variant of sparse vector-vector addition between a compressed sparse vector and a dense vector.

`aoclsparse_(s/d/c/z)axpyi` adds a scalar multiple of compressed sparse vector to a dense vector.

Let $y \in C^m$ be a dense vector, x be a compressed sparse vector and I_x be an indices vector of length at least $\mathtt{nnz}$ described by `indx`, then

$$y_{I_{x_i}} = a * x_i + y_{I_{x_i}}, i \in \{1, \ldots, \mathrm{nnz}\}.$$

A possible C implementation could be

```
for(i = 0; i < nnz; ++i)
   y[indx[i]] = a*x[i] + y[indx[i]];
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in	*a*	Scalar value.
in	*x*	Sparse vector stored in compressed form of nnz elements.
in	*indx*	Indices of nnz elements. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of `y`. Array should follow 0-based indexing.
in,out	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ elements.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that provided `nnz` is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative.

3.4.2.2 aoclsparse_daxpyi()

```
DLL_PUBLIC aoclsparse_status aoclsparse_daxpyi (
        const aoclsparse_int nnz,
        const double a,
        const double * x,
        const aoclsparse_int * indx,
        double * y )
```

A variant of sparse vector-vector addition between a compressed sparse vector and a dense vector.

`aoclsparse_(s/d/c/z)axpyi` adds a scalar multiple of compressed sparse vector to a dense vector.

Let $y \in C^m$ be a dense vector, x be a compressed sparse vector and I_x be an indices vector of length at least `nnz` described by `indx`, then

$$y_{I_{x_i}} = a * x_i + y_{I_{x_i}}, i \in \{1, \ldots, \mathrm{nnz}\}.$$

A possible C implementation could be

```
for(i = 0; i < nnz; ++i)
   y[indx[i]] = a*x[i] + y[indx[i]];
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in	*a*	Scalar value.
in	*x*	Sparse vector stored in compressed form of nnz elements.
in	*indx*	Indices of nnz elements. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of `y`. Array should follow 0-based indexing.
in,out	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ elements.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that provided `nnz` is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative.

3.4.2.3 aoclsparse_caxpyi()

```
DLL_PUBLIC aoclsparse_status aoclsparse_caxpyi (
        const aoclsparse_int nnz,
        const void * a,
        const void * x,
        const aoclsparse_int * indx,
        void * y )
```

A variant of sparse vector-vector addition between a compressed sparse vector and a dense vector.

`aoclsparse_(s/d/c/z)axpyi` adds a scalar multiple of compressed sparse vector to a dense vector.

Let $y \in C^m$ be a dense vector, x be a compressed sparse vector and I_x be an indices vector of length at least `nnz` described by `indx`, then

$$y_{I_{x_i}} = a * x_i + y_{I_{x_i}}, i \in \{1, \ldots, \mathrm{nnz}\}.$$

A possible C implementation could be

```
for(i = 0; i < nnz; ++i)
   y[indx[i]] = a*x[i] + y[indx[i]];
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in	*a*	Scalar value.
in	*x*	Sparse vector stored in compressed form of nnz elements.
in	*indx*	Indices of nnz elements. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of `y`. Array should follow 0-based indexing.
in,out	*y*	Array of at least $\max(indx_i, i \in \{1,\ldots,nnz\})$ elements.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that provided `nnz` is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative.

3.4.2.4 aoclsparse_zaxpyi()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zaxpyi (
        const aoclsparse_int nnz,
        const void * a,
        const void * x,
        const aoclsparse_int * indx,
        void * y )
```

A variant of sparse vector-vector addition between a compressed sparse vector and a dense vector.

`aoclsparse_(s/d/c/z)axpyi` adds a scalar multiple of compressed sparse vector to a dense vector.

Let $y \in C^m$ be a dense vector, x be a compressed sparse vector and I_x be an indices vector of length at least `nnz` described by `indx`, then

$$y_{I_{x_i}} = a * x_i + y_{I_{x_i}}, i \in \{1,\ldots,\mathrm{nnz}\}.$$

A possible C implementation could be

```
for(i = 0; i < nnz; ++i)
   y[indx[i]] = a*x[i] + y[indx[i]];
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in	*a*	Scalar value.
in	*x*	Sparse vector stored in compressed form of nnz elements.
in	*indx*	Indices of nnz elements. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of `y`. Array should follow 0-based indexing.
in,out	*y*	Array of at least $\max(indx_i, i \in \{1,\ldots,nnz\})$ elements.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that provided `nnz` is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative.

3.4.2.5 aoclsparse_cdotci()

```
DLL_PUBLIC aoclsparse_status aoclsparse_cdotci (
        const aoclsparse_int nnz,
        const void * x,
        const aoclsparse_int * indx,
        const void * y,
        void * dot )
```

Sparse conjugate dot product for single and double data precision complex types.

`aoclsparse_cdotci` (complex float) and `aoclsparse_zdotci` (complex double) compute the dot product of the conjugate of a complex vector stored in a compressed format and a complex dense vector. Let x and y be respectively a sparse and dense vectors in C^m with `indx` an indices vector of length at least nnz that is used to index into the entries of dense vector y, then these functions return

$$\text{dot} = \sum_{i=0}^{nnz-1} \text{conj}(x_i) * y_{indx_i}.$$

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements (length) of vectors x and $indx$.
in	*x*	Array of at least nnz complex elements.
in	*indx*	Vector of indices of length at least nnz. Each entry of this vector must contain a valid index into y and be unique. The entries of `indx` are not checked for validity.
in	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ complex elements.
out	*dot*	The dot product of conjugate of x and y when $nnz > 0$. If $nnz \leq 0$, `dot` is set to 0.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y`, `dot` is invalid.
aoclsparse_status_invalid_size	Indicates that the provided `nnz` is not positive.

3.4.2.6 aoclsparse_zdotci()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zdotci (
```

```
const aoclsparse_int nnz,
const void * x,
const aoclsparse_int * indx,
const void * y,
void * dot )
```

Sparse conjugate dot product for single and double data precision complex types.

`aoclsparse_cdotci` (complex float) and `aoclsparse_zdotci` (complex double) compute the dot product of the conjugate of a complex vector stored in a compressed format and a complex dense vector. Let x and y be respectively a sparse and dense vectors in C^m with `indx` an indices vector of length at least nnz that is used to index into the entries of dense vector y, then these functions return

$$\text{dot} = \sum_{i=0}^{nnz-1} \text{conj}(x_i) * y_{indx_i}.$$

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements (length) of vectors x and $indx$.
in	*x*	Array of at least nnz complex elements.
in	*indx*	Vector of indices of length at least nnz. Each entry of this vector must contain a valid index into y and be unique. The entries of `indx` are not checked for validity.
in	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ complex elements.
out	*dot*	The dot product of conjugate of x and y when $nnz > 0$. If $nnz \le 0$, `dot` is set to 0.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y`, `dot` is invalid.
aoclsparse_status_invalid_size	Indicates that the provided `nnz` is not positive.

3.4.2.7 aoclsparse_cdotui()

```
DLL_PUBLIC aoclsparse_status aoclsparse_cdotui (
const aoclsparse_int nnz,
const void * x,
const aoclsparse_int * indx,
const void * y,
void * dot )
```

Sparse dot product for single and double data precision complex types.

`aoclsparse_cdotui` (complex float) and `aoclsparse_zdotui` (complex double) compute the dot product of a complex vector stored in a compressed format and a complex dense vector. Let x and y be respectively a sparse and dense vectors in C^m with `indx` an indices vector of length at least nnz that is used to index into the entries of dense vector y, then these functions return

$$\text{dot} = \sum_{i=0}^{nnz-1} x_i * y_{indx_i}.$$

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements (length) of vectors x and $indx$.
in	*x*	Array of at least nnz complex elements.
in	*indx*	Vector of indices of length at least nnz. Each entry of this vector must contain a valid index into y and be unique. The entries of `indx` are not checked for validity.
in	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ complex elements.
out	*dot*	The dot product of x and y when $nnz > 0$. If $nnz \leq 0$, `dot` is set to 0.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y`, `dot` is invalid.
aoclsparse_status_invalid_size	Indicates that the provided `nnz` is not positive.

3.4.2.8 aoclsparse_zdotui()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zdotui (
        const aoclsparse_int nnz,
        const void * x,
        const aoclsparse_int * indx,
        const void * y,
        void * dot )
```

Sparse dot product for single and double data precision complex types.

`aoclsparse_cdotui` (complex float) and `aoclsparse_zdotui` (complex double) compute the dot product of a complex vector stored in a compressed format and a complex dense vector. Let x and y be respectively a sparse and dense vectors in C^m with `indx` an indices vector of length at least nnz that is used to index into the entries of dense vector y, then these functions return

$$\text{dot} = \sum_{i=0}^{nnz-1} x_i * y_{indx_i}.$$

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements (length) of vectors x and $indx$.

Parameters

in	*x*	Array of at least nnz complex elements.
in	*indx*	Vector of indices of length at least nnz. Each entry of this vector must contain a valid index into y and be unique. The entries of `indx` are not checked for validity.
in	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ complex elements.
out	*dot*	The dot product of x and y when $nnz > 0$. If $nnz \leq 0$, `dot` is set to 0.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y`, `dot` is invalid.
aoclsparse_status_invalid_size	Indicates that the provided `nnz` is not positive.

3.4.2.9 aoclsparse_sdoti()

```
DLL_PUBLIC float aoclsparse_sdoti (
        const aoclsparse_int nnz,
        const float * x,
        const aoclsparse_int * indx,
        const float * y )
```

Sparse dot product for single and double data precision real types.

`aoclsparse_sdoti` (float) and `aoclsparse_ddoti` (double) compute the dot product of a real vector stored in a compressed format and a real dense vector. Let x and y be respectively a sparse and dense vectors in R^m with `indx` an indices vector of length at least nnz that is used to index into the entries of dense vector y, then these functions return

$$\text{dot} = \sum_{i=0}^{nnz-1} x_i * y_{indx_i}.$$

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements (length) of vectors x and $indx$.
in	*x*	Array of at least nnz real elements.
in	*indx*	Vector of indices of length at least nnz. Each entry of this vector must contain a valid index into y and be unique. The entries of `indx` are not checked for validity.
in	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ complex elements.

Return values

Float/double	Value of the dot product if `nnz` is positive, otherwise it is set to 0.

3.4.2.10 aoclsparse_ddoti()

```
DLL_PUBLIC double aoclsparse_ddoti (
        const aoclsparse_int nnz,
        const double * x,
        const aoclsparse_int * indx,
        const double * y )
```

Sparse dot product for single and double data precision real types.

`aoclsparse_sdoti` (float) and `aoclsparse_ddoti` (double) compute the dot product of a real vector stored in a compressed format and a real dense vector. Let x and y be respectively a sparse and dense vectors in R^m with `indx` an indices vector of length at least nnz that is used to index into the entries of dense vector y, then these functions return

$$\text{dot} = \sum_{i=0}^{nnz-1} x_i * y_{indx_i}.$$

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements (length) of vectors x and $indx$.
in	*x*	Array of at least nnz real elements.
in	*indx*	Vector of indices of length at least nnz. Each entry of this vector must contain a valid index into y and be unique. The entries of `indx` are not checked for validity.
in	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ complex elements.

Return values

Float/double	Value of the dot product if `nnz` is positive, otherwise it is set to 0.

3.4.2.11 aoclsparse_ssctr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ssctr (
        const aoclsparse_int nnz,
        const float * x,
        const aoclsparse_int * indx,
        float * y )
```

Sparse scatter for single and double precision real and complex types.

`aoclsparse_?sctr` scatter the elements of a compressed sparse vector into a dense vector.

Let $y \in R^m$ (or C^m) be a dense vector, x be a compressed sparse vector and I_x be an indices vector of length at least `nnz` described by `indx`, then

$$y_{I_{x_i}} = x_i, i \in \{1, \ldots, \mathrm{nnz}\}.$$

A possible C implementation for real vectors could be
```
for(i = 0; i < nnz; ++i)
   y[indx[i]] = x[i];
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in	*x*	Array of nnz elements to be scattered.
in	*indx*	Indices of nnz elements to be scattered. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of `y`.
out	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ elements.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that provided `nnz` is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative.

3.4.2.12 aoclsparse_dsctr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dsctr (
        const aoclsparse_int nnz,
        const double * x,
        const aoclsparse_int * indx,
        double * y )
```

Sparse scatter for single and double precision real and complex types.

`aoclsparse_?sctr` scatter the elements of a compressed sparse vector into a dense vector.

Let $y \in R^m$ (or C^m) be a dense vector, x be a compressed sparse vector and I_x be an indices vector of length at least `nnz` described by `indx`, then

$$y_{I_{x_i}} = x_i, i \in \{1, \ldots, \mathrm{nnz}\}.$$

A possible C implementation for real vectors could be
```
for(i = 0; i < nnz; ++i)
   y[indx[i]] = x[i];
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in	*x*	Array of nnz elements to be scattered.
in	*indx*	Indices of nnz elements to be scattered. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of `y`.
out	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ elements.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that provided `nnz` is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative.

3.4.2.13 aoclsparse_csctr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_csctr (
        const aoclsparse_int nnz,
        const void * x,
        const aoclsparse_int * indx,
        void * y )
```

Sparse scatter for single and double precision real and complex types.

`aoclsparse_?sctr` scatter the elements of a compressed sparse vector into a dense vector.

Let $y \in R^m$ (or C^m) be a dense vector, x be a compressed sparse vector and I_x be an indices vector of length at least `nnz` described by `indx`, then

$$y_{I_{x_i}} = x_i, i \in \{1, \ldots, \text{nnz}\}.$$

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
   y[indx[i]] = x[i];
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in	*x*	Array of nnz elements to be scattered.
in	*indx*	Indices of nnz elements to be scattered. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of `y`.
out	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ elements.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that provided `nnz` is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative.

3.4.2.14 aoclsparse_zsctr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zsctr (
    const aoclsparse_int nnz,
    const void * x,
    const aoclsparse_int * indx,
    void * y )
```

Sparse scatter for single and double precision real and complex types.

`aoclsparse_?sctr` scatter the elements of a compressed sparse vector into a dense vector.

Let $y \in R^m$ (or C^m) be a dense vector, x be a compressed sparse vector and I_x be an indices vector of length at least `nnz` described by `indx`, then

$$y_{I_{x_i}} = x_i, i \in \{1, \ldots, \mathrm{nnz}\}.$$

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
   y[indx[i]] = x[i];
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in	*x*	Array of nnz elements to be scattered.
in	*indx*	Indices of nnz elements to be scattered. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of `y`.
out	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ elements.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that provided `nnz` is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative.

3.4.2.15 aoclsparse_ssctrs()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ssctrs (
        const aoclsparse_int nnz,
        const float * x,
        aoclsparse_int stride,
        float * y )
```

Sparse scatter with stride for real/complex single and double data precisions.

`aoclsparse_?sctrs` scatters the elements of a compressed sparse vector into a dense vector using a stride.

Let y be a dense vector of length $n > 0$, x be a compressed sparse vector with $\mathtt{nnz} > 0$ nonzeros, and `stride` be a striding distance, then $y_{\mathtt{stride} \times i} = x_i, \quad i \in \{1, \ldots, \mathtt{nnz}\}$.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
   y[stride * i] = x[i];
```

Note

Contents of the vector `x` are accessed but not checked.

Parameters

in	*nnz*	Number of nonzero elements in x.
in	*x*	Array of `nnz` elements to be scattered into `y`.
in	*stride*	(Positive) striding distance used to store elements in vector `y`.
out	*y*	Array of size at least `stride` × `nnz`.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that one or more of the values provided in `nnz` or `stride` is not positive.

3.4.2.16 aoclsparse_dsctrs()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dsctrs (
        const aoclsparse_int nnz,
        const double * x,
        aoclsparse_int stride,
        double * y )
```

Sparse scatter with stride for real/complex single and double data precisions.

`aoclsparse_?sctrs` scatters the elements of a compressed sparse vector into a dense vector using a stride.

Let y be a dense vector of length $n > 0$, x be a compressed sparse vector with $\mathtt{nnz} > 0$ nonzeros, and `stride` be a striding distance, then $y_{\mathtt{stride} \times i} = x_i, \quad i \in \{1, \ldots, \mathtt{nnz}\}$.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
   y[stride * i] = x[i];
```

Note

Contents of the vector `x` are accessed but not checked.

Parameters

in	*nnz*	Number of nonzero elements in x.
in	*x*	Array of `nnz` elements to be scattered into `y`.
in	*stride*	(Positive) striding distance used to store elements in vector `y`.
out	*y*	Array of size at least `stride` $\times$ `nnz`.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that one or more of the values provided in `nnz` or `stride` is not positive.

3.4.2.17 aoclsparse_csctrs()

```
DLL_PUBLIC aoclsparse_status aoclsparse_csctrs (
        const aoclsparse_int nnz,
        const void * x,
        aoclsparse_int stride,
        void * y )
```

Sparse scatter with stride for real/complex single and double data precisions.

`aoclsparse_?sctrs` scatters the elements of a compressed sparse vector into a dense vector using a stride.

Let y be a dense vector of length $n > 0$, x be a compressed sparse vector with $\mathrm{nnz} > 0$ nonzeros, and `stride` be a striding distance, then $y_{\mathrm{stride} \times i} = x_i, \quad i \in \{1, \ldots, \mathrm{nnz}\}$.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
   y[stride * i] = x[i];
```

Note

Contents of the vector `x` are accessed but not checked.

Parameters

in	*nnz*	Number of nonzero elements in x.
in	*x*	Array of `nnz` elements to be scattered into `y`.
in	*stride*	(Positive) striding distance used to store elements in vector `y`.
out	*y*	Array of size at least `stride` $\times$ `nnz`.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that one or more of the values provided in `nnz` or `stride` is not positive.

3.4.2.18 aoclsparse_zsctrs()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zsctrs (
    const aoclsparse_int nnz,
    const void * x,
    aoclsparse_int stride,
    void * y )
```

Sparse scatter with stride for real/complex single and double data precisions.

`aoclsparse_?sctrs` scatters the elements of a compressed sparse vector into a dense vector using a stride.

Let y be a dense vector of length $n > 0$, x be a compressed sparse vector with $\mathrm{nnz} > 0$ nonzeros, and `stride` be a striding distance, then $y_{\mathtt{stride} \times i} = x_i, \quad i \in \{1, \ldots, \mathrm{nnz}\}$.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
   y[stride * i] = x[i];
```

Note

Contents of the vector `x` are accessed but not checked.

Parameters

in	*nnz*	Number of nonzero elements in x.
in	*x*	Array of `nnz` elements to be scattered into `y`.
in	*stride*	(Positive) striding distance used to store elements in vector `y`.
out	*y*	Array of size at least `stride` × `nnz`.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that one or more of the values provided in `nnz` or `stride` is not positive.

3.4.2.19 aoclsparse_sroti()

```
DLL_PUBLIC aoclsparse_status aoclsparse_sroti (
    const aoclsparse_int nnz,
    float * x,
    const aoclsparse_int * indx,
    float * y,
    const float c,
    const float s )
```

Applies Givens rotations to single and double precision real vectors.

`aoclsparse_sroti` (float) and `aoclsparse_droti` (double) apply the Givens rotations on elements of two real vectors.

Let $y \in R^m$ be a vector in full storage form, x be a vector in a compressed form and I_x be an indices vector of length at least nnz described by indx, then

$$x_i = c * x_i + s * y_{I_{x_i}}$$
$$y_{I_{x_i}} = c * y_{I_{x_i}} - s * x_i$$

where c, s are scalars.

A possible C implementation could be

```
for(i = 0; i < nnz; ++i)
{
   temp = x[i];
   x[i] = c * x[i] + s * y[indx[i]];
   y[indx[i]] = c * y[indx[i]] - s * temp;
}
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in,out	*x*	Array of at least nnz elements in compressed form. The elements of the array are updated after applying Givens rotation.
in	*indx*	Indices of nnz elements used for Givens rotation. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of y and are distinct.
in,out	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ elements in full storage form. The elements of the array are updated after applying Givens rotation.
in	*c*	A scalar.
in	*s*	A scalar.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers x, indx, y is invalid.
aoclsparse_status_invalid_size	Indicates that provided nnz is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative. With this error, the values of vectors x and y are undefined.

3.4.2.20 aoclsparse_droti()

```
DLL_PUBLIC aoclsparse_status aoclsparse_droti (
        const aoclsparse_int nnz,
        double * x,
        const aoclsparse_int * indx,
        double * y,
        const double c,
        const double s )
```

Applies Givens rotations to single and double precision real vectors.

`aoclsparse_sroti` (float) and `aoclsparse_droti` (double) apply the Givens rotations on elements of two real vectors.

Let $y \in R^m$ be a vector in full storage form, x be a vector in a compressed form and I_x be an indices vector of length at least `nnz` described by `indx`, then

$$x_i = c * x_i + s * y_{I_{x_i}}$$
$$y_{I_{x_i}} = c * y_{I_{x_i}} - s * x_i$$

where `c`, `s` are scalars.

A possible C implementation could be

```
for(i = 0; i < nnz; ++i)
{
   temp = x[i];
   x[i] = c * x[i] + s * y[indx[i]];
   y[indx[i]] = c * y[indx[i]] - s * temp;
}
```

Note

The contents of the vectors are not checked for NaNs.

Parameters

in	*nnz*	The number of elements in x and $indx$.
in,out	*x*	Array of at least nnz elements in compressed form. The elements of the array are updated after applying Givens rotation.
in	*indx*	Indices of nnz elements used for Givens rotation. The elements in this vector are only checked for non-negativity. The user should make sure that index is less than the size of `y` and are distinct.
in,out	*y*	Array of at least $\max(indx_i, i \in \{1, \ldots, nnz\})$ elements in full storage form. The elements of the array are updated after applying Givens rotation.
in	*c*	A scalar.
in	*s*	A scalar.

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	At least one of the pointers `x`, `indx`, `y` is invalid.
aoclsparse_status_invalid_size	Indicates that provided `nnz` is less than zero.
aoclsparse_status_invalid_index_value	At least one of the indices in indx is negative. With this error, the values of vectors x and y are undefined.

3.4.2.21 aoclsparse_sgthr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_sgthr (
            aoclsparse_int nnz,
            const float * y,
            float * x,
            const aoclsparse_int * indx )
```

Gather elements from a dense vector and store them into a sparse vector.

The aoclsparse_?gthr is a group of functions that gather the elements indexed in indx from the dense vector y into the sparse vector x.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and I_x be a set of indices of size $0 < \mathrm{nnz} \leq m$ described by indx, then

$$x_i = y_{I_{x_i}}, i \in \{1, \ldots, \mathrm{nnz}\}.$$

For double precision complex vectors use aoclsparse_zgthr and for single precision complex vectors use aoclsparse_cgthr.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
{
   x[i] = y[indx[i]];
}
```

Parameters

in	*nnz*	number of non-zero entries of x. If nnz is zero, then none of the entries of vectors x, y, and indx are touched.
in	*y*	pointer to dense vector y of size at least m.
out	*x*	pointer to sparse vector x with at least nnz non-zero elements.
in	*indx*	index vector of size nnz, containing the indices of the non-zero values of x. Indices should range from 0 to $m-1$, need not be ordered. The elements in this vector are only checked for non-negativity. The user should make sure that no index is out-of-bound and that it does not contains any duplicates.

Return values

aoclsparse_status_success	the operation completed successfully
aoclsparse_status_invalid_size	nnz parameter value is negative
aoclsparse_status_invalid_pointer	at least one of the pointers y, x or indx is invalid
aoclsparse_status_invalid_index_value	at least one of the indices in indx is negative

Note

These functions assume that the indices stored in indx are less than m without duplicate elements, and that x and indx are pointers to vectors of size at least nnz.

3.4.2.22 aoclsparse_dgthr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dgthr (
            aoclsparse_int nnz,
            const double * y,
            double * x,
            const aoclsparse_int * indx )
```

Gather elements from a dense vector and store them into a sparse vector.

The aoclsparse_?gthr is a group of functions that gather the elements indexed in indx from the dense vector y into the sparse vector x.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and I_x be a set of indices of size $0 < \text{nnz} \le m$ described by indx, then

$$x_i = y_{I_{x_i}}, i \in \{1, \ldots, \text{nnz}\}.$$

For double precision complex vectors use `aoclsparse_zgthr` and for single precision complex vectors use `aoclsparse_cgthr`.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
{
    x[i] = y[indx[i]];
}
```

Parameters

in	*nnz*	number of non-zero entries of x. If nnz is zero, then none of the entries of vectors x, y, and indx are touched.
in	*y*	pointer to dense vector y of size at least m.
out	*x*	pointer to sparse vector x with at least nnz non-zero elements.
in	*indx*	index vector of size nnz, containing the indices of the non-zero values of x. Indices should range from 0 to $m - 1$, need not be ordered. The elements in this vector are only checked for non-negativity. The user should make sure that no index is out-of-bound and that it does not contains any duplicates.

Return values

aoclsparse_status_success	the operation completed successfully
aoclsparse_status_invalid_size	nnz parameter value is negative
aoclsparse_status_invalid_pointer	at least one of the pointers y, x or indx is invalid
aoclsparse_status_invalid_index_value	at least one of the indices in indx is negative

Note

These functions assume that the indices stored in indx are less than m without duplicate elements, and that x and indx are pointers to vectors of size at least nnz.

3.4.2.23 aoclsparse_cgthr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_cgthr (
        aoclsparse_int nnz,
        const void * y,
        void * x,
        const aoclsparse_int * indx )
```

Gather elements from a dense vector and store them into a sparse vector.

The `aoclsparse_?gthr` is a group of functions that gather the elements indexed in indx from the dense vector y into the sparse vector x.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and I_x be a set of indices of size $0 < \text{nnz} \le m$ described by indx, then

$$x_i = y_{I_{x_i}}, i \in \{1, \ldots, \text{nnz}\}.$$

For double precision complex vectors use `aoclsparse_zgthr` and for single precision complex vectors use `aoclsparse_cgthr`.

A possible C implementation for real vectors could be
```
for(i = 0; i < nnz; ++i)
{
   x[i] = y[indx[i]];
}
```

Parameters

in	*nnz*	number of non-zero entries of x. If `nnz` is zero, then none of the entries of vectors `x`, `y`, and `indx` are touched.
in	*y*	pointer to dense vector y of size at least m.
out	*x*	pointer to sparse vector x with at least `nnz` non-zero elements.
in	*indx*	index vector of size `nnz`, containing the indices of the non-zero values of x. Indices should range from 0 to $m-1$, need not be ordered. The elements in this vector are only checked for non-negativity. The user should make sure that no index is out-of-bound and that it does not contains any duplicates.

Return values

aoclsparse_status_success	the operation completed successfully
aoclsparse_status_invalid_size	`nnz` parameter value is negative
aoclsparse_status_invalid_pointer	at least one of the pointers `y`, `x` or `indx` is invalid
aoclsparse_status_invalid_index_value	at least one of the indices in `indx` is negative

Note

These functions assume that the indices stored in `indx` are less than m without duplicate elements, and that `x` and `indx` are pointers to vectors of size at least `nnz`.

3.4.2.24 aoclsparse_zgthr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zgthr (
            aoclsparse_int nnz,
            const void * y,
            void * x,
            const aoclsparse_int * indx )
```

Gather elements from a dense vector and store them into a sparse vector.

The `aoclsparse_?gthr` is a group of functions that gather the elements indexed in `indx` from the dense vector `y` into the sparse vector `x`.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and I_x be a set of indices of size $0 < \mathrm{nnz} \le m$ described by `indx`, then

$$x_i = y_{I_{x_i}}, i \in \{1, \ldots, \mathrm{nnz}\}.$$

For double precision complex vectors use `aoclsparse_zgthr` and for single precision complex vectors use `aoclsparse_cgthr`.

A possible C implementation for real vectors could be
```
for(i = 0; i < nnz; ++i)
{
   x[i] = y[indx[i]];
}
```

Parameters

in	*nnz*	number of non-zero entries of x. If nnz is zero, then none of the entries of vectors x, y, and indx are touched.
in	*y*	pointer to dense vector y of size at least m.
out	*x*	pointer to sparse vector x with at least nnz non-zero elements.
in	*indx*	index vector of size nnz, containing the indices of the non-zero values of x. Indices should range from 0 to $m-1$, need not be ordered. The elements in this vector are only checked for non-negativity. The user should make sure that no index is out-of-bound and that it does not contains any duplicates.

Return values

aoclsparse_status_success	the operation completed successfully
aoclsparse_status_invalid_size	nnz parameter value is negative
aoclsparse_status_invalid_pointer	at least one of the pointers y, x or indx is invalid
aoclsparse_status_invalid_index_value	at least one of the indices in indx is negative

Note

These functions assume that the indices stored in indx are less than m without duplicate elements, and that x and indx are pointers to vectors of size at least nnz.

3.4.2.25 aoclsparse_sgthrz()

```
DLL_PUBLIC aoclsparse_status aoclsparse_sgthrz (
    aoclsparse_int nnz,
    float * y,
    float * x,
    const aoclsparse_int * indx )
```

Gather and zero out elements from a dense vector and store them into a sparse vector.

The aoclsparse_?gthrz is a group of functions that gather the elements

indexed in indx from the dense vector y into the sparse vector x. The gathered elements in y are replaced by zero.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and I_x be a set of indices of size $0 < \mathrm{nnz} \le m$ described by indx, then

$$x_i = y_{I_{x_i}}, i \in \{1, \ldots, \mathrm{nnz}\}, \text{ and after the assignment, } y_{I_{x_i}} = 0, i \in \{1, \ldots, \mathrm{nnz}\}.$$

For double precision complex vectors use aoclsparse_zgthrz and for single precision complex vectors use aoclsparse_cgthrz.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
{
   x[i] = y[indx[i]];
   y[indx[i]] = 0;
}
```

Parameters

in	*nnz*	number of non-zero entries of x. If nnz is zero, then none of the entries of vectors x, y, and indx are touched.
in	*y*	pointer to dense vector y of size at least m.
out	*x*	pointer to sparse vector x with at least nnz non-zero elements.
in	*indx*	index vector of size nnz, containing the indices of the non-zero values of x. Indices should range from 0 to $m-1$, need not be ordered. The elements in this vector are only checked for non-negativity. The user should make sure that no index is out-of-bound and that it does not contains any duplicates.

Return values

aoclsparse_status_success	the operation completed successfully
aoclsparse_status_invalid_size	nnz parameter value is negative
aoclsparse_status_invalid_pointer	at least one of the pointers y, x or indx is invalid
aoclsparse_status_invalid_index_value	at least one of the indices in indx is negative

Note

These functions assume that the indices stored in indx are less than m without duplicate elements, and that x and indx are pointers to vectors of size at least nnz.

3.4.2.26 aoclsparse_dgthrz()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dgthrz (
        aoclsparse_int nnz,
        double * y,
        double * x,
        const aoclsparse_int * indx )
```

Gather and zero out elements from a dense vector and store them into a sparse vector.

The aoclsparse_?gthrz is a group of functions that gather the elements

indexed in indx from the dense vector y into the sparse vector x. The gathered elements in y are replaced by zero.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and I_x be a set of indices of size $0 < \mathrm{nnz} \leq m$ described by indx, then

$$x_i = y_{I_{x_i}}, i \in \{1, \ldots, \mathrm{nnz}\}, \text{ and after the assignment, } y_{I_{x_i}} = 0, i \in \{1, \ldots, \mathrm{nnz}\}.$$

For double precision complex vectors use aoclsparse_zgthrz and for single precision complex vectors use aoclsparse_cgthrz.

A possible C implementation for real vectors could be

```c
for(i = 0; i < nnz; ++i)
{
    x[i] = y[indx[i]];
    y[indx[i]] = 0;
}
```

Parameters

in	*nnz*	number of non-zero entries of x. If nnz is zero, then none of the entries of vectors x, y, and indx are touched.
in	*y*	pointer to dense vector y of size at least m.
out	*x*	pointer to sparse vector x with at least nnz non-zero elements.
in	*indx*	index vector of size nnz, containing the indices of the non-zero values of x. Indices should range from 0 to $m-1$, need not be ordered. The elements in this vector are only checked for non-negativity. The user should make sure that no index is out-of-bound and that it does not contains any duplicates.

Return values

aoclsparse_status_success	the operation completed successfully
aoclsparse_status_invalid_size	nnz parameter value is negative
aoclsparse_status_invalid_pointer	at least one of the pointers y, x or indx is invalid
aoclsparse_status_invalid_index_value	at least one of the indices in indx is negative

Note

These functions assume that the indices stored in indx are less than m without duplicate elements, and that x and indx are pointers to vectors of size at least nnz.

3.4.2.27 aoclsparse_cgthrz()

```
DLL_PUBLIC aoclsparse_status aoclsparse_cgthrz (
    aoclsparse_int nnz,
    void * y,
    void * x,
    const aoclsparse_int * indx )
```

Gather and zero out elements from a dense vector and store them into a sparse vector.

The aoclsparse_?gthrz is a group of functions that gather the elements

indexed in indx from the dense vector y into the sparse vector x. The gathered elements in y are replaced by zero.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and I_x be a set of indices of size $0 < \mathrm{nnz} \leq m$ described by indx, then

$$x_i = y_{I_{x_i}}, i \in \{1, \ldots, \mathrm{nnz}\}, \text{ and after the assignment, } y_{I_{x_i}} = 0, i \in \{1, \ldots, \mathrm{nnz}\}.$$

For double precision complex vectors use aoclsparse_zgthrz and for single precision complex vectors use aoclsparse_cgthrz.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
{
   x[i] = y[indx[i]];
   y[indx[i]] = 0;
}
```

Parameters

in	*nnz*	number of non-zero entries of x. If nnz is zero, then none of the entries of vectors x, y, and indx are touched.
in	*y*	pointer to dense vector y of size at least m.
out	*x*	pointer to sparse vector x with at least nnz non-zero elements.
in	*indx*	index vector of size nnz, containing the indices of the non-zero values of x. Indices should range from 0 to $m-1$, need not be ordered. The elements in this vector are only checked for non-negativity. The user should make sure that no index is out-of-bound and that it does not contains any duplicates.

Return values

aoclsparse_status_success	the operation completed successfully
aoclsparse_status_invalid_size	nnz parameter value is negative
aoclsparse_status_invalid_pointer	at least one of the pointers y, x or indx is invalid
aoclsparse_status_invalid_index_value	at least one of the indices in indx is negative

Note

These functions assume that the indices stored in indx are less than m without duplicate elements, and that x and indx are pointers to vectors of size at least nnz.

3.4.2.28 aoclsparse_zgthrz()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zgthrz (
        aoclsparse_int nnz,
        void * y,
        void * x,
        const aoclsparse_int * indx )
```

Gather and zero out elements from a dense vector and store them into a sparse vector.

The aoclsparse_?gthrz is a group of functions that gather the elements

indexed in indx from the dense vector y into the sparse vector x. The gathered elements in y are replaced by zero.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and I_x be a set of indices of size $0 < \mathrm{nnz} \leq m$ described by indx, then

$$x_i = y_{I_{x_i}}, i \in \{1, \ldots, \mathrm{nnz}\}, \text{ and after the assignment, } y_{I_{x_i}} = 0, i \in \{1, \ldots, \mathrm{nnz}\}.$$

For double precision complex vectors use aoclsparse_zgthrz and for single precision complex vectors use aoclsparse_cgthrz.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
{
   x[i] = y[indx[i]];
   y[indx[i]] = 0;
}
```

Parameters

in	*nnz*	number of non-zero entries of x. If `nnz` is zero, then none of the entries of vectors `x`, `y`, and `indx` are touched.
in	*y*	pointer to dense vector y of size at least m.
out	*x*	pointer to sparse vector x with at least `nnz` non-zero elements.
in	*indx*	index vector of size `nnz`, containing the indices of the non-zero values of x. Indices should range from 0 to $m-1$, need not be ordered. The elements in this vector are only checked for non-negativity. The user should make sure that no index is out-of-bound and that it does not contains any duplicates.

Return values

aoclsparse_status_success	the operation completed successfully
aoclsparse_status_invalid_size	`nnz` parameter value is negative
aoclsparse_status_invalid_pointer	at least one of the pointers `y`, `x` or `indx` is invalid
aoclsparse_status_invalid_index_value	at least one of the indices in `indx` is negative

Note

These functions assume that the indices stored in `indx` are less than m without duplicate elements, and that `x` and `indx` are pointers to vectors of size at least `nnz`.

3.4.2.29 aoclsparse_sgthrs()

```
DLL_PUBLIC aoclsparse_status aoclsparse_sgthrs (
        aoclsparse_int nnz,
        const float * y,
        float * x,
        aoclsparse_int stride )
```

Gather elements from a dense vector using a stride and store them into a sparse vector.

The `aoclsparse_?gthrs` is a group of functions that gather the elements from the dense vector `y` using a fixed stride distance and copies them into the sparse vector `x`.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and `stride` be a (positive) striding distance, then $x_i = y_{\mathtt{stride} \times i}, \quad i \in \{1, \ldots, \mathtt{nnz}\}$.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
    x[i] = y[stride *i];
```

Note

These functions are taylored for the case where `stride` is greater than 1. If stride is 1, then it is recommended to use the `aoclsparse_?gthr` set of functions.

Parameters

in	*nnz*	Number of non-zero entries of x. If `nnz` is zero, then none of the entries of vectors `x` and `y` are accessed.
in	*y*	Pointer to dense vector y of size at least m.
out	*x*	Pointer to sparse vector x with at least `nnz` non-zero elements.
in	*stride*	Striding distance used to access elements in the dense vector `y`.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	either `nnz` or the `stride` parameter values are not positive.
aoclsparse_status_invalid_pointer	at least one of the pointers `y`, or `x` is invalid.

3.4.2.30 aoclsparse_dgthrs()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dgthrs (
        aoclsparse_int nnz,
        const double * y,
        double * x,
        aoclsparse_int stride )
```

Gather elements from a dense vector using a stride and store them into a sparse vector.

The `aoclsparse_?gthrs` is a group of functions that gather the elements from the dense vector `y` using a fixed stride distance and copies them into the sparse vector `x`.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and `stride` be a (positive) striding distance, then $x_i = y_{\mathtt{stride} \times i}, \quad i \in \{1, \ldots, \mathtt{nnz}\}$.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
   x[i] = y[stride *i];
```

Note

These functions are taylored for the case where `stride` is greater than 1. If stride is 1, then it is recommended to use the `aoclsparse_?gthr` set of functions.

Parameters

in	*nnz*	Number of non-zero entries of x. If `nnz` is zero, then none of the entries of vectors `x` and `y` are accessed.
in	*y*	Pointer to dense vector y of size at least m.
out	*x*	Pointer to sparse vector x with at least `nnz` non-zero elements.
in	*stride*	Striding distance used to access elements in the dense vector `y`.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	either `nnz` or the `stride` parameter values are not positive.
aoclsparse_status_invalid_pointer	at least one of the pointers `y`, or `x` is invalid.

3.4.2.31 aoclsparse_cgthrs()

```
DLL_PUBLIC aoclsparse_status aoclsparse_cgthrs (
        aoclsparse_int nnz,
        const void * y,
```

```
void * x,
aoclsparse_int stride )
```

Gather elements from a dense vector using a stride and store them into a sparse vector.

The `aoclsparse_?gthrs` is a group of functions that gather the elements from the dense vector `y` using a fixed stride distance and copies them into the sparse vector `x`.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and `stride` be a (positive) striding distance, then $x_i = y_{\mathtt{stride} \times i}, \quad i \in \{1, \ldots, \mathtt{nnz}\}$.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
  x[i] = y[stride *i];
```

Note

These functions are taylored for the case where `stride` is greater than 1. If stride is 1, then it is recommended to use the `aoclsparse_?gthr` set of functions.

Parameters

in	*nnz*	Number of non-zero entries of x. If `nnz` is zero, then none of the entries of vectors `x` and `y` are accessed.
in	*y*	Pointer to dense vector y of size at least m.
out	*x*	Pointer to sparse vector x with at least `nnz` non-zero elements.
in	*stride*	Striding distance used to access elements in the dense vector `y`.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	either `nnz` or the `stride` parameter values are not positive.
aoclsparse_status_invalid_pointer	at least one of the pointers `y`, or `x` is invalid.

3.4.2.32 aoclsparse_zgthrs()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zgthrs (
aoclsparse_int nnz,
const void * y,
void * x,
aoclsparse_int stride )
```

Gather elements from a dense vector using a stride and store them into a sparse vector.

The `aoclsparse_?gthrs` is a group of functions that gather the elements from the dense vector `y` using a fixed stride distance and copies them into the sparse vector `x`.

Let $y \in R^m$ (or C^m) be a dense vector, x be a sparse vector from the same space and `stride` be a (positive) striding distance, then $x_i = y_{\mathtt{stride} \times i}, \quad i \in \{1, \ldots, \mathtt{nnz}\}$.

A possible C implementation for real vectors could be

```
for(i = 0; i < nnz; ++i)
  x[i] = y[stride *i];
```

Note

These functions are taylored for the case where `stride` is greater than 1. If stride is 1, then it is recommended to use the `aoclsparse_?gthr` set of functions.

Parameters

in	*nnz*	Number of non-zero entries of x. If nnz is zero, then none of the entries of vectors x and y are accessed.
in	*y*	Pointer to dense vector y of size at least m.
out	*x*	Pointer to sparse vector x with at least nnz non-zero elements.
in	*stride*	Striding distance used to access elements in the dense vector y.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	either nnz or the stride parameter values are not positive.
aoclsparse_status_invalid_pointer	at least one of the pointers y, or x is invalid.

3.4.2.33 aoclsparse_scsrmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsrmv (
    aoclsparse_operation trans,
    const float * alpha,
    aoclsparse_int m,
    aoclsparse_int n,
    aoclsparse_int nnz,
    const float * csr_val,
    const aoclsparse_int * csr_col_ind,
    const aoclsparse_int * csr_row_ptr,
    const aoclsparse_mat_descr descr,
    const float * x,
    const float * beta,
    float * y )
```

Single and double precision sparse matrix vector multiplication using CSR storage format.

aoclsparse_csrmv multiplies the scalar α with a sparse $m \times n$ matrix, defined in CSR storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if trans = aoclsparse\_operation\_none} \\ A^T, & \text{if trans = aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans = aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Parameters

in	*trans*	matrix operation type.
in	*alpha*	scalar α.
in	*m*	number of rows of the sparse CSR matrix.
in	*n*	number of columns of the sparse CSR matrix.
in	*nnz*	number of non-zero entries of the sparse CSR matrix.
in	*csr_val*	array of nnz elements of the sparse CSR matrix.
in	*csr_col_ind*	array of nnz elements containing the column indices of the sparse CSR matrix.
in	*csr_row_ptr*	array of m+1 elements that point to the start of every row of the sparse CSR matrix.

Parameters

in	*descr*	descriptor of the sparse CSR matrix. Currently, only aoclsparse_matrix_type_general and aoclsparse_matrix_type_symmetric is supported.
in	*x*	array of n elements ($op(A) = A$) or m elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	scalar β.
in,out	*y*	array of m elements ($op(A) = A$) or n elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	m, n or nnz is invalid.
aoclsparse_status_invalid_pointer	descr, alpha, csr_val, csr_row_ptr, csr_col_ind, x, beta or y pointer is invalid.
aoclsparse_status_not_implemented	trans is not aoclsparse_operation_none and trans is not aoclsparse_operation_transpose. aoclsparse_matrix_type is not aoclsparse_matrix_type_general, or aoclsparse_matrix_type is not aoclsparse_matrix_type_symmetric.

Example

This example performs a sparse matrix vector multiplication in CSR format using additional meta data to improve performance.

3.4.2.34 aoclsparse_dcsrmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsrmv (
    aoclsparse_operation trans,
    const double * alpha,
    aoclsparse_int m,
    aoclsparse_int n,
    aoclsparse_int nnz,
    const double * csr_val,
    const aoclsparse_int * csr_col_ind,
    const aoclsparse_int * csr_row_ptr,
    const aoclsparse_mat_descr descr,
    const double * x,
    const double * beta,
    double * y )
```

Single and double precision sparse matrix vector multiplication using CSR storage format.

aoclsparse_csrmv multiplies the scalar α with a sparse $m \times n$ matrix, defined in CSR storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if trans = aoclsparse\_operation\_none} \\ A^T, & \text{if trans = aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans = aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Parameters

`in`	*trans*	matrix operation type.
`in`	*alpha*	scalar α.
`in`	*m*	number of rows of the sparse CSR matrix.
`in`	*n*	number of columns of the sparse CSR matrix.
`in`	*nnz*	number of non-zero entries of the sparse CSR matrix.
`in`	*csr_val*	array of `nnz` elements of the sparse CSR matrix.
`in`	*csr_col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
`in`	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
`in`	*descr*	descriptor of the sparse CSR matrix. Currently, only aoclsparse_matrix_type_general and aoclsparse_matrix_type_symmetric is supported.
`in`	*x*	array of `n` elements ($op(A) = A$) or `m` elements ($op(A) = A^T$ or $op(A) = A^H$).
`in`	*beta*	scalar β.
`in,out`	*y*	array of `m` elements ($op(A) = A$) or `n` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `csr_val`, `csr_row_ptr`, `csr_col_ind`, `x`, `beta` or `y` pointer is invalid.
aoclsparse_status_not_implemented	`trans` is not aoclsparse_operation_none and `trans` is not aoclsparse_operation_transpose. aoclsparse_matrix_type is not aoclsparse_matrix_type_general, or aoclsparse_matrix_type is not aoclsparse_matrix_type_symmetric.

Example

This example performs a sparse matrix vector multiplication in CSR format using additional meta data to improve performance.

3.4.2.35 aoclsparse_sellmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_sellmv (
    aoclsparse_operation trans,
    const float * alpha,
    aoclsparse_int m,
    aoclsparse_int n,
    aoclsparse_int nnz,
    const float * ell_val,
    const aoclsparse_int * ell_col_ind,
    aoclsparse_int ell_width,
    const aoclsparse_mat_descr descr,
    const float * x,
    const float * beta,
    float * y )
```

Single & Double precision sparse matrix vector multiplication using ELL storage format.

`aoclsparse_ellmv` multiplies the scalar α with a sparse $m \times n$ matrix, defined in ELL storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if trans = aoclsparse\_operation\_none} \\ A^T, & \text{if trans = aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans = aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

Currently, only `trans` = aoclsparse_operation_none is supported.

Parameters

in	*trans*	matrix operation type.
in	*alpha*	scalar α.
in	*m*	number of rows of the sparse ELL matrix.
in	*n*	number of columns of the sparse ELL matrix.
in	*nnz*	number of non-zero entries of the sparse ELL matrix.
in	*descr*	descriptor of the sparse ELL matrix. Both, base-zero and base-one input arrays of ELL matrix are supported
in	*ell_val*	array that contains the elements of the sparse ELL matrix. Padded elements should be zero.
in	*ell_col_ind*	array that contains the column indices of the sparse ELL matrix. Padded column indices should be -1.
in	*ell_width*	number of non-zero elements per row of the sparse ELL matrix.
in	*x*	array of `n` elements ($op(A) = A$) or `m` elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	scalar β.
in,out	*y*	array of `m` elements ($op(A) = A$) or `n` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `ell_width` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `ell_val`, `ell_col_ind`, `x`, `beta` or `y` pointer is invalid.
aoclsparse_status_not_implemented	`trans` != aoclsparse_operation_none or aoclsparse_matrix_type != aoclsparse_matrix_type_general.

3.4.2.36 aoclsparse_dellmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dellmv (
        aoclsparse_operation trans,
        const double * alpha,
        aoclsparse_int m,
        aoclsparse_int n,
        aoclsparse_int nnz,
        const double * ell_val,
        const aoclsparse_int * ell_col_ind,
```

```
aoclsparse_int ell_width,
const aoclsparse_mat_descr descr,
const double * x,
const double * beta,
double * y )
```

Single & Double precision sparse matrix vector multiplication using ELL storage format.

`aoclsparse_ellmv` multiplies the scalar α with a sparse $m \times n$ matrix, defined in ELL storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if trans = aoclsparse\_operation\_none} \\ A^T, & \text{if trans = aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans = aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

Currently, only `trans` = aoclsparse_operation_none is supported.

Parameters

in	*trans*	matrix operation type.
in	*alpha*	scalar α.
in	*m*	number of rows of the sparse ELL matrix.
in	*n*	number of columns of the sparse ELL matrix.
in	*nnz*	number of non-zero entries of the sparse ELL matrix.
in	*descr*	descriptor of the sparse ELL matrix. Both, base-zero and base-one input arrays of ELL matrix are supported
in	*ell_val*	array that contains the elements of the sparse ELL matrix. Padded elements should be zero.
in	*ell_col_ind*	array that contains the column indices of the sparse ELL matrix. Padded column indices should be -1.
in	*ell_width*	number of non-zero elements per row of the sparse ELL matrix.
in	*x*	array of `n` elements ($op(A) = A$) or `m` elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	scalar β.
in,out	*y*	array of `m` elements ($op(A) = A$) or `n` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `ell_width` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `ell_val`, `ell_col_ind`, `x`, `beta` or `y` pointer is invalid.
aoclsparse_status_not_implemented	`trans` != aoclsparse_operation_none or aoclsparse_matrix_type != aoclsparse_matrix_type_general.

3.4.2.37 aoclsparse_sdiamv() `DLL_PUBLIC aoclsparse_status aoclsparse_sdiamv (`

```
aoclsparse_operation trans,
const float * alpha,
aoclsparse_int m,
aoclsparse_int n,
aoclsparse_int nnz,
const float * dia_val,
const aoclsparse_int * dia_offset,
aoclsparse_int dia_num_diag,
const aoclsparse_mat_descr descr,
const float * x,
const float * beta,
float * y )
```

Single & Double precision sparse matrix vector multiplication using DIA storage format.

`aoclsparse_diamv` multiplies the scalar α with a sparse $m \times n$ matrix, defined in DIA storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

Currently, only `trans` = aoclsparse_operation_none is supported.

Parameters

in	*trans*	matrix operation type.
in	*alpha*	scalar α.
in	*m*	number of rows of the sparse DIA matrix.
in	*n*	number of columns of the sparse DIA matrix.
in	*nnz*	number of non-zero entries of the sparse DIA matrix.
in	*descr*	descriptor of the sparse DIA matrix.
in	*dia_val*	array that contains the elements of the sparse DIA matrix. Padded elements should be zero.
in	*dia_offset*	array that contains the offsets of each diagonal of the sparse DIAL matrix.
in	*dia_num_diag*	number of diagonals in the sparse DIA matrix.
in	*x*	array of `n` elements ($op(A) = A$) or `m` elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	scalar β.
in,out	*y*	array of `m` elements ($op(A) = A$) or `n` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `ell_width` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `ell_val`, `ell_col_ind`, `x`, `beta` or `y` pointer is invalid.

Return values

aoclsparse_status_not_implemented	`trans` != aoclsparse_operation_none or aoclsparse_matrix_type != aoclsparse_matrix_type_general.

3.4.2.38 aoclsparse_ddiamv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ddiamv (
    aoclsparse_operation trans,
    const double * alpha,
    aoclsparse_int m,
    aoclsparse_int n,
    aoclsparse_int nnz,
    const double * dia_val,
    const aoclsparse_int * dia_offset,
    aoclsparse_int dia_num_diag,
    const aoclsparse_mat_descr descr,
    const double * x,
    const double * beta,
    double * y )
```

Single & Double precision sparse matrix vector multiplication using DIA storage format.

`aoclsparse_diamv` multiplies the scalar α with a sparse $m \times n$ matrix, defined in DIA storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

Currently, only `trans` = aoclsparse_operation_none is supported.

Parameters

`in`	*trans*	matrix operation type.
`in`	*alpha*	scalar α.
`in`	*m*	number of rows of the sparse DIA matrix.
`in`	*n*	number of columns of the sparse DIA matrix.
`in`	*nnz*	number of non-zero entries of the sparse DIA matrix.
`in`	*descr*	descriptor of the sparse DIA matrix.
`in`	*dia_val*	array that contains the elements of the sparse DIA matrix. Padded elements should be zero.
`in`	*dia_offset*	array that contains the offsets of each diagonal of the sparse DIAL matrix.
`in`	*dia_num_diag*	number of diagonals in the sparse DIA matrix.
`in`	*x*	array of `n` elements ($op(A) = A$) or `m` elements ($op(A) = A^T$ or $op(A) = A^H$).
`in`	*beta*	scalar β.
`in,out`	*y*	array of `m` elements ($op(A) = A$) or `n` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `ell_width` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `ell_val`, `ell_col_ind`, `x`, `beta` or `y` pointer is invalid.
aoclsparse_status_not_implemented	`trans` != aoclsparse_operation_none or aoclsparse_matrix_type != aoclsparse_matrix_type_general.

3.4.2.39 aoclsparse_sbsrmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_sbsrmv (
        aoclsparse_operation trans,
        const float * alpha,
        aoclsparse_int mb,
        aoclsparse_int nb,
        aoclsparse_int bsr_dim,
        const float * bsr_val,
        const aoclsparse_int * bsr_col_ind,
        const aoclsparse_int * bsr_row_ptr,
        const aoclsparse_mat_descr descr,
        const float * x,
        const float * beta,
        float * y )
```

Single & Double precision Sparse matrix vector multiplication using BSR storage format.

`aoclsparse_bsrmv` multiplies the scalar α with a sparse $(mb \cdot \text{bsr\_dim}) \times (nb \cdot \text{bsr\_dim})$ matrix, defined in BSR storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if trans = aoclsparse\_operation\_none} \\ A^T, & \text{if trans = aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans = aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

Currently, only `trans` = aoclsparse_operation_none is supported.

Parameters

in	*trans*	matrix operation type.
in	*mb*	number of block rows of the sparse BSR matrix.
in	*nb*	number of block columns of the sparse BSR matrix.
in	*alpha*	scalar α.
in	*descr*	descriptor of the sparse BSR matrix. Both, base-zero and base-one input arrays of BSR matrix are supported
in	*bsr_val*	array of `nnzb` blocks of the sparse BSR matrix.
in	*bsr_row_ptr*	array of `mb+1` elements that point to the start of every block row of the sparse BSR matrix.
in	*bsr_col_ind*	array of `nnz` containing the block column indices of the sparse BSR matrix.

Parameters

in	*bsr_dim*	block dimension of the sparse BSR matrix.
in	*x*	array of `nb*bsr_dim` elements ($op(A) = A$) or `mb*bsr_dim` elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	scalar β.
in,out	*y*	array of `mb*bsr_dim` elements ($op(A) = A$) or `nb*bsr_dim` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_handle	the library context was not initialized.
aoclsparse_status_invalid_size	`mb`, `nb`, `nnzb` or `bsr_dim` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `bsr_val`, `bsr_row_ind`, `bsr_col_ind`, `x`, `beta` or `y` pointer is invalid.
aoclsparse_status_arch_mismatch	the device is not supported.
aoclsparse_status_not_implemented	`trans` != aoclsparse_operation_none or aoclsparse_matrix_type != aoclsparse_matrix_type_general.

3.4.2.40 aoclsparse_dbsrmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dbsrmv (
    aoclsparse_operation trans,
    const double * alpha,
    aoclsparse_int mb,
    aoclsparse_int nb,
    aoclsparse_int bsr_dim,
    const double * bsr_val,
    const aoclsparse_int * bsr_col_ind,
    const aoclsparse_int * bsr_row_ptr,
    const aoclsparse_mat_descr descr,
    const double * x,
    const double * beta,
    double * y )
```

Single & Double precision Sparse matrix vector multiplication using BSR storage format.

`aoclsparse_bsrmv` multiplies the scalar α with a sparse $(mb \cdot \text{bsr\_dim}) \times (nb \cdot \text{bsr\_dim})$ matrix, defined in BSR storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if trans = aoclsparse\_operation\_none} \\ A^T, & \text{if trans = aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans = aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

Currently, only `trans` = aoclsparse_operation_none is supported.

Parameters

in	*trans*	matrix operation type.
in	*mb*	number of block rows of the sparse BSR matrix.
in	*nb*	number of block columns of the sparse BSR matrix.
in	*alpha*	scalar α.
in	*descr*	descriptor of the sparse BSR matrix. Both, base-zero and base-one input arrays of BSR matrix are supported
in	*bsr_val*	array of `nnzb` blocks of the sparse BSR matrix.
in	*bsr_row_ptr*	array of `mb+1` elements that point to the start of every block row of the sparse BSR matrix.
in	*bsr_col_ind*	array of `nnz` containing the block column indices of the sparse BSR matrix.
in	*bsr_dim*	block dimension of the sparse BSR matrix.
in	*x*	array of `nb*bsr_dim` elements ($op(A) = A$) or `mb*bsr_dim` elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	scalar β.
in,out	*y*	array of `mb*bsr_dim` elements ($op(A) = A$) or `nb*bsr_dim` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_handle	the library context was not initialized.
aoclsparse_status_invalid_size	`mb`, `nb`, `nnzb` or `bsr_dim` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `bsr_val`, `bsr_row_ind`, `bsr_col_ind`, `x`, `beta` or `y` pointer is invalid.
aoclsparse_status_arch_mismatch	the device is not supported.
aoclsparse_status_not_implemented	`trans` != aoclsparse_operation_none or aoclsparse_matrix_type != aoclsparse_matrix_type_general.

3.4.2.41 aoclsparse_smv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_smv (
        aoclsparse_operation op,
        const float * alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        const float * x,
        const float * beta,
        float * y )
```

Computes sparse matrix vector multiplication for real/complex single and double data precisions.

`aoclsparse_`(s/d/c/z)mv performs a sparse matrix vector multiplication such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

where, x and y are dense vectors, alpha and beta are scalars, and A is a sparse matrix structure. The matrix operation 'op' is defined as:

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

Parameters

in	*op*	Matrix operation.
in	*alpha*	Scalar α.
in	*A*	The sparse matrix structure containing a sparse matrix of dimension ($m \cdot n$) that is created using aoclsparse_create_?csr.
in	*descr*	Descriptor of the sparse matrix can be one of the following: aoclsparse_matrix_type_general, aoclsparse_matrix_type_triangular, aoclsparse_matrix_type_symmetric, and aoclsparse_matrix_type_hermitian. Both base-zero and base-one are supported, however, the index base needs to match the one used at when aoclsparse_matrix was created.
in	*x*	An array of n elements ($op(A) = A$) or m elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	Scalar β.
in,out	*y*	An array of m elements ($op(A) = A$) or n elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_size	The value of m, n or nnz is invalid.
aoclsparse_status_invalid_pointer	descr, alpha, internal structures related to the sparse matrix A, x, beta or y has an invalid pointer.
aoclsparse_status_not_implemented	The requested functionality is not implemented.

3.4.2.42 aoclsparse_dmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dmv (
    aoclsparse_operation op,
    const double * alpha,
    aoclsparse_matrix A,
    const aoclsparse_mat_descr descr,
    const double * x,
    const double * beta,
    double * y )
```

Computes sparse matrix vector multiplication for real/complex single and double data precisions.

aoclsparse_(s/d/c/z)mv performs a sparse matrix vector multiplication such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

where, x and y are dense vectors, alpha and beta are scalars, and A is a sparse matrix structure. The matrix operation 'op' is defined as:

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

Parameters

in	*op*	Matrix operation.
in	*alpha*	Scalar α.
in	*A*	The sparse matrix structure containing a sparse matrix of dimension ($m \cdot n$) that is created using aoclsparse_create_?csr.
in	*descr*	Descriptor of the sparse matrix can be one of the following: aoclsparse_matrix_type_general, aoclsparse_matrix_type_triangular, aoclsparse_matrix_type_symmetric, and aoclsparse_matrix_type_hermitian. Both base-zero and base-one are supported, however, the index base needs to match the one used at when aoclsparse_matrix was created.
in	*x*	An array of `n` elements ($op(A) = A$) or `m` elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	Scalar β.
in,out	*y*	An array of `m` elements ($op(A) = A$) or `n` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_size	The value of `m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, internal structures related to the sparse matrix `A`, `x`, `beta` or `y` has an invalid pointer.
aoclsparse_status_not_implemented	The requested functionality is not implemented.

3.4.2.43 aoclsparse_cmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_cmv (
    aoclsparse_operation op,
    const aoclsparse_float_complex * alpha,
    aoclsparse_matrix A,
    const aoclsparse_mat_descr descr,
    const aoclsparse_float_complex * x,
    const aoclsparse_float_complex * beta,
    aoclsparse_float_complex * y )
```

Computes sparse matrix vector multiplication for real/complex single and double data precisions.

`aoclsparse_`(s/d/c/z)mv performs a sparse matrix vector multiplication such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

where, x and y are dense vectors, alpha and beta are scalars, and A is a sparse matrix structure. The matrix operation 'op' is defined as:

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

Parameters

in	*op*	Matrix operation.
in	*alpha*	Scalar α.
in	*A*	The sparse matrix structure containing a sparse matrix of dimension ($m \cdot n$) that is created using aoclsparse_create_?csr.
in	*descr*	Descriptor of the sparse matrix can be one of the following: aoclsparse_matrix_type_general, aoclsparse_matrix_type_triangular, aoclsparse_matrix_type_symmetric, and aoclsparse_matrix_type_hermitian. Both base-zero and base-one are supported, however, the index base needs to match the one used at when aoclsparse_matrix was created.
in	*x*	An array of `n` elements ($op(A) = A$) or `m` elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	Scalar β.
in,out	*y*	An array of `m` elements ($op(A) = A$) or `n` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_size	The value of `m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, internal structures related to the sparse matrix `A`, `x`, `beta` or `y` has an invalid pointer.
aoclsparse_status_not_implemented	The requested functionality is not implemented.

3.4.2.44 aoclsparse_zmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zmv (
        aoclsparse_operation op,
        const aoclsparse_double_complex * alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        const aoclsparse_double_complex * x,
        const aoclsparse_double_complex * beta,
        aoclsparse_double_complex * y )
```

Computes sparse matrix vector multiplication for real/complex single and double data precisions.

`aoclsparse_`(s/d/c/z)mv performs a sparse matrix vector multiplication such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

where, x and y are dense vectors, alpha and beta are scalars, and A is a sparse matrix structure. The matrix operation 'op' is defined as:

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

Parameters

in	*op*	Matrix operation.
in	*alpha*	Scalar α.
in	*A*	The sparse matrix structure containing a sparse matrix of dimension ($m \cdot n$) that is created using aoclsparse_create_?csr.
in	*descr*	Descriptor of the sparse matrix can be one of the following: aoclsparse_matrix_type_general, aoclsparse_matrix_type_triangular, aoclsparse_matrix_type_symmetric, and aoclsparse_matrix_type_hermitian. Both base-zero and base-one are supported, however, the index base needs to match the one used at when aoclsparse_matrix was created.
in	*x*	An array of `n` elements ($op(A) = A$) or `m` elements ($op(A) = A^T$ or $op(A) = A^H$).
in	*beta*	Scalar β.
in,out	*y*	An array of `m` elements ($op(A) = A$) or `n` elements ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_size	The value of `m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, internal structures related to the sparse matrix `A`, `x`, `beta` or `y` has an invalid pointer.
aoclsparse_status_not_implemented	The requested functionality is not implemented.

3.4.2.45 aoclsparse_scsrsv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsrsv (
        aoclsparse_operation trans,
        const float * alpha,
        aoclsparse_int m,
        const float * csr_val,
        const aoclsparse_int * csr_col_ind,
        const aoclsparse_int * csr_row_ptr,
        const aoclsparse_mat_descr descr,
        const float * x,
        float * y )
```

Sparse triangular solve using CSR storage format for single and double data precisions.

`aoclsparse_`?srsv solves a sparse triangular linear system of a sparse $m \times m$ matrix, defined in CSR storage format, a dense solution vector y and the right-hand side x that is multiplied by α, such that

$$op(A) \cdot y = \alpha \cdot x,$$

with

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

Currently, only `trans` = aoclsparse_operation_none is supported.

The input matrix has to be sparse upper or lower triangular matrix with unit or non-unit main diagonal. Matrix has to be sorted. No diagonal element can be omitted from a sparse storage if the solver is called with the non-unit indicator.

Parameters

in	*trans*	matrix operation type.
in	*alpha*	scalar α.
in	*m*	number of rows of the sparse CSR matrix.
in	*csr_val*	array of `nnz` elements of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
in	*descr*	descriptor of the sparse CSR matrix.
in	*x*	array of `m` elements, holding the right-hand side.
out	*y*	array of `m` elements, holding the solution.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `csr_val`, `csr_row_ptr`, `csr_col_ind`, `x` or `y` pointer is invalid.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	`trans` = aoclsparse_operation_conjugate_transpose or `trans` = aoclsparse_operation_transpose or aoclsparse_matrix_type is not aoclsparse_matrix_type_general.

3.4.2.46 aoclsparse_dcsrsv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsrsv (
    aoclsparse_operation trans,
    const double * alpha,
    aoclsparse_int m,
    const double * csr_val,
    const aoclsparse_int * csr_col_ind,
    const aoclsparse_int * csr_row_ptr,
    const aoclsparse_mat_descr descr,
    const double * x,
    double * y )
```

Sparse triangular solve using CSR storage format for single and double data precisions.

`aoclsparse_?srsv` solves a sparse triangular linear system of a sparse $m \times m$ matrix, defined in CSR storage format, a dense solution vector y and the right-hand side x that is multiplied by α, such that

$$op(A) \cdot y = \alpha \cdot x,$$

with

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

Currently, only `trans` = aoclsparse_operation_none is supported.

The input matrix has to be sparse upper or lower triangular matrix with unit or non-unit main diagonal. Matrix has to be sorted. No diagonal element can be omitted from a sparse storage if the solver is called with the non-unit indicator.

Parameters

in	*trans*	matrix operation type.
in	*alpha*	scalar α.
in	*m*	number of rows of the sparse CSR matrix.
in	*csr_val*	array of `nnz` elements of the sparse CSR matrix.
in	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
in	*csr_col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
in	*descr*	descriptor of the sparse CSR matrix.
in	*x*	array of `m` elements, holding the right-hand side.
out	*y*	array of `m` elements, holding the solution.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `csr_val`, `csr_row_ptr`, `csr_col_ind`, `x` or `y` pointer is invalid.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	`trans` = aoclsparse_operation_conjugate_transpose or `trans` = aoclsparse_operation_transpose or aoclsparse_matrix_type is not aoclsparse_matrix_type_general.

3.4.2.47 aoclsparse_strsv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_strsv (
        aoclsparse_operation trans,
        const float alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        const float * b,
        float * x )
```

Sparse triangular solver for real/complex single and double data precisions.

The functions `aoclsparse_?trsv` solve sparse lower (or upper) triangular linear system of equations. The system is defined by the sparse $m \times m$ matrix A, the dense solution m-vector x, and the right-hand side dense m-vector b. Vector b is multiplied by α. The solution x is estimated by solving

$$op(L) \cdot x = \alpha \cdot b, \quad \text{or} \quad op(U) \cdot x = \alpha \cdot b,$$

where $L = \text{tril}(A)$ is the lower triangle of matrix A, similarly, $U = \text{triu}(A)$ is the upper triangle of matrix A. The operator $op()$ is regarded as the matrix linear operation,

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

If the matrix descriptor `descr` specifies that the matrix A is to be regarded has having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are all considered to be unitary.

The input matrix need not be (upper or lower) triangular matrix, in the `descr`, the `fill_mode` entity specifies which triangle to consider, namely, if `fill_mode` = aoclsparse_fill_mode_lower, then

$$op(L) \cdot x = \alpha \cdot b,$$

otherwise, if `fill_mode` = aoclsparse_fill_mode_upper, then

$$op(U) \cdot x = \alpha \cdot b$$

is solved.

To increase performance and if the matrix A is to be used more than once to solve for different right-hand sides b's, then it is encouraged to provide hints using `aoclsparse_set_sv_hint` and `aoclsparse↩ _optimize`, otherwise, the optimization for the matrix will be done by the solver on entry.

There is a `_kid` (Kernel ID) variation of TRSV, namely with a suffix of `_kid`, this solver allows to choose which TRSV kernel to use (if possible). Currently the possible choices are: `kid=0` Reference implementation (No explicit AVX instructions). `kid=1` Reference AVX 256-bit implementation only for double data precision and for operations aoclsparse_operation_none and aoclsparse_operation_transpose. `kid=2` Kernel Template version using AVX/AVX2 extensions. `kid=3` Kernel Template version using AVX512F+ CPU extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

`in`	*trans*	matrix operation type, either aoclsparse_operation_none, aoclsparse_operation_transpose, or aoclsparse_operation_conjugate_transpose.
`in`	*alpha*	scalar α, used to premultiply right-hand side vector b.
`in,out`	*A*	matrix data. `A` is modified only if solver requires to optimize matrix data.
`in`	*descr*	matrix descriptor. Supported matrix types are aoclsparse_matrix_type_symmetric and aoclsparse_matrix_type_triangular.
`in`	*b*	array of `m` elements, storing the right-hand side.
`out`	*x*	array of `m` elements, storing the solution if solver returns aoclsparse_status_success.
`in`	*kid*	Kernel ID, hints a request on which TRSV kernel to use.

Return values

aoclsparse_status_success	the operation completed successfully and x contains the solution to the linear system of equations.
aoclsparse_status_invalid_size	matrix A or $op(A)$ is invalid.
aoclsparse_status_invalid_pointer	One or more of `A`, `descr`, `x`, `b` are invalid pointers.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	the requested operation is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.48 aoclsparse_dtrsv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dtrsv (
        aoclsparse_operation trans,
        const double alpha,
```

```
aoclsparse_matrix A,
const aoclsparse_mat_descr descr,
const double * b,
double * x )
```

Sparse triangular solver for real/complex single and double data precisions.

The functions `aoclsparse_?trsv` solve sparse lower (or upper) triangular linear system of equations. The system is defined by the sparse $m \times m$ matrix A, the dense solution m-vector x, and the right-hand side dense m-vector b. Vector b is multiplied by α. The solution x is estimated by solving

$$op(L) \cdot x = \alpha \cdot b, \quad \text{or} \quad op(U) \cdot x = \alpha \cdot b,$$

where $L = \text{tril}(A)$ is the lower triangle of matrix A, similarly, $U = \text{triu}(A)$ is the upper triangle of matrix A. The operator $op()$ is regarded as the matrix linear operation,

$$op(A) = \begin{cases} A, & \text{if trans = aoclsparse\_operation\_none} \\ A^T, & \text{if trans = aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans = aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

If the matrix descriptor `descr` specifies that the matrix A is to be regarded has having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are all considered to be unitary.

The input matrix need not be (upper or lower) triangular matrix, in the `descr`, the `fill_mode` entity specifies which triangle to consider, namely, if `fill_mode` = aoclsparse_fill_mode_lower, then

$$op(L) \cdot x = \alpha \cdot b,$$

otherwise, if `fill_mode` = aoclsparse_fill_mode_upper, then

$$op(U) \cdot x = \alpha \cdot b$$

is solved.

To increase performance and if the matrix A is to be used more than once to solve for different right-hand sides b's, then it is encouraged to provide hints using `aoclsparse_set_sv_hint` and `aoclsparse_optimize`, otherwise, the optimization for the matrix will be done by the solver on entry.

There is a `_kid` (Kernel ID) variation of TRSV, namely with a suffix of `_kid`, this solver allows to choose which TRSV kernel to use (if possible). Currently the possible choices are: `kid=0` Reference implementation (No explicit AVX instructions). `kid=1` Reference AVX 256-bit implementation only for double data precision and for operations aoclsparse_operation_none and aoclsparse_operation_transpose. `kid=2` Kernel Template version using AVX/AVX2 extensions. `kid=3` Kernel Template version using AVX512F+ CPU extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

`in`	*trans*	matrix operation type, either aoclsparse_operation_none, aoclsparse_operation_transpose, or aoclsparse_operation_conjugate_transpose.
`in`	*alpha*	scalar α, used to premultiply right-hand side vector b.
`in,out`	*A*	matrix data. `A` is modified only if solver requires to optimize matrix data.
`in`	*descr*	matrix descriptor. Supported matrix types are aoclsparse_matrix_type_symmetric and aoclsparse_matrix_type_triangular.
`in`	*b*	array of `m` elements, storing the right-hand side.
`out`	*x*	array of `m` elements, storing the solution if solver returns aoclsparse_status_success.
`in`	*kid*	Kernel ID, hints a request on which TRSV kernel to use.

Return values

aoclsparse_status_success	the operation completed successfully and x contains the solution to the linear system of equations.
aoclsparse_status_invalid_size	matrix A or $op(A)$ is invalid.
aoclsparse_status_invalid_pointer	One or more of `A`, `descr`, `x`, `b` are invalid pointers.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	the requested operation is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.49 aoclsparse_ctrsv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ctrsv (
    aoclsparse_operation trans,
    const aoclsparse_float_complex alpha,
    aoclsparse_matrix A,
    const aoclsparse_mat_descr descr,
    const aoclsparse_float_complex * b,
    aoclsparse_float_complex * x )
```

Sparse triangular solver for real/complex single and double data precisions.

The functions `aoclsparse_?trsv` solve sparse lower (or upper) triangular linear system of equations. The system is defined by the sparse $m \times m$ matrix A, the dense solution m-vector x, and the right-hand side dense m-vector b. Vector b is multiplied by α. The solution x is estimated by solving

$$op(L) \cdot x = \alpha \cdot b, \quad \text{or} \quad op(U) \cdot x = \alpha \cdot b,$$

where $L = \text{tril}(A)$ is the lower triangle of matrix A, similarly, $U = \text{triu}(A)$ is the upper triangle of matrix A. The operator $op()$ is regarded as the matrix linear operation,

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

If the matrix descriptor `descr` specifies that the matrix A is to be regarded has having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are all considered to be unitary.

The input matrix need not be (upper or lower) triangular matrix, in the `descr`, the `fill_mode` entity specifies which triangle to consider, namely, if `fill_mode` = aoclsparse_fill_mode_lower, then

$$op(L) \cdot x = \alpha \cdot b,$$

otherwise, if `fill_mode` = aoclsparse_fill_mode_upper, then

$$op(U) \cdot x = \alpha \cdot b$$

is solved.

To increase performance and if the matrix A is to be used more than once to solve for different right-hand sides b's, then it is encouraged to provide hints using `aoclsparse_set_sv_hint` and `aoclsparse_optimize`, otherwise, the optimization for the matrix will be done by the solver on entry.

There is a `_kid` (Kernel ID) variation of TRSV, namely with a suffix of `_kid`, this solver allows to choose which TRSV kernel to use (if possible). Currently the possible choices are: `kid=0` Reference implementation (No explicit AVX instructions). `kid=1` Reference AVX 256-bit implementation only for double data precision and for operations aoclsparse_operation_none and aoclsparse_operation_transpose. `kid=2` Kernel Template version using AVX/AVX2 extensions. `kid=3` Kernel Template version using AVX512F+ CPU extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

in	*trans*	matrix operation type, either aoclsparse_operation_none, aoclsparse_operation_transpose, or aoclsparse_operation_conjugate_transpose.
in	*alpha*	scalar α, used to premultiply right-hand side vector b.
in,out	*A*	matrix data. `A` is modified only if solver requires to optimize matrix data.
in	*descr*	matrix descriptor. Supported matrix types are aoclsparse_matrix_type_symmetric and aoclsparse_matrix_type_triangular.
in	*b*	array of `m` elements, storing the right-hand side.
out	*x*	array of `m` elements, storing the solution if solver returns aoclsparse_status_success.
in	*kid*	Kernel ID, hints a request on which TRSV kernel to use.

Return values

aoclsparse_status_success	the operation completed successfully and x contains the solution to the linear system of equations.
aoclsparse_status_invalid_size	matrix A or $op(A)$ is invalid.
aoclsparse_status_invalid_pointer	One or more of `A`, `descr`, `x`, `b` are invalid pointers.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	the requested operation is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.50 aoclsparse_ztrsv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ztrsv (
        aoclsparse_operation trans,
        const aoclsparse_double_complex alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        const aoclsparse_double_complex * b,
        aoclsparse_double_complex * x )
```

Sparse triangular solver for real/complex single and double data precisions.

The functions `aoclsparse_?trsv` solve sparse lower (or upper) triangular linear system of equations. The system is defined by the sparse $m \times m$ matrix A, the dense solution m-vector x, and the right-hand side dense m-vector b. Vector b is multiplied by α. The solution x is estimated by solving

$$op(L) \cdot x = \alpha \cdot b, \quad \text{or} \quad op(U) \cdot x = \alpha \cdot b,$$

where $L = \text{tril}(A)$ is the lower triangle of matrix A, similarly, $U = \text{triu}(A)$ is the upper triangle of matrix A. The operator $op()$ is regarded as the matrix linear operation,

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

If the matrix descriptor `descr` specifies that the matrix A is to be regarded has having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are all considered to be unitary.

The input matrix need not be (upper or lower) triangular matrix, in the `descr`, the `fill_mode` entity specifies which triangle to consider, namely, if `fill_mode` = aoclsparse_fill_mode_lower, then

$$op(L) \cdot x = \alpha \cdot b,$$

otherwise, if `fill_mode` = aoclsparse_fill_mode_upper, then

$$op(U) \cdot x = \alpha \cdot b$$

is solved.

To increase performance and if the matrix A is to be used more than once to solve for different right-hand sides b's, then it is encouraged to provide hints using `aoclsparse_set_sv_hint` and `aoclsparse↵_optimize`, otherwise, the optimization for the matrix will be done by the solver on entry.

There is a `_kid` (Kernel ID) variation of TRSV, namely with a suffix of `_kid`, this solver allows to choose which TRSV kernel to use (if possible). Currently the possible choices are: `kid=0` Reference implementation (No explicit AVX instructions). `kid=1` Reference AVX 256-bit implementation only for double data precision and for operations aoclsparse_operation_none and aoclsparse_operation_transpose. `kid=2` Kernel Template version using AVX/AVX2 extensions. `kid=3` Kernel Template version using AVX512F+ CPU extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

in	*trans*	matrix operation type, either aoclsparse_operation_none, aoclsparse_operation_transpose, or aoclsparse_operation_conjugate_transpose.
in	*alpha*	scalar α, used to premultiply right-hand side vector b.
in,out	*A*	matrix data. `A` is modified only if solver requires to optimize matrix data.
in	*descr*	matrix descriptor. Supported matrix types are aoclsparse_matrix_type_symmetric and aoclsparse_matrix_type_triangular.
in	*b*	array of `m` elements, storing the right-hand side.
out	*x*	array of `m` elements, storing the solution if solver returns aoclsparse_status_success.
in	*kid*	Kernel ID, hints a request on which TRSV kernel to use.

Return values

aoclsparse_status_success	the operation completed successfully and x contains the solution to the linear system of equations.
aoclsparse_status_invalid_size	matrix A or $op(A)$ is invalid.
aoclsparse_status_invalid_pointer	One or more of `A`, `descr`, `x`, `b` are invalid pointers.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	the requested operation is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.51 aoclsparse_strsv_kid()

```
DLL_PUBLIC aoclsparse_status aoclsparse_strsv_kid (
        aoclsparse_operation trans,
        const float alpha,
```

```
aoclsparse_matrix A,
const aoclsparse_mat_descr descr,
const float * b,
float * x,
const aoclsparse_int kid )
```

Sparse triangular solver for real/complex single and double data precisions.

The functions `aoclsparse_?trsv` solve sparse lower (or upper) triangular linear system of equations. The system is defined by the sparse $m \times m$ matrix A, the dense solution m-vector x, and the right-hand side dense m-vector b. Vector b is multiplied by α. The solution x is estimated by solving

$$op(L) \cdot x = \alpha \cdot b, \quad \text{or} \quad op(U) \cdot x = \alpha \cdot b,$$

where $L = \text{tril}(A)$ is the lower triangle of matrix A, similarly, $U = \text{triu}(A)$ is the upper triangle of matrix A. The operator $op()$ is regarded as the matrix linear operation,

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

If the matrix descriptor `descr` specifies that the matrix A is to be regarded has having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are all considered to be unitary.

The input matrix need not be (upper or lower) triangular matrix, in the `descr`, the `fill_mode` entity specifies which triangle to consider, namely, if `fill_mode` = aoclsparse_fill_mode_lower, then

$$op(L) \cdot x = \alpha \cdot b,$$

otherwise, if `fill_mode` = aoclsparse_fill_mode_upper, then

$$op(U) \cdot x = \alpha \cdot b$$

is solved.

To increase performance and if the matrix A is to be used more than once to solve for different right-hand sides b's, then it is encouraged to provide hints using `aoclsparse_set_sv_hint` and `aoclsparse_optimize`, otherwise, the optimization for the matrix will be done by the solver on entry.

There is a `_kid` (Kernel ID) variation of TRSV, namely with a suffix of `_kid`, this solver allows to choose which TRSV kernel to use (if possible). Currently the possible choices are: `kid=0` Reference implementation (No explicit AVX instructions). `kid=1` Reference AVX 256-bit implementation only for double data precision and for operations aoclsparse_operation_none and aoclsparse_operation_transpose. `kid=2` Kernel Template version using AVX/AVX2 extensions. `kid=3` Kernel Template version using AVX512F+ CPU extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

in	*trans*	matrix operation type, either aoclsparse_operation_none, aoclsparse_operation_transpose, or aoclsparse_operation_conjugate_transpose.
in	*alpha*	scalar α, used to premultiply right-hand side vector b.
in,out	*A*	matrix data. `A` is modified only if solver requires to optimize matrix data.
in	*descr*	matrix descriptor. Supported matrix types are aoclsparse_matrix_type_symmetric and aoclsparse_matrix_type_triangular.
in	*b*	array of `m` elements, storing the right-hand side.
out	*x*	array of `m` elements, storing the solution if solver returns aoclsparse_status_success.
in	*kid*	Kernel ID, hints a request on which TRSV kernel to use.

Return values

aoclsparse_status_success	the operation completed successfully and x contains the solution to the linear system of equations.
aoclsparse_status_invalid_size	matrix A or $op(A)$ is invalid.
aoclsparse_status_invalid_pointer	One or more of `A`, `descr`, `x`, `b` are invalid pointers.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	the requested operation is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.52 aoclsparse_dtrsv_kid()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dtrsv_kid (
        aoclsparse_operation trans,
        const double alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        const double * b,
        double * x,
        const aoclsparse_int kid )
```

Sparse triangular solver for real/complex single and double data precisions.

The functions `aoclsparse_?trsv` solve sparse lower (or upper) triangular linear system of equations. The system is defined by the sparse $m \times m$ matrix A, the dense solution m-vector x, and the right-hand side dense m-vector b. Vector b is multiplied by α. The solution x is estimated by solving

$$op(L) \cdot x = \alpha \cdot b, \quad \text{or} \quad op(U) \cdot x = \alpha \cdot b,$$

where $L = \text{tril}(A)$ is the lower triangle of matrix A, similarly, $U = \text{triu}(A)$ is the upper triangle of matrix A. The operator $op()$ is regarded as the matrix linear operation,

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

If the matrix descriptor `descr` specifies that the matrix A is to be regarded has having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are all considered to be unitary.

The input matrix need not be (upper or lower) triangular matrix, in the `descr`, the `fill_mode` entity specifies which triangle to consider, namely, if `fill_mode` = aoclsparse_fill_mode_lower, then

$$op(L) \cdot x = \alpha \cdot b,$$

otherwise, if `fill_mode` = aoclsparse_fill_mode_upper, then

$$op(U) \cdot x = \alpha \cdot b$$

is solved.

To increase performance and if the matrix A is to be used more than once to solve for different right-hand sides b's, then it is encouraged to provide hints using `aoclsparse_set_sv_hint` and `aoclsparse_optimize`, otherwise, the optimization for the matrix will be done by the solver on entry.

There is a `_kid` (Kernel ID) variation of TRSV, namely with a suffix of `_kid`, this solver allows to choose which TRSV kernel to use (if possible). Currently the possible choices are: `kid=0` Reference implementation (No explicit AVX instructions). `kid=1` Reference AVX 256-bit implementation only for double data precision and for operations aoclsparse_operation_none and aoclsparse_operation_transpose. `kid=2` Kernel Template version using AVX/AVX2 extensions. `kid=3` Kernel Template version using AVX512F+ CPU extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

in	*trans*	matrix operation type, either aoclsparse_operation_none, aoclsparse_operation_transpose, or aoclsparse_operation_conjugate_transpose.
in	*alpha*	scalar α, used to premultiply right-hand side vector b.
in,out	*A*	matrix data. `A` is modified only if solver requires to optimize matrix data.
in	*descr*	matrix descriptor. Supported matrix types are aoclsparse_matrix_type_symmetric and aoclsparse_matrix_type_triangular.
in	*b*	array of `m` elements, storing the right-hand side.
out	*x*	array of `m` elements, storing the solution if solver returns aoclsparse_status_success.
in	*kid*	Kernel ID, hints a request on which TRSV kernel to use.

Return values

aoclsparse_status_success	the operation completed successfully and x contains the solution to the linear system of equations.
aoclsparse_status_invalid_size	matrix A or $op(A)$ is invalid.
aoclsparse_status_invalid_pointer	One or more of `A`, `descr`, `x`, `b` are invalid pointers.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	the requested operation is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.53 aoclsparse_ctrsv_kid()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ctrsv_kid (
        aoclsparse_operation trans,
        const aoclsparse_float_complex alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        const aoclsparse_float_complex * b,
        aoclsparse_float_complex * x,
        const aoclsparse_int kid )
```

Sparse triangular solver for real/complex single and double data precisions.

The functions `aoclsparse_?trsv` solve sparse lower (or upper) triangular linear system of equations. The system is defined by the sparse $m \times m$ matrix A, the dense solution m-vector x, and the right-hand side dense m-vector b. Vector b is multiplied by α. The solution x is estimated by solving

$$op(L) \cdot x = \alpha \cdot b, \quad \text{or} \quad op(U) \cdot x = \alpha \cdot b,$$

where $L = \text{tril}(A)$ is the lower triangle of matrix A, similarly, $U = \text{triu}(A)$ is the upper triangle of matrix A. The operator $op()$ is regarded as the matrix linear operation,

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

If the matrix descriptor `descr` specifies that the matrix A is to be regarded has having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are all considered to be unitary.

The input matrix need not be (upper or lower) triangular matrix, in the `descr`, the `fill_mode` entity specifies which triangle to consider, namely, if `fill_mode` = aoclsparse_fill_mode_lower, then

$$op(L) \cdot x = \alpha \cdot b,$$

otherwise, if `fill_mode` = aoclsparse_fill_mode_upper, then

$$op(U) \cdot x = \alpha \cdot b$$

is solved.

To increase performance and if the matrix A is to be used more than once to solve for different right-hand sides b's, then it is encouraged to provide hints using `aoclsparse_set_sv_hint` and `aoclsparse↩_optimize`, otherwise, the optimization for the matrix will be done by the solver on entry.

There is a `_kid` (Kernel ID) variation of TRSV, namely with a suffix of `_kid`, this solver allows to choose which TRSV kernel to use (if possible). Currently the possible choices are: `kid=0` Reference implementation (No explicit AVX instructions). `kid=1` Reference AVX 256-bit implementation only for double data precision and for operations aoclsparse_operation_none and aoclsparse_operation_transpose. `kid=2` Kernel Template version using AVX/AVX2 extensions. `kid=3` Kernel Template version using AVX512F+ CPU extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

in	*trans*	matrix operation type, either aoclsparse_operation_none, aoclsparse_operation_transpose, or aoclsparse_operation_conjugate_transpose.
in	*alpha*	scalar α, used to premultiply right-hand side vector b.
in,out	*A*	matrix data. `A` is modified only if solver requires to optimize matrix data.
in	*descr*	matrix descriptor. Supported matrix types are aoclsparse_matrix_type_symmetric and aoclsparse_matrix_type_triangular.
in	*b*	array of `m` elements, storing the right-hand side.
out	*x*	array of `m` elements, storing the solution if solver returns aoclsparse_status_success.
in	*kid*	Kernel ID, hints a request on which TRSV kernel to use.

Return values

aoclsparse_status_success	the operation completed successfully and x contains the solution to the linear system of equations.
aoclsparse_status_invalid_size	matrix A or $op(A)$ is invalid.
aoclsparse_status_invalid_pointer	One or more of `A`, `descr`, `x`, `b` are invalid pointers.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	the requested operation is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.54 aoclsparse_ztrsv_kid()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ztrsv_kid (
        aoclsparse_operation trans,
        const aoclsparse_double_complex alpha,
```

```
aoclsparse_matrix A,
const aoclsparse_mat_descr descr,
const aoclsparse_double_complex * b,
aoclsparse_double_complex * x,
const aoclsparse_int kid )
```

Sparse triangular solver for real/complex single and double data precisions.

The functions `aoclsparse_?trsv` solve sparse lower (or upper) triangular linear system of equations. The system is defined by the sparse $m \times m$ matrix A, the dense solution m-vector x, and the right-hand side dense m-vector b. Vector b is multiplied by α. The solution x is estimated by solving

$$op(L) \cdot x = \alpha \cdot b, \quad \text{or} \quad op(U) \cdot x = \alpha \cdot b,$$

where $L = \text{tril}(A)$ is the lower triangle of matrix A, similarly, $U = \text{triu}(A)$ is the upper triangle of matrix A. The operator $op()$ is regarded as the matrix linear operation,

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Note

This routine supports only sparse matrices in CSR format.

If the matrix descriptor `descr` specifies that the matrix A is to be regarded has having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are all considered to be unitary.

The input matrix need not be (upper or lower) triangular matrix, in the `descr`, the `fill_mode` entity specifies which triangle to consider, namely, if `fill_mode` = aoclsparse_fill_mode_lower, then

$$op(L) \cdot x = \alpha \cdot b,$$

otherwise, if `fill_mode` = aoclsparse_fill_mode_upper, then

$$op(U) \cdot x = \alpha \cdot b$$

is solved.

To increase performance and if the matrix A is to be used more than once to solve for different right-hand sides b's, then it is encouraged to provide hints using `aoclsparse_set_sv_hint` and `aoclsparse_optimize`, otherwise, the optimization for the matrix will be done by the solver on entry.

There is a `_kid` (Kernel ID) variation of TRSV, namely with a suffix of `_kid`, this solver allows to choose which TRSV kernel to use (if possible). Currently the possible choices are: `kid=0` Reference implementation (No explicit AVX instructions). `kid=1` Reference AVX 256-bit implementation only for double data precision and for operations aoclsparse_operation_none and aoclsparse_operation_transpose. `kid=2` Kernel Template version using AVX/AVX2 extensions. `kid=3` Kernel Template version using AVX512F+ CPU extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

`in`	*trans*	matrix operation type, either aoclsparse_operation_none, aoclsparse_operation_transpose, or aoclsparse_operation_conjugate_transpose.
`in`	*alpha*	scalar α, used to premultiply right-hand side vector b.
`in,out`	*A*	matrix data. `A` is modified only if solver requires to optimize matrix data.
`in`	*descr*	matrix descriptor. Supported matrix types are aoclsparse_matrix_type_symmetric and aoclsparse_matrix_type_triangular.
`in`	*b*	array of `m` elements, storing the right-hand side.
`out`	*x*	array of `m` elements, storing the solution if solver returns aoclsparse_status_success.
`in`	*kid*	Kernel ID, hints a request on which TRSV kernel to use.

Return values

aoclsparse_status_success	the operation completed successfully and x contains the solution to the linear system of equations.
aoclsparse_status_invalid_size	matrix A or $op(A)$ is invalid.
aoclsparse_status_invalid_pointer	One or more of `A`, `descr`, `x`, `b` are invalid pointers.
aoclsparse_status_internal_error	an internal error occurred.
aoclsparse_status_not_implemented	the requested operation is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.55 aoclsparse_sdotmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_sdotmv (
    const aoclsparse_operation op,
    const float alpha,
    aoclsparse_matrix A,
    const aoclsparse_mat_descr descr,
    const float * x,
    const float beta,
    float * y,
    float * d )
```

Performs sparse matrix-vector multiplication followed by vector-vector multiplication.

`aoclsparse_`?dotmv multiplies the scalar α with a sparse $m \times n$ matrix, defined in a sparse storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

followed by dot product of dense vectors x and y such that

$$\text{d} = \begin{cases} \sum_{i=0}^{min(m,n)-1} x_i * y_i, & \text{real case} \\ \sum_{i=0}^{min(m,n)-1} \text{conj}(x_i) * y_i, & \text{complex case} \end{cases}$$

Note

Currently, Hermitian matrix is not supported.

Parameters

`in`	*op*	matrix operation type.
`in`	*alpha*	scalar α.
`in`	*A*	the sparse $m \times n$ matrix structure that is created using `aoclsparse_create_(s/d/c/z)csr`
`in`	*descr*	descriptor of the sparse CSR matrix. Both base-zero and base-one are supported, however, the index base needs to match the one used when aoclsparse_matrix was created.
`in`	*x*	array of atleast `n` elements if $op(A) = A$ or atleast `m` elements if $op(A) = A^T or A^H$.
`in`	*beta*	scalar β.
`in,out`	*y*	array of atleast `m` elements if $op(A) = A$ or atleast `n` elements if $op(A) = A^T or A^H$.
`out`	*d*	dot product of y and x

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_value	(base != aoclsparse_index_base_zero) or, (base != aoclsparse_index_base_one) or, matrix base and descr base value do not match.
aoclsparse_status_invalid_pointer	`descr`, `internal` structures related to the sparse matrix `A`, `x`, `y` or `d` are invalid pointer.
aoclsparse_status_wrong_type	matrix data type is not supported.
aoclsparse_status_not_implemented	(aoclsparse_matrix_type == aoclsparse_matrix_type_hermitian) or, (aoclsparse_matrix_format_type != aoclsparse_csr_mat)

3.4.2.56 aoclsparse_ddotmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ddotmv (
        const aoclsparse_operation op,
        const double alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        const double * x,
        const double beta,
        double * y,
        double * d )
```

Performs sparse matrix-vector multiplication followed by vector-vector multiplication.

`aoclsparse_`?dotmv multiplies the scalar α with a sparse $m \times n$ matrix, defined in a sparse storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

followed by dot product of dense vectors x and y such that

$$\text{d} = \begin{cases} \sum_{i=0}^{min(m,n)-1} x_i * y_i, & \text{real case} \\ \sum_{i=0}^{min(m,n)-1} \text{conj}(x_i) * y_i, & \text{complex case} \end{cases}$$

Note

Currently, Hermitian matrix is not supported.

Parameters

in	*op*	matrix operation type.
in	*alpha*	scalar α.
in	*A*	the sparse $m \times n$ matrix structure that is created using `aoclsparse_create_(s/d/c/z)csr`

Parameters

in	*descr*	descriptor of the sparse CSR matrix. Both base-zero and base-one are supported, however, the index base needs to match the one used when aoclsparse_matrix was created.
in	*x*	array of atleast `n` elements if $op(A) = A$ or atleast `m` elements if $op(A) = A^T or A^H$.
in	*beta*	scalar β.
in,out	*y*	array of atleast `m` elements if $op(A) = A$ or atleast `n` elements if $op(A) = A^T or A^H$.
out	*d*	dot product of y and x

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_value	(base != aoclsparse_index_base_zero) or, (base != aoclsparse_index_base_one) or, matrix base and descr base value do not match.
aoclsparse_status_invalid_pointer	`descr`, `internal` structures related to the sparse matrix `A`, `x`, `y` or `d` are invalid pointer.
aoclsparse_status_wrong_type	matrix data type is not supported.
aoclsparse_status_not_implemented	(aoclsparse_matrix_type == aoclsparse_matrix_type_hermitian) or, (aoclsparse_matrix_format_type != aoclsparse_csr_mat)

3.4.2.57 aoclsparse_cdotmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_cdotmv (
        const aoclsparse_operation op,
        const aoclsparse_float_complex alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        const aoclsparse_float_complex * x,
        const aoclsparse_float_complex beta,
        aoclsparse_float_complex * y,
        aoclsparse_float_complex * d )
```

Performs sparse matrix-vector multiplication followed by vector-vector multiplication.

`aoclsparse_`?dotmv multiplies the scalar α with a sparse $m \times n$ matrix, defined in a sparse storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

followed by dot product of dense vectors x and y such that

$$\text{d} = \begin{cases} \sum_{i=0}^{min(m,n)-1} x_i * y_i, & \text{real case} \\ \sum_{i=0}^{min(m,n)-1} \text{conj}(x_i) * y_i, & \text{complex case} \end{cases}$$

Note

Currently, Hermitian matrix is not supported.

Parameters

in	*op*	matrix operation type.
in	*alpha*	scalar α.
in	*A*	the sparse $m \times n$ matrix structure that is created using `aoclsparse_create_(s/d/c/z)csr`
in	*descr*	descriptor of the sparse CSR matrix. Both base-zero and base-one are supported, however, the index base needs to match the one used when aoclsparse_matrix was created.
in	*x*	array of atleast `n` elements if $op(A) = A$ or atleast `m` elements if $op(A) = A^T or A^H$.
in	*beta*	scalar β.
in,out	*y*	array of atleast `m` elements if $op(A) = A$ or atleast `n` elements if $op(A) = A^T or A^H$.
out	*d*	dot product of y and x

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_value	(base != aoclsparse_index_base_zero) or, (base != aoclsparse_index_base_one) or, matrix base and descr base value do not match.
aoclsparse_status_invalid_pointer	`descr`, `internal` structures related to the sparse matrix `A`, `x`, `y` or `d` are invalid pointer.
aoclsparse_status_wrong_type	matrix data type is not supported.
aoclsparse_status_not_implemented	(aoclsparse_matrix_type == aoclsparse_matrix_type_hermitian) or, (aoclsparse_matrix_format_type != aoclsparse_csr_mat)

3.4.2.58 aoclsparse_zdotmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zdotmv (
        const aoclsparse_operation op,
        const aoclsparse_double_complex alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        const aoclsparse_double_complex * x,
        const aoclsparse_double_complex beta,
        aoclsparse_double_complex * y,
        aoclsparse_double_complex * d )
```

Performs sparse matrix-vector multiplication followed by vector-vector multiplication.

`aoclsparse_`?dotmv multiplies the scalar α with a sparse $m \times n$ matrix, defined in a sparse storage format, and the dense vector x and adds the result to the dense vector y that is multiplied by the scalar β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

followed by dot product of dense vectors x and y such that

$$\text{d} = \begin{cases} \sum_{i=0}^{min(m,n)-1} x_i * y_i, & \text{real case} \\ \sum_{i=0}^{min(m,n)-1} \text{conj}(x_i) * y_i, & \text{complex case} \end{cases}$$

Note

Currently, Hermitian matrix is not supported.

Parameters

in	*op*	matrix operation type.
in	*alpha*	scalar α.
in	*A*	the sparse $m \times n$ matrix structure that is created using `aoclsparse_create_(s/d/c/z)csr`
in	*descr*	descriptor of the sparse CSR matrix. Both base-zero and base-one are supported, however, the index base needs to match the one used when aoclsparse_matrix was created.
in	*x*	array of atleast `n` elements if $op(A) = A$ or atleast `m` elements if $op(A) = A^T or A^H$.
in	*beta*	scalar β.
in,out	*y*	array of atleast `m` elements if $op(A) = A$ or atleast `n` elements if $op(A) = A^T or A^H$.
out	*d*	dot product of y and x

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	`m`, `n` or `nnz` is invalid.
aoclsparse_status_invalid_value	(base != aoclsparse_index_base_zero) or, (base != aoclsparse_index_base_one) or, matrix base and descr base value do not match.
aoclsparse_status_invalid_pointer	`descr`, `internal` structures related to the sparse matrix `A`, `x`, `y` or `d` are invalid pointer.
aoclsparse_status_wrong_type	matrix data type is not supported.
aoclsparse_status_not_implemented	(aoclsparse_matrix_type == aoclsparse_matrix_type_hermitian) or, (aoclsparse_matrix_format_type != aoclsparse_csr_mat)

3.4.2.59 aoclsparse_strsm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_strsm (
    const aoclsparse_operation trans,
    const float alpha,
    aoclsparse_matrix A,
    const aoclsparse_mat_descr descr,
    aoclsparse_order order,
    const float * B,
    aoclsparse_int n,
    aoclsparse_int ldb,
    float * X,
    aoclsparse_int ldx )
```

Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

`aoclsparse_?trsm` solves a sparse triangular linear system of equations with multiple right hand sides, of the form

$$op(A)\ X = \alpha B,$$

where A is a sparse matrix of size m, $op()$ is a linear operator, X and B are rectangular dense matrices of appropiate size, while α is a scalar. The sparse matrix A can be interpreted either as a lower triangular or upper triangular. This is indicated by `fill_mode` from the matrix descriptor `descr` where either upper or lower triangular portion of the matrix is only referenced. The matrix can also be of class symmetric in which case only the selected triangular part is used. Matrix A must be of full rank, that is, the matrix must be invertible. The linear operator $op()$ can define the transposition or Hermitian transposition operations. By default, no transposition is performed. The right-hand-side matrix B and the solution matrix X are dense and must be of the correct size, that is m by n, see `ldb` and `ldx` input parameters for further details.

Explicitly, this kernel solves

$$op(A)\ X = \alpha\ B, \text{ with solution } X = \alpha\ (op(A)^{-1})\ B,$$

where

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none}, \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose}, \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose}. \end{cases}$$

If a linear operator is applied then, the possible problems solved are

$$A^T\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-T}\ B, \text{ and } \ A^H\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-H}\ B.$$

Note

1. If the matrix descriptor `descr` specifies that the matrix A is to be regarded as having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are considered to all be ones.
2. If the matrix A is described as upper triangular, then only the upper triangular portion of the matrix is referenced. Conversely, if the matrix A is described lower triangular, then only the lower triangular portion of the matrix is used.
3. This set of APIs allocates work array of size m for each case where the matrices B or X are stored in row-major format (aoclsparse_order_row).
4. A subset of kernels are parallel (on parallel builds) and can be expected potential acceleration in the solve. These kernels are available when both dense matrices X and B are stored in column-major format (aoclsparse_order_column) and thread count is greater than 1 on a parallel build.
5. There is `_kid` (Kernel ID) variation of TRSM, namely with a suffix of `_kid`, this solver allows to choose which underlying TRSV kernels to use (if possible). Currently, all the existing `aoclsparse_?trsm` kernels are supported.
6. This routine supports only sparse matrices in CSR format.

Parameters

in	*trans*	matrix operation to perform on A. Possible values are aoclsparse_operation_none, aoclsparse_operation_transpose, and aoclsparse_operation_conjugate_transpose.
in	*alpha*	scalar α.
in	*A*	sparse matrix A of size m.
in	*descr*	descriptor of the sparse matrix A.
in	*order*	storage order of dense matrices B and X. Possible options are aoclsparse_order_row and aoclsparse_order_column.
in	*B*	dense matrix, potentially rectangular, of size $m \times n$.
in	*n*	n, number of columns of the dense matrix B.

Parameters

<table>
<tr><td>in</td><td>ldb</td><td>leading dimension of B. Eventhough the matrix B is considered of size $m \times n$, its memory layout may correspond to a larger matrix (<code>ldb</code> by $N > n$) in which only the submatrix B is of interest. In this case, this parameter provides means to access the correct elements of B within the larger layout.
<table>
<tr><th>matrix layout</th><th>row count</th><th>column count</th></tr>
<tr><td>aoclsparse_order_row</td><td>m</td><td><code>ldb</code> with <code>ldb</code> $\geq n$</td></tr>
<tr><td>aoclsparse_order_column</td><td><code>ldb</code> with <code>ldb</code> $\geq m$</td><td>n</td></tr>
</table></td></tr>
<tr><td>out</td><td>X</td><td>solution matrix X, dense and potentially rectangular matrix of size $m \times n$.</td></tr>
<tr><td>in</td><td>ldx</td><td>leading dimension of X. Eventhough the matrix X is considered of size $m \times n$, its memory layout may correspond to a larger matrix (<code>ldx</code> by $N > n$) in which only the submatrix X is of interest. In this case, this parameter provides means to access the correct elements of X within the larger layout.
<table>
<tr><th>matrix layout</th><th>row count</th><th>column count</th></tr>
<tr><td>aoclsparse_order_row</td><td>m</td><td><code>ldx</code> with <code>ldx</code> $\geq n$</td></tr>
<tr><td>aoclsparse_order_column</td><td><code>ldx</code> with <code>ldx</code> $\geq m$</td><td>n</td></tr>
</table></td></tr>
<tr><td>in</td><td>kid</td><td>kernel ID, hints a request on which kernel to use (see notes).</td></tr>
</table>

Return values

aoclsparse_status_success	indicates that the operation completed successfully.
aoclsparse_status_invalid_size	informs that either `m`, `n`, `nnz`, `ldb` or `ldx` is invalid.
aoclsparse_status_invalid_pointer	informs that either `descr`, `alpha`, `A`, `B`, or `X` pointer is invalid.
aoclsparse_status_not_implemented	this error occurs when the provided matrix aoclsparse_matrix_type is aoclsparse_matrix_type_general or aoclsparse_matrix_type_hermitian or when matrix `A` is not in CSR format.

3.4.2.60 aoclsparse_dtrsm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dtrsm (
        const aoclsparse_operation trans,
        const double alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        aoclsparse_order order,
        const double * B,
        aoclsparse_int n,
        aoclsparse_int ldb,
        double * X,
        aoclsparse_int ldx )
```

Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

`aoclsparse_?trsm` solves a sparse triangular linear system of equations with multiple right hand sides, of the form

$$op(A)\ X = \alpha B,$$

where A is a sparse matrix of size m, $op()$ is a linear operator, X and B are rectangular dense matrices of appropriate size, while α is a scalar. The sparse matrix A can be interpreted either as a lower triangular or upper triangular. This is indicated by `fill_mode` from the matrix descriptor `descr` where either upper or lower triangular portion

of the matrix is only referenced. The matrix can also be of class symmetric in which case only the selected triangular part is used. Matrix A must be of full rank, that is, the matrix must be invertible. The linear operator $op()$ can define the transposition or Hermitian transposition operations. By default, no transposition is performed. The right-hand-side matrix B and the solution matrix X are dense and must be of the correct size, that is m by n, see `ldb` and `ldx` input parameters for further details.

Explicitly, this kernel solves

$$op(A)\ X = \alpha\ B, \text{ with solution } X = \alpha\ (op(A)^{-1})\ B,$$

where

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none}, \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose}, \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose}. \end{cases}$$

If a linear operator is applied then, the possible problems solved are

$$A^T\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-T}\ B, \text{ and } \ A^H\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-H}\ B.$$

Note

1. If the matrix descriptor `descr` specifies that the matrix A is to be regarded as having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are considered to all be ones.
2. If the matrix A is described as upper triangular, then only the upper triangular portion of the matrix is referenced. Conversely, if the matrix A is described lower triangular, then only the lower triangular portion of the matrix is used.
3. This set of APIs allocates work array of size m for each case where the matrices B or X are stored in row-major format (aoclsparse_order_row).
4. A subset of kernels are parallel (on parallel builds) and can be expected potential acceleration in the solve. These kernels are available when both dense matrices X and B are stored in column-major format (aoclsparse_order_column) and thread count is greater than 1 on a parallel build.
5. There is `_kid` (Kernel ID) variation of TRSM, namely with a suffix of `_kid`, this solver allows to choose which underlying TRSV kernels to use (if possible). Currently, all the existing `aoclsparse_?trsm` kernels are supported.
6. This routine supports only sparse matrices in CSR format.

Parameters

`in`	*trans*	matrix operation to perform on A. Possible values are aoclsparse_operation_none, aoclsparse_operation_transpose, and aoclsparse_operation_conjugate_transpose.
`in`	*alpha*	scalar α.
`in`	*A*	sparse matrix A of size m.
`in`	*descr*	descriptor of the sparse matrix A.
`in`	*order*	storage order of dense matrices B and X. Possible options are aoclsparse_order_row and aoclsparse_order_column.
`in`	*B*	dense matrix, potentially rectangular, of size $m \times n$.
`in`	*n*	n, number of columns of the dense matrix B.

Parameters

<table>
<tr><td>in</td><td>ldb</td><td>leading dimension of B. Eventhough the matrix B is considered of size $m \times n$, its memory layout may correspond to a larger matrix (`ldb` by $N > n$) in which only the submatrix B is of interest. In this case, this parameter provides means to access the correct elements of B within the larger layout.
<table>
<tr><th>matrix layout</th><th>row count</th><th>column count</th></tr>
<tr><td>aoclsparse_order_row</td><td>m</td><td>ldb with ldb $\geq n$</td></tr>
<tr><td>aoclsparse_order_column</td><td>ldb with ldb $\geq m$</td><td>n</td></tr>
</table>
</td></tr>
<tr><td>out</td><td>X</td><td>solution matrix X, dense and potentially rectangular matrix of size $m \times n$.</td></tr>
<tr><td>in</td><td>ldx</td><td>leading dimension of X. Eventhough the matrix X is considered of size $m \times n$, its memory layout may correspond to a larger matrix (`ldx` by $N > n$) in which only the submatrix X is of interest. In this case, this parameter provides means to access the correct elements of X within the larger layout.
<table>
<tr><th>matrix layout</th><th>row count</th><th>column count</th></tr>
<tr><td>aoclsparse_order_row</td><td>m</td><td>ldx with ldx $\geq n$</td></tr>
<tr><td>aoclsparse_order_column</td><td>ldx with ldx $\geq m$</td><td>n</td></tr>
</table>
</td></tr>
<tr><td>in</td><td>kid</td><td>kernel ID, hints a request on which kernel to use (see notes).</td></tr>
</table>

Return values

aoclsparse_status_success	indicates that the operation completed successfully.
aoclsparse_status_invalid_size	informs that either `m`, `n`, `nnz`, `ldb` or `ldx` is invalid.
aoclsparse_status_invalid_pointer	informs that either `descr`, `alpha`, `A`, `B`, or `X` pointer is invalid.
aoclsparse_status_not_implemented	this error occurs when the provided matrix aoclsparse_matrix_type is aoclsparse_matrix_type_general or aoclsparse_matrix_type_hermitian or when matrix `A` is not in CSR format.

3.4.2.61 aoclsparse_ctrsm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ctrsm (
        aoclsparse_operation trans,
        const aoclsparse_float_complex alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        aoclsparse_order order,
        const aoclsparse_float_complex * B,
        aoclsparse_int n,
        aoclsparse_int ldb,
        aoclsparse_float_complex * X,
        aoclsparse_int ldx )
```

Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

`aoclsparse_?trsm` solves a sparse triangular linear system of equations with multiple right hand sides, of the form

$$op(A)\; X = \alpha B,$$

where A is a sparse matrix of size m, $op()$ is a linear operator, X and B are rectangular dense matrices of appropriate size, while α is a scalar. The sparse matrix A can be interpreted either as a lower triangular or upper triangular. This is indicated by `fill_mode` from the matrix descriptor `descr` where either upper or lower triangular portion

of the matrix is only referenced. The matrix can also be of class symmetric in which case only the selected triangular part is used. Matrix A must be of full rank, that is, the matrix must be invertible. The linear operator $op()$ can define the transposition or Hermitian transposition operations. By default, no transposition is performed. The right-hand-side matrix B and the solution matrix X are dense and must be of the correct size, that is m by n, see `ldb` and `ldx` input parameters for further details.

Explicitly, this kernel solves

$$op(A)\ X = \alpha\ B, \text{ with solution } X = \alpha\ (op(A)^{-1})\ B,$$

where

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none}, \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose}, \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose}. \end{cases}$$

If a linear operator is applied then, the possible problems solved are

$$A^T\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-T}\ B, \text{ and } \ A^H\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-H}\ B.$$

Note

1. If the matrix descriptor `descr` specifies that the matrix A is to be regarded as having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are considered to all be ones.
2. If the matrix A is described as upper triangular, then only the upper triangular portion of the matrix is referenced. Conversely, if the matrix A is described lower triangular, then only the lower triangular portion of the matrix is used.
3. This set of APIs allocates work array of size m for each case where the matrices B or X are stored in row-major format (aoclsparse_order_row).
4. A subset of kernels are parallel (on parallel builds) and can be expected potential acceleration in the solve. These kernels are available when both dense matrices X and B are stored in column-major format (aoclsparse_order_column) and thread count is greater than 1 on a parallel build.
5. There is `_kid` (Kernel ID) variation of TRSM, namely with a suffix of `_kid`, this solver allows to choose which underlying TRSV kernels to use (if possible). Currently, all the existing `aoclsparse_?trsm` kernels are supported.
6. This routine supports only sparse matrices in CSR format.

Parameters

`in`	*trans*	matrix operation to perform on A. Possible values are aoclsparse_operation_none, aoclsparse_operation_transpose, and aoclsparse_operation_conjugate_transpose.
`in`	*alpha*	scalar α.
`in`	*A*	sparse matrix A of size m.
`in`	*descr*	descriptor of the sparse matrix A.
`in`	*order*	storage order of dense matrices B and X. Possible options are aoclsparse_order_row and aoclsparse_order_column.
`in`	*B*	dense matrix, potentially rectangular, of size $m \times n$.
`in`	*n*	n, number of columns of the dense matrix B.

Parameters

<table>
<tr><td>in</td><td>ldb</td><td>leading dimension of B. Eventhough the matrix B is considered of size $m \times n$, its memory layout may correspond to a larger matrix (ldb by $N > n$) in which only the submatrix B is of interest. In this case, this parameter provides means to access the correct elements of B within the larger layout.
<table>
<tr><th>matrix layout</th><th>row count</th><th>column count</th></tr>
<tr><td>aoclsparse_order_row</td><td>m</td><td>ldb with ldb $\geq n$</td></tr>
<tr><td>aoclsparse_order_column</td><td>ldb with ldb $\geq m$</td><td>n</td></tr>
</table>
</td></tr>
<tr><td>out</td><td>X</td><td>solution matrix X, dense and potentially rectangular matrix of size $m \times n$.</td></tr>
<tr><td>in</td><td>ldx</td><td>leading dimension of X. Eventhough the matrix X is considered of size $m \times n$, its memory layout may correspond to a larger matrix (ldx by $N > n$) in which only the submatrix X is of interest. In this case, this parameter provides means to access the correct elements of X within the larger layout.
<table>
<tr><th>matrix layout</th><th>row count</th><th>column count</th></tr>
<tr><td>aoclsparse_order_row</td><td>m</td><td>ldx with ldx $\geq n$</td></tr>
<tr><td>aoclsparse_order_column</td><td>ldx with ldx $\geq m$</td><td>n</td></tr>
</table>
</td></tr>
<tr><td>in</td><td>kid</td><td>kernel ID, hints a request on which kernel to use (see notes).</td></tr>
</table>

Return values

aoclsparse_status_success	indicates that the operation completed successfully.
aoclsparse_status_invalid_size	informs that either m, n, nnz, ldb or ldx is invalid.
aoclsparse_status_invalid_pointer	informs that either descr, alpha, A, B, or X pointer is invalid.
aoclsparse_status_not_implemented	this error occurs when the provided matrix aoclsparse_matrix_type is aoclsparse_matrix_type_general or aoclsparse_matrix_type_hermitian or when matrix A is not in CSR format.

3.4.2.62 aoclsparse_ztrsm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ztrsm (
        aoclsparse_operation trans,
        const aoclsparse_double_complex alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        aoclsparse_order order,
        const aoclsparse_double_complex * B,
        aoclsparse_int n,
        aoclsparse_int ldb,
        aoclsparse_double_complex * X,
        aoclsparse_int ldx )
```

Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

aoclsparse_?trsm solves a sparse triangular linear system of equations with multiple right hand sides, of the form

$$op(A)\;X = \alpha B,$$

where A is a sparse matrix of size m, $op()$ is a linear operator, X and B are rectangular dense matrices of appropriate size, while α is a scalar. The sparse matrix A can be interpreted either as a lower triangular or upper triangular. This is indicated by fill_mode from the matrix descriptor descr where either upper or lower triangular portion

of the matrix is only referenced. The matrix can also be of class symmetric in which case only the selected triangular part is used. Matrix A must be of full rank, that is, the matrix must be invertible. The linear operator $op()$ can define the transposition or Hermitian transposition operations. By default, no transposition is performed. The right-hand-side matrix B and the solution matrix X are dense and must be of the correct size, that is m by n, see `ldb` and `ldx` input parameters for further details.

Explicitly, this kernel solves

$$op(A)\ X = \alpha\ B, \text{ with solution } X = \alpha\ (op(A)^{-1})\ B,$$

where

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none,} \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose,} \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose.} \end{cases}$$

If a linear operator is applied then, the possible problems solved are

$$A^T\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-T}\ B, \text{ and }\ A^H\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-H}\ B.$$

Note

1. If the matrix descriptor `descr` specifies that the matrix A is to be regarded as having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are considered to all be ones.
2. If the matrix A is described as upper triangular, then only the upper triangular portion of the matrix is referenced. Conversely, if the matrix A is described lower triangular, then only the lower triangular portion of the matrix is used.
3. This set of APIs allocates work array of size m for each case where the matrices B or X are stored in row-major format (aoclsparse_order_row).
4. A subset of kernels are parallel (on parallel builds) and can be expected potential acceleration in the solve. These kernels are available when both dense matrices X and B are stored in column-major format (aoclsparse_order_column) and thread count is greater than 1 on a parallel build.
5. There is `_kid` (Kernel ID) variation of TRSM, namely with a suffix of `_kid`, this solver allows to choose which underlying TRSV kernels to use (if possible). Currently, all the existing `aoclsparse_?trsm` kernels are supported.
6. This routine supports only sparse matrices in CSR format.

Parameters

`in`	*trans*	matrix operation to perform on A. Possible values are aoclsparse_operation_none, aoclsparse_operation_transpose, and aoclsparse_operation_conjugate_transpose.
`in`	*alpha*	scalar α.
`in`	*A*	sparse matrix A of size m.
`in`	*descr*	descriptor of the sparse matrix A.
`in`	*order*	storage order of dense matrices B and X. Possible options are aoclsparse_order_row and aoclsparse_order_column.
`in`	*B*	dense matrix, potentially rectangular, of size $m \times n$.
`in`	*n*	n, number of columns of the dense matrix B.

Parameters

in	*ldb*	leading dimension of B. Eventhough the matrix B is considered of size $m \times n$, its memory layout may correspond to a larger matrix (`ldb` by $N > n$) in which only the submatrix B is of interest. In this case, this parameter provides means to access the correct elements of B within the larger layout. **matrix layout** / **row count** / **column count** aoclsparse_order_row / m / `ldb` with `ldb` $\geq n$ aoclsparse_order_column / `ldb` with `ldb` $\geq m$ / n
out	*X*	solution matrix X, dense and potentially rectangular matrix of size $m \times n$.
in	*ldx*	leading dimension of X. Eventhough the matrix X is considered of size $m \times n$, its memory layout may correspond to a larger matrix (`ldx` by $N > n$) in which only the submatrix X is of interest. In this case, this parameter provides means to access the correct elements of X within the larger layout. **matrix layout** / **row count** / **column count** aoclsparse_order_row / m / `ldx` with `ldx` $\geq n$ aoclsparse_order_column / `ldx` with `ldx` $\geq m$ / n
in	*kid*	kernel ID, hints a request on which kernel to use (see notes).

Return values

aoclsparse_status_success	indicates that the operation completed successfully.
aoclsparse_status_invalid_size	informs that either `m`, `n`, `nnz`, `ldb` or `ldx` is invalid.
aoclsparse_status_invalid_pointer	informs that either `descr`, `alpha`, `A`, `B`, or `X` pointer is invalid.
aoclsparse_status_not_implemented	this error occurs when the provided matrix aoclsparse_matrix_type is aoclsparse_matrix_type_general or aoclsparse_matrix_type_hermitian or when matrix `A` is not in CSR format.

3.4.2.63 aoclsparse_strsm_kid()

```
DLL_PUBLIC aoclsparse_status aoclsparse_strsm_kid (
        const aoclsparse_operation trans,
        const float alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        aoclsparse_order order,
        const float * B,
        aoclsparse_int n,
        aoclsparse_int ldb,
        float * X,
        aoclsparse_int ldx,
        const aoclsparse_int kid )
```

Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

`aoclsparse_?trsm` solves a sparse triangular linear system of equations with multiple right hand sides, of the form

$$op(A)\; X = \alpha B,$$

where A is a sparse matrix of size m, $op()$ is a linear operator, X and B are rectangular dense matrices of appropriate size, while α is a scalar. The sparse matrix A can be interpreted either as a lower triangular or upper triangular. This is indicated by `fill_mode` from the matrix descriptor `descr` where either upper or lower triangular portion

of the matrix is only referenced. The matrix can also be of class symmetric in which case only the selected triangular part is used. Matrix A must be of full rank, that is, the matrix must be invertible. The linear operator $op()$ can define the transposition or Hermitian transposition operations. By default, no transposition is performed. The right-hand-side matrix B and the solution matrix X are dense and must be of the correct size, that is m by n, see `ldb` and `ldx` input parameters for further details.

Explicitly, this kernel solves

$$op(A)\ X = \alpha\ B, \text{ with solution } X = \alpha\ (op(A)^{-1})\ B,$$

where

$$op(A) = \begin{cases} A, & \text{if trans = aoclsparse\_operation\_none,} \\ A^T, & \text{if trans = aoclsparse\_operation\_transpose,} \\ A^H, & \text{if trans = aoclsparse\_operation\_conjugate\_transpose.} \end{cases}$$

If a linear operator is applied then, the possible problems solved are

$$A^T\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-T}\ B, \text{ and }\ A^H\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-H}\ B.$$

Note

1. If the matrix descriptor `descr` specifies that the matrix A is to be regarded as having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are considered to all be ones.
2. If the matrix A is described as upper triangular, then only the upper triangular portion of the matrix is referenced. Conversely, if the matrix A is described lower triangular, then only the lower triangular portion of the matrix is used.
3. This set of APIs allocates work array of size m for each case where the matrices B or X are stored in row-major format (aoclsparse_order_row).
4. A subset of kernels are parallel (on parallel builds) and can be expected potential acceleration in the solve. These kernels are available when both dense matrices X and B are stored in column-major format (aoclsparse_order_column) and thread count is greater than 1 on a parallel build.
5. There is `_kid` (Kernel ID) variation of TRSM, namely with a suffix of `_kid`, this solver allows to choose which underlying TRSV kernels to use (if possible). Currently, all the existing `aoclsparse_?trsm` kernels are supported.
6. This routine supports only sparse matrices in CSR format.

Parameters

`in`	*trans*	matrix operation to perform on A. Possible values are aoclsparse_operation_none, aoclsparse_operation_transpose, and aoclsparse_operation_conjugate_transpose.
`in`	*alpha*	scalar α.
`in`	*A*	sparse matrix A of size m.
`in`	*descr*	descriptor of the sparse matrix A.
`in`	*order*	storage order of dense matrices B and X. Possible options are aoclsparse_order_row and aoclsparse_order_column.
`in`	*B*	dense matrix, potentially rectangular, of size $m \times n$.
`in`	*n*	n, number of columns of the dense matrix B.

Parameters

in	*ldb*	leading dimension of B. Eventhough the matrix B is considered of size $m \times n$, its memory layout may correspond to a larger matrix (`ldb` by $N > n$) in which only the submatrix B is of interest. In this case, this parameter provides means to access the correct elements of B within the larger layout. **matrix layout** / **row count** / **column count** aoclsparse_order_row / m / `ldb` with `ldb` $\geq n$ aoclsparse_order_column / `ldb` with `ldb` $\geq m$ / n
out	*X*	solution matrix X, dense and potentially rectangular matrix of size $m \times n$.
in	*ldx*	leading dimension of X. Eventhough the matrix X is considered of size $m \times n$, its memory layout may correspond to a larger matrix (`ldx` by $N > n$) in which only the submatrix X is of interest. In this case, this parameter provides means to access the correct elements of X within the larger layout. **matrix layout** / **row count** / **column count** aoclsparse_order_row / m / `ldx` with `ldx` $\geq n$ aoclsparse_order_column / `ldx` with `ldx` $\geq m$ / n
in	*kid*	kernel ID, hints a request on which kernel to use (see notes).

Return values

aoclsparse_status_success	indicates that the operation completed successfully.
aoclsparse_status_invalid_size	informs that either `m`, `n`, `nnz`, `ldb` or `ldx` is invalid.
aoclsparse_status_invalid_pointer	informs that either `descr`, `alpha`, `A`, `B`, or `X` pointer is invalid.
aoclsparse_status_not_implemented	this error occurs when the provided matrix aoclsparse_matrix_type is aoclsparse_matrix_type_general or aoclsparse_matrix_type_hermitian or when matrix `A` is not in CSR format.

3.4.2.64 aoclsparse_dtrsm_kid()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dtrsm_kid (
        const aoclsparse_operation trans,
        const double alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        aoclsparse_order order,
        const double * B,
        aoclsparse_int n,
        aoclsparse_int ldb,
        double * X,
        aoclsparse_int ldx,
        const aoclsparse_int kid )
```

Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

`aoclsparse_?trsm` solves a sparse triangular linear system of equations with multiple right hand sides, of the form

$$op(A)\ X = \alpha B,$$

where A is a sparse matrix of size m, $op()$ is a linear operator, X and B are rectangular dense matrices of appropriate size, while α is a scalar. The sparse matrix A can be interpreted either as a lower triangular or upper triangular. This is indicated by `fill_mode` from the matrix descriptor `descr` where either upper or lower triangular portion

of the matrix is only referenced. The matrix can also be of class symmetric in which case only the selected triangular part is used. Matrix A must be of full rank, that is, the matrix must be invertible. The linear operator $op()$ can define the transposition or Hermitian transposition operations. By default, no transposition is performed. The right-hand-side matrix B and the solution matrix X are dense and must be of the correct size, that is m by n, see `ldb` and `ldx` input parameters for further details.

Explicitly, this kernel solves

$$op(A)\ X = \alpha\ B, \text{ with solution } X = \alpha\ (op(A)^{-1})\ B,$$

where

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none}, \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose}, \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose}. \end{cases}$$

If a linear operator is applied then, the possible problems solved are

$$A^T\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-T}\ B, \text{ and } \ A^H\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-H}\ B.$$

Note

1. If the matrix descriptor `descr` specifies that the matrix A is to be regarded as having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are considered to all be ones.
2. If the matrix A is described as upper triangular, then only the upper triangular portion of the matrix is referenced. Conversely, if the matrix A is described lower triangular, then only the lower triangular portion of the matrix is used.
3. This set of APIs allocates work array of size m for each case where the matrices B or X are stored in row-major format (aoclsparse_order_row).
4. A subset of kernels are parallel (on parallel builds) and can be expected potential acceleration in the solve. These kernels are available when both dense matrices X and B are stored in column-major format (aoclsparse_order_column) and thread count is greater than 1 on a parallel build.
5. There is `_kid` (Kernel ID) variation of TRSM, namely with a suffix of `_kid`, this solver allows to choose which underlying TRSV kernels to use (if possible). Currently, all the existing `aoclsparse_?trsm` kernels are supported.
6. This routine supports only sparse matrices in CSR format.

Parameters

in	*trans*	matrix operation to perform on A. Possible values are aoclsparse_operation_none, aoclsparse_operation_transpose, and aoclsparse_operation_conjugate_transpose.
in	*alpha*	scalar α.
in	*A*	sparse matrix A of size m.
in	*descr*	descriptor of the sparse matrix A.
in	*order*	storage order of dense matrices B and X. Possible options are aoclsparse_order_row and aoclsparse_order_column.
in	*B*	dense matrix, potentially rectangular, of size $m \times n$.
in	*n*	n, number of columns of the dense matrix B.

Parameters

<table>
<tr><td>in</td><td>ldb</td><td>leading dimension of B. Eventhough the matrix B is considered of size $m \times n$, its memory layout may correspond to a larger matrix (ldb by $N > n$) in which only the submatrix B is of interest. In this case, this parameter provides means to access the correct elements of B within the larger layout.
<table>
<tr><th>matrix layout</th><th>row count</th><th>column count</th></tr>
<tr><td>aoclsparse_order_row</td><td>m</td><td>ldb with ldb $\geq n$</td></tr>
<tr><td>aoclsparse_order_column</td><td>ldb with ldb $\geq m$</td><td>n</td></tr>
</table></td></tr>
<tr><td>out</td><td>X</td><td>solution matrix X, dense and potentially rectangular matrix of size $m \times n$.</td></tr>
<tr><td>in</td><td>ldx</td><td>leading dimension of X. Eventhough the matrix X is considered of size $m \times n$, its memory layout may correspond to a larger matrix (ldx by $N > n$) in which only the submatrix X is of interest. In this case, this parameter provides means to access the correct elements of X within the larger layout.
<table>
<tr><th>matrix layout</th><th>row count</th><th>column count</th></tr>
<tr><td>aoclsparse_order_row</td><td>m</td><td>ldx with ldx $\geq n$</td></tr>
<tr><td>aoclsparse_order_column</td><td>ldx with ldx $\geq m$</td><td>n</td></tr>
</table></td></tr>
<tr><td>in</td><td>kid</td><td>kernel ID, hints a request on which kernel to use (see notes).</td></tr>
</table>

Return values

aoclsparse_status_success	indicates that the operation completed successfully.
aoclsparse_status_invalid_size	informs that either m, n, nnz, ldb or ldx is invalid.
aoclsparse_status_invalid_pointer	informs that either descr, alpha, A, B, or X pointer is invalid.
aoclsparse_status_not_implemented	this error occurs when the provided matrix aoclsparse_matrix_type is aoclsparse_matrix_type_general or aoclsparse_matrix_type_hermitian or when matrix A is not in CSR format.

3.4.2.65 aoclsparse_ctrsm_kid()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ctrsm_kid (
    aoclsparse_operation trans,
    const aoclsparse_float_complex alpha,
    aoclsparse_matrix A,
    const aoclsparse_mat_descr descr,
    aoclsparse_order order,
    const aoclsparse_float_complex * B,
    aoclsparse_int n,
    aoclsparse_int ldb,
    aoclsparse_float_complex * X,
    aoclsparse_int ldx,
    const aoclsparse_int kid )
```

Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

aoclsparse_?trsm solves a sparse triangular linear system of equations with multiple right hand sides, of the form

$$op(A)\ X = \alpha B,$$

where A is a sparse matrix of size m, $op()$ is a linear operator, X and B are rectangular dense matrices of appropriate size, while α is a scalar. The sparse matrix A can be interpreted either as a lower triangular or upper triangular. This is indicated by fill_mode from the matrix descriptor descr where either upper or lower triangular portion

of the matrix is only referenced. The matrix can also be of class symmetric in which case only the selected triangular part is used. Matrix A must be of full rank, that is, the matrix must be invertible. The linear operator $op()$ can define the transposition or Hermitian transposition operations. By default, no transposition is performed. The right-hand-side matrix B and the solution matrix X are dense and must be of the correct size, that is m by n, see `ldb` and `ldx` input parameters for further details.

Explicitly, this kernel solves

$$op(A)\; X = \alpha\; B, \text{ with solution } X = \alpha\; (op(A)^{-1})\; B,$$

where

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none}, \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose}, \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose}. \end{cases}$$

If a linear operator is applied then, the possible problems solved are

$$A^T\; X = \alpha\; B, \text{ with solution } X = \alpha\; A^{-T}\; B, \text{ and } \; A^H\; X = \alpha\; B, \text{ with solution } X = \alpha\; A^{-H}\; B.$$

Note

1. If the matrix descriptor `descr` specifies that the matrix A is to be regarded as having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are considered to all be ones.
2. If the matrix A is described as upper triangular, then only the upper triangular portion of the matrix is referenced. Conversely, if the matrix A is described lower triangular, then only the lower triangular portion of the matrix is used.
3. This set of APIs allocates work array of size m for each case where the matrices B or X are stored in row-major format (aoclsparse_order_row).
4. A subset of kernels are parallel (on parallel builds) and can be expected potential acceleration in the solve. These kernels are available when both dense matrices X and B are stored in column-major format (aoclsparse_order_column) and thread count is greater than 1 on a parallel build.
5. There is `_kid` (Kernel ID) variation of TRSM, namely with a suffix of `_kid`, this solver allows to choose which underlying TRSV kernels to use (if possible). Currently, all the existing `aoclsparse_?trsm` kernels are supported.
6. This routine supports only sparse matrices in CSR format.

Parameters

in	*trans*	matrix operation to perform on A. Possible values are aoclsparse_operation_none, aoclsparse_operation_transpose, and aoclsparse_operation_conjugate_transpose.
in	*alpha*	scalar α.
in	*A*	sparse matrix A of size m.
in	*descr*	descriptor of the sparse matrix A.
in	*order*	storage order of dense matrices B and X. Possible options are aoclsparse_order_row and aoclsparse_order_column.
in	*B*	dense matrix, potentially rectangular, of size $m \times n$.
in	*n*	n, number of columns of the dense matrix B.

Parameters

in	*ldb*	leading dimension of B. Eventhough the matrix B is considered of size $m \times n$, its memory layout may correspond to a larger matrix (`ldb` by $N > n$) in which only the submatrix B is of interest. In this case, this parameter provides means to access the correct elements of B within the larger layout. **matrix layout** / **row count** / **column count** aoclsparse_order_row / m / `ldb` with `ldb` $\geq n$ aoclsparse_order_column / `ldb` with `ldb` $\geq m$ / n
out	*X*	solution matrix X, dense and potentially rectangular matrix of size $m \times n$.
in	*ldx*	leading dimension of X. Eventhough the matrix X is considered of size $m \times n$, its memory layout may correspond to a larger matrix (`ldx` by $N > n$) in which only the submatrix X is of interest. In this case, this parameter provides means to access the correct elements of X within the larger layout. **matrix layout** / **row count** / **column count** aoclsparse_order_row / m / `ldx` with `ldx` $\geq n$ aoclsparse_order_column / `ldx` with `ldx` $\geq m$ / n
in	*kid*	kernel ID, hints a request on which kernel to use (see notes).

Return values

aoclsparse_status_success	indicates that the operation completed successfully.
aoclsparse_status_invalid_size	informs that either `m`, `n`, `nnz`, `ldb` or `ldx` is invalid.
aoclsparse_status_invalid_pointer	informs that either `descr`, `alpha`, `A`, `B`, or `X` pointer is invalid.
aoclsparse_status_not_implemented	this error occurs when the provided matrix aoclsparse_matrix_type is aoclsparse_matrix_type_general or aoclsparse_matrix_type_hermitian or when matrix `A` is not in CSR format.

3.4.2.66 aoclsparse_ztrsm_kid()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ztrsm_kid (
        aoclsparse_operation trans,
        const aoclsparse_double_complex alpha,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        aoclsparse_order order,
        const aoclsparse_double_complex * B,
        aoclsparse_int n,
        aoclsparse_int ldb,
        aoclsparse_double_complex * X,
        aoclsparse_int ldx,
        const aoclsparse_int kid )
```

Solve sparse triangular linear system of equations with multiple right hand sides for real/complex single and double data precisions.

`aoclsparse_?trsm` solves a sparse triangular linear system of equations with multiple right hand sides, of the form

$$op(A)\ X = \alpha B,$$

where A is a sparse matrix of size m, $op()$ is a linear operator, X and B are rectangular dense matrices of appropriate size, while α is a scalar. The sparse matrix A can be interpreted either as a lower triangular or upper triangular. This is indicated by `fill_mode` from the matrix descriptor `descr` where either upper or lower triangular portion

of the matrix is only referenced. The matrix can also be of class symmetric in which case only the selected triangular part is used. Matrix A must be of full rank, that is, the matrix must be invertible. The linear operator $op()$ can define the transposition or Hermitian transposition operations. By default, no transposition is performed. The right-hand-side matrix B and the solution matrix X are dense and must be of the correct size, that is m by n, see `ldb` and `ldx` input parameters for further details.

Explicitly, this kernel solves

$$op(A)\ X = \alpha\ B, \text{ with solution } X = \alpha\ (op(A)^{-1})\ B,$$

where

$$op(A) = \begin{cases} A, & \text{if trans} = \text{aoclsparse\_operation\_none}, \\ A^T, & \text{if trans} = \text{aoclsparse\_operation\_transpose}, \\ A^H, & \text{if trans} = \text{aoclsparse\_operation\_conjugate\_transpose}. \end{cases}$$

If a linear operator is applied then, the possible problems solved are

$$A^T\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-T}\ B, \text{ and }\ A^H\ X = \alpha\ B, \text{ with solution } X = \alpha\ A^{-H}\ B.$$

Note

1. If the matrix descriptor `descr` specifies that the matrix A is to be regarded as having a unitary diagonal, then the main diagonal entries of matrix A are not accessed and are considered to all be ones.
2. If the matrix A is described as upper triangular, then only the upper triangular portion of the matrix is referenced. Conversely, if the matrix A is described lower triangular, then only the lower triangular portion of the matrix is used.
3. This set of APIs allocates work array of size m for each case where the matrices B or X are stored in row-major format (aoclsparse_order_row).
4. A subset of kernels are parallel (on parallel builds) and can be expected potential acceleration in the solve. These kernels are available when both dense matrices X and B are stored in column-major format (aoclsparse_order_column) and thread count is greater than 1 on a parallel build.
5. There is `_kid` (Kernel ID) variation of TRSM, namely with a suffix of `_kid`, this solver allows to choose which underlying TRSV kernels to use (if possible). Currently, all the existing `aoclsparse_?trsm` kernels are supported.
6. This routine supports only sparse matrices in CSR format.

Parameters

`in`	*trans*	matrix operation to perform on A. Possible values are aoclsparse_operation_none, aoclsparse_operation_transpose, and aoclsparse_operation_conjugate_transpose.
`in`	*alpha*	scalar α.
`in`	*A*	sparse matrix A of size m.
`in`	*descr*	descriptor of the sparse matrix A.
`in`	*order*	storage order of dense matrices B and X. Possible options are aoclsparse_order_row and aoclsparse_order_column.
`in`	*B*	dense matrix, potentially rectangular, of size $m \times n$.
`in`	*n*	n, number of columns of the dense matrix B.

Parameters

in	*ldb*	leading dimension of B. Eventhough the matrix B is considered of size $m \times n$, its memory layout may correspond to a larger matrix (ldb by $N > n$) in which only the submatrix B is of interest. In this case, this parameter provides means to access the correct elements of B within the larger layout. **matrix layout** / **row count** / **column count** aoclsparse_order_row / m / ldb with ldb $\geq n$ aoclsparse_order_column / ldb with ldb $\geq m$ / n
out	*X*	solution matrix X, dense and potentially rectangular matrix of size $m \times n$.
in	*ldx*	leading dimension of X. Eventhough the matrix X is considered of size $m \times n$, its memory layout may correspond to a larger matrix (ldx by $N > n$) in which only the submatrix X is of interest. In this case, this parameter provides means to access the correct elements of X within the larger layout. **matrix layout** / **row count** / **column count** aoclsparse_order_row / m / ldx with ldx $\geq n$ aoclsparse_order_column / ldx with ldx $\geq m$ / n
in	*kid*	kernel ID, hints a request on which kernel to use (see notes).

Return values

aoclsparse_status_success	indicates that the operation completed successfully.
aoclsparse_status_invalid_size	informs that either m, n, nnz, ldb or ldx is invalid.
aoclsparse_status_invalid_pointer	informs that either descr, alpha, A, B, or X pointer is invalid.
aoclsparse_status_not_implemented	this error occurs when the provided matrix aoclsparse_matrix_type is aoclsparse_matrix_type_general or aoclsparse_matrix_type_hermitian or when matrix A is not in CSR format.

3.4.2.67 aoclsparse_sp2m()

```
DLL_PUBLIC aoclsparse_status aoclsparse_sp2m (
    aoclsparse_operation opA,
    const aoclsparse_mat_descr descrA,
    const aoclsparse_matrix A,
    aoclsparse_operation opB,
    const aoclsparse_mat_descr descrB,
    const aoclsparse_matrix B,
    const aoclsparse_request request,
    aoclsparse_matrix * C )
```

Sparse matrix Sparse matrix multiplication for real and complex datatypes.

aoclsparse_sp2m multiplies two sparse matrices in CSR storage format. The result is stored in a newly allocated sparse matrix in CSR format, such that

$$C := op(A) \cdot op(B),$$

with

$$op(A) = \begin{cases} A, & \text{if opA} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if opA} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if opA} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

and

$$op(B) = \begin{cases} B, & \text{if opB} = \text{aoclsparse\_operation\_none} \\ B^T, & \text{if opB} = \text{aoclsparse\_operation\_transpose} \\ B^H, & \text{if opB} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

where A is a $m \times k$ matrix , B is a $k \times n$ matrix, resulting in $m \times n$ matrix C, for opA and opB = aoclsparse_operation_none. A is a $k \times m$ matrix when opA = aoclsparse_operation_transpose or aoclsparse_operation_conjugate_transpose and B is a $n \times k$ matrix when opB = aoclsparse_operation_transpose or aoclsparse_operation_conjugate_transpose

aoclsparse_sp2m can be run in single-stage or two-stage. The single-stage algorithm allocates and computes the entire output matrix in a single stage aoclsparse_stage_full_computation. Whereas, in two-stage algorithm, the first stage aoclsparse_stage_nnz_count allocates memory for the output matrix and computes the number of entries of the matrix. The second stage aoclsparse_stage_finalize computes column indices of non-zero elements and values of the output matrix. The second stage has to be invoked only after the first stage. But, it can be also be invoked multiple times consecutively when the sparsity structure of input matrices remains unchanged, with only the values getting updated.

Parameters

in	*opA*	matrix A operation type.
in	*descrA*	descriptor of the sparse CSR matrix A. Currently, only aoclsparse_matrix_type_general is supported.
in	*A*	sparse CSR matrix A .
in	*opB*	matrix B operation type.
in	*descrB*	descriptor of the sparse CSR matrix B. Currently, only aoclsparse_matrix_type_general is supported.
in	*B*	sparse CSR matrix B .
in	*request*	Specifies full computation or two-stage algorithm aoclsparse_stage_nnz_count , Only rowIndex array of the CSR matrix is computed internally. The output sparse CSR matrix can be extracted to measure the memory required for full operation. aoclsparse_stage_finalize . Finalize computation of remaining output arrays (column indices and values of output matrix entries) . Has to be called only after aoclsparse_sp2m call with aoclsparse_stage_nnz_count parameter. aoclsparse_stage_full_computation . Perform the entire computation in a single step.
out	**C*	Pointer to sparse CSR matrix C . Matrix C arrays will always have zero-based indexing, irrespective of matrix A or matrix B being one-based or zero-based indexing. The column indices of the output matrix in CSR format can appear unsorted.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	`descrA`, `descrB`, `A`, `B`, `C` is invalid.
aoclsparse_status_invalid_size	input size parameters contain an invalid value.
aoclsparse_status_invalid_value	input parameters contain an invalid value.
aoclsparse_status_wrong_type	A and B matrix datatypes dont match.
aoclsparse_status_memory_error	Memory allocation failure.
aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_general or input matrices `A` or `B` is not in CSR format

Example

Shows multiplication of 2 sparse matrices to give a newly allocated sparse matrix

```
aoclsparse_matrix  A;
```

```
aoclsparse_create_dcsr(&A, base, M, K, nnz_A, csr_row_ptr_A.data(), csr_col_ind_A.data(),
 csr_val_A.data());
aoclsparse_matrix  B;
aoclsparse_create_dcsr(&B, base, K, N, nnz_B, csr_row_ptr_B.data(), csr_col_ind_B.data(),
 csr_val_B.data());
aoclsparse_matrix  C = NULL;
aoclsparse_int *csr_row_ptr_C = NULL;
aoclsparse_int *csr_col_ind_C = NULL;
double             *csr_val_C = NULL;
aoclsparse_int C_M, C_N;
aoclsparse_status status;
request =  aoclsparse_stage_full_computation;
status = aoclsparse_sp2m(opA,
        descrA,
        A,
        opB,
        descrB,
        B,
        request,
        &C);
aoclsparse_export_dcsr(C, &base, &C_M, &C_N, &nnz_C, &csr_row_ptr_C, &csr_col_ind_C, (void
 **)&csr_val_C);
```

3.4.2.68 aoclsparse_spmm() `DLL_PUBLIC aoclsparse_status aoclsparse_spmm ( aoclsparse_operation opA, const aoclsparse_matrix A, const aoclsparse_matrix B, aoclsparse_matrix * C )`

Sparse matrix Sparse matrix multiplication for real and complex datatypes.

`aoclsparse_spmm` multiplies two sparse matrices in CSR storage format. The result is stored in a newly allocated sparse matrix in CSR format, such that

$$C := op(A) \cdot B,$$

with

$$op(A) = \begin{cases} A, & \text{if opA} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if opA} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if opA} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

where A is a $m \times k$ matrix , B is a $k \times n$ matrix, resulting in $m \times n$ matrix C, for opA = aoclsparse_operation_none. A is a $k \times m$ matrix when opA = aoclsparse_operation_transpose or aoclsparse_operation_conjugate_transpose

Parameters

in	*opA*	matrix A operation type.
in	*A*	sparse CSR matrix A.
in	*B*	sparse CSR matrix B.
out	*$*$C*	Pointer to sparse CSR matrix C . Matrix C arrays will always have zero-based indexing, irrespective of matrix A or matrix B being one-based or zero-based indexing. The column indices of the output matrix in CSR format can appear unsorted.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_pointer	A, B, C is invalid.
aoclsparse_status_invalid_size	input size parameters contain an invalid value.
aoclsparse_status_invalid_value	input parameters contain an invalid value.
aoclsparse_status_wrong_type	A and B matrix datatypes dont match.

Return values

aoclsparse_status_memory_error	Memory allocation failure.
aoclsparse_status_not_implemented	Input matrices A or \B is not in CSR format

Example

Shows multiplication of 2 sparse matrices to give a newly allocated sparse matrix

```
aoclsparse_matrix  A;
aoclsparse_create_dcsr(&A, base, M, K, nnz_A, csr_row_ptr_A.data(), csr_col_ind_A.data(),
 csr_val_A.data());
aoclsparse_matrix  B;
aoclsparse_create_dcsr(&B, base, K, N, nnz_B, csr_row_ptr_B.data(), csr_col_ind_B.data(),
 csr_val_B.data());
aoclsparse_matrix  C = NULL;
aoclsparse_int *csr_row_ptr_C = NULL;
aoclsparse_int *csr_col_ind_C = NULL;
double          *csr_val_C = NULL;
aoclsparse_int C_M, C_N;
aoclsparse_status status;
status = aoclsparse_spmm(opA,
        A,
        B,
        &C);
aoclsparse_export_dcsr(C, &base, &C_M, &C_N, &nnz_C, &csr_row_ptr_C, &csr_col_ind_C, (void
 **)&csr_val_C);
```

3.4.2.69 aoclsparse_scsrmm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsrmm (
        aoclsparse_operation op,
        const float alpha,
        const aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        aoclsparse_order order,
        const float * B,
        aoclsparse_int n,
        aoclsparse_int ldb,
        const float beta,
        float * C,
        aoclsparse_int ldc )
```

Sparse matrix dense matrix multiplication using CSR storage format.

`aoclsparse_`(s/d/c/z)csrmm multiplies a scalar α with a sparse $m \times k$ matrix A, defined in CSR storage format, and a dense $k \times n$ matrix B and adds the result to the dense $m \times n$ matrix C that is multiplied by a scalar β, such that

$$C := \alpha \cdot op(A) \cdot B + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & \text{if trans\_A = aoclsparse\_operation\_none} \\ A^T, & \text{if trans\_A = aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans\_A = aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

```
for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];
        for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
        {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}
```

Parameters

in	*Op*	Matrix A operation type.
in	*Alpha*	Scalar α.
in	*A*	Sparse CSR matrix A structure.
in	*descr*	descriptor of the sparse CSR matrix A. Currently, only aoclsparse_matrix_type_general is supported. Both, base-zero and base-one input arrays of CSR matrix are supported
in	*Order*	Aoclsparse_order_row/aoclsparse_order_column for dense matrix
in	*B*	Array of dimension $ldb \times n$ or $ldb \times k$.
in	*N*	Number of columns of the dense matrix B and C.
in	*Ldb*	Leading dimension of B, must be at least $\max(1, k)$ ($op(A) = A$) or $\max(1, m)$ ($op(A) = A^T$ or $op(A) = A^H$).
in	*Beta*	Scalar β.
in,out	*C*	Array of dimension $ldc \times n$.
in	*Ldc*	Leading dimension of C, must be at least $\max(1, m)$ ($op(A) = A$) or $\max(1, k)$ ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_size	The value of `m`, `n`, `k`, `nnz`, `ldb` or `ldc` is invalid.
aoclsparse_status_invalid_pointer	The pointer `descr`, `A`, `B`, or `C` is invalid.
aoclsparse_status_invalid_value	The value of `descr->base`, `A->base` is invalid.
aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_general or input matrix `A` is not in CSR format

3.4.2.70 aoclsparse_dcsrmm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsrmm (
    aoclsparse_operation op,
    const double alpha,
    const aoclsparse_matrix A,
    const aoclsparse_mat_descr descr,
    aoclsparse_order order,
    const double * B,
    aoclsparse_int n,
    aoclsparse_int ldb,
    const double beta,
    double * C,
    aoclsparse_int ldc )
```

Sparse matrix dense matrix multiplication using CSR storage format.

`aoclsparse_`(s/d/c/z)csrmm multiplies a scalar α with a sparse $m \times k$ matrix A, defined in CSR storage format, and a dense $k \times n$ matrix B and adds the result to the dense $m \times n$ matrix C that is multiplied by a scalar β, such that

$$C := \alpha \cdot op(A) \cdot B + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & \text{if trans\_A} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans\_A} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans\_A} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

```
for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];
        for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
        {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}
```

Parameters

in	*Op*	Matrix A operation type.
in	*Alpha*	Scalar α.
in	*A*	Sparse CSR matrix A structure.
in	*descr*	descriptor of the sparse CSR matrix A. Currently, only aoclsparse_matrix_type_general is supported. Both, base-zero and base-one input arrays of CSR matrix are supported
in	*Order*	Aoclsparse_order_row/aoclsparse_order_column for dense matrix
in	*B*	Array of dimension $ldb \times n$ or $ldb \times k$.
in	*N*	Number of columns of the dense matrix B and C.
in	*Ldb*	Leading dimension of B, must be at least $\max(1, k)$ ($op(A) = A$) or $\max(1, m)$ ($op(A) = A^T$ or $op(A) = A^H$).
in	*Beta*	Scalar β.
in,out	*C*	Array of dimension $ldc \times n$.
in	*Ldc*	Leading dimension of C, must be at least $\max(1, m)$ ($op(A) = A$) or $\max(1, k)$ ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_size	The value of `m`, `n`, `k`, `nnz`, `ldb` or `ldc` is invalid.
aoclsparse_status_invalid_pointer	The pointer `descr`, `A`, `B`, or `C` is invalid.
aoclsparse_status_invalid_value	The value of `descr->base`, `A->base` is invalid.
aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_general or input matrix `A` is not in CSR format

3.4.2.71 aoclsparse_ccsrmm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_ccsrmm (
    aoclsparse_operation op,
    const aoclsparse_float_complex alpha,
    const aoclsparse_matrix A,
    const aoclsparse_mat_descr descr,
    aoclsparse_order order,
    const aoclsparse_float_complex * B,
    aoclsparse_int n,
    aoclsparse_int ldb,
    const aoclsparse_float_complex beta,
    aoclsparse_float_complex * C,
    aoclsparse_int ldc )
```

Sparse matrix dense matrix multiplication using CSR storage format.

aoclsparse_(s/d/c/z)csrmm multiplies a scalar α with a sparse $m \times k$ matrix A, defined in CSR storage format, and a dense $k \times n$ matrix B and adds the result to the dense $m \times n$ matrix C that is multiplied by a scalar β, such that

$$C := \alpha \cdot op(A) \cdot B + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & \text{if trans\_A = aoclsparse\_operation\_none} \\ A^T, & \text{if trans\_A = aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans\_A = aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

```
for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];
        for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
        {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}
```

Parameters

in	*Op*	Matrix A operation type.
in	*Alpha*	Scalar α.
in	*A*	Sparse CSR matrix A structure.
in	*descr*	descriptor of the sparse CSR matrix A. Currently, only aoclsparse_matrix_type_general is supported. Both, base-zero and base-one input arrays of CSR matrix are supported
in	*Order*	Aoclsparse_order_row/aoclsparse_order_column for dense matrix
in	*B*	Array of dimension $ldb \times n$ or $ldb \times k$.
in	*N*	Number of columns of the dense matrix B and C.
in	*Ldb*	Leading dimension of B, must be at least $\max(1, k)$ ($op(A) = A$) or $\max(1, m)$ ($op(A) = A^T$ or $op(A) = A^H$).
in	*Beta*	Scalar β.
in,out	*C*	Array of dimension $ldc \times n$.
in	*Ldc*	Leading dimension of C, must be at least $\max(1, m)$ ($op(A) = A$) or $\max(1, k)$ ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_size	The value of `m`, `n`, `k`, `nnz`, `ldb` or `ldc` is invalid.
aoclsparse_status_invalid_pointer	The pointer `descr`, `A`, `B`, or `C` is invalid.
aoclsparse_status_invalid_value	The value of `descr->base`, `A->base` is invalid.
aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_general or input matrix `A` is not in CSR format

3.4.2.72 aoclsparse_zcsrmm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zcsrmm (
            aoclsparse_operation op,
            const aoclsparse_double_complex alpha,
            const aoclsparse_matrix A,
            const aoclsparse_mat_descr descr,
            aoclsparse_order order,
```

```
const aoclsparse_double_complex * B,
aoclsparse_int n,
aoclsparse_int ldb,
const aoclsparse_double_complex beta,
aoclsparse_double_complex * C,
aoclsparse_int ldc )
```

Sparse matrix dense matrix multiplication using CSR storage format.

`aoclsparse_`(s/d/c/z)csrmm multiplies a scalar α with a sparse $m \times k$ matrix A, defined in CSR storage format, and a dense $k \times n$ matrix B and adds the result to the dense $m \times n$ matrix C that is multiplied by a scalar β, such that

$$C := \alpha \cdot op(A) \cdot B + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & \text{if trans\_A} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans\_A} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans\_A} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

```
for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];
        for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
        {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}
```

Parameters

in	*Op*	Matrix A operation type.
in	*Alpha*	Scalar α.
in	*A*	Sparse CSR matrix A structure.
in	*descr*	descriptor of the sparse CSR matrix A. Currently, only aoclsparse_matrix_type_general is supported. Both, base-zero and base-one input arrays of CSR matrix are supported
in	*Order*	Aoclsparse_order_row/aoclsparse_order_column for dense matrix
in	*B*	Array of dimension $ldb \times n$ or $ldb \times k$.
in	*N*	Number of columns of the dense matrix B and C.
in	*Ldb*	Leading dimension of B, must be at least $\max(1, k)$ ($op(A) = A$) or $\max(1, m)$ ($op(A) = A^T$ or $op(A) = A^H$).
in	*Beta*	Scalar β.
in,out	*C*	Array of dimension $ldc \times n$.
in	*Ldc*	Leading dimension of C, must be at least $\max(1, m)$ ($op(A) = A$) or $\max(1, k)$ ($op(A) = A^T$ or $op(A) = A^H$).

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_size	The value of `m`, `n`, `k`, `nnz`, `ldb` or `ldc` is invalid.
aoclsparse_status_invalid_pointer	The pointer `descr`, `A`, `B`, or `C` is invalid.
aoclsparse_status_invalid_value	The value of `descr->base`, `A->base` is invalid.
aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_general or input matrix `A` is not in CSR format

3.4.2.73 aoclsparse_dcsr2m()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2m (
        aoclsparse_operation trans_A,
        const aoclsparse_mat_descr descrA,
        const aoclsparse_matrix csrA,
        aoclsparse_operation trans_B,
        const aoclsparse_mat_descr descrB,
        const aoclsparse_matrix csrB,
        const aoclsparse_request request,
        aoclsparse_matrix * csrC )
```

Sparse matrix Sparse matrix multiplication using CSR storage format for single and double precision datatypes.

`aoclsparse_csr2m` multiplies a sparse $m \times k$ matrix A, defined in CSR storage format, and the sparse $k \times n$ matrix B, defined in CSR storage format and stores the result to the sparse $m \times n$ matrix C, such that

$$C := op(A) \cdot op(B),$$

with

$$op(A) = \begin{cases} A, & \text{if trans\_A} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans\_A} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans\_A} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

and

$$op(B) = \begin{cases} B, & \text{if trans\_B} = \text{aoclsparse\_operation\_none} \\ B^T, & \text{if trans\_B} = \text{aoclsparse\_operation\_transpose} \\ B^H, & \text{if trans\_B} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Parameters

in	*trans_A*	matrix A operation type.
in	*descrA*	descriptor of the sparse CSR matrix A. Currently, only aoclsparse_matrix_type_general is supported.
in	*csrA*	sparse CSR matrix A structure.
in	*trans_B*	matrix B operation type.
in	*descrB*	descriptor of the sparse CSR matrix B. Currently, only aoclsparse_matrix_type_general is supported.
in	*csrB*	sparse CSR matrix B structure.
in	*request*	Specifies full computation or two-stage algorithm aoclsparse_stage_nnz_count , Only rowIndex array of the CSR matrix is computed internally. The output sparse CSR matrix can be extracted to measure the memory required for full operation. aoclsparse_stage_finalize . Finalize computation of remaining output arrays (column indices and values of output matrix entries) . Has to be called only after aoclsparse_dcsr2m call with aoclsparse_stage_nnz_count parameter. aoclsparse_stage_full_computation . Perform the entire computation in a single step.
out	**csrC*	Pointer to sparse CSR matrix C structure.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	input parameters contain an invalid value.
aoclsparse_status_invalid_pointer	`descrA`, `csr`, `descrB`, `csrB`, `csrC` is invalid.
aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_general or input matrices `A` or `B` is not in CSR format

Example

Shows multiplication of 2 sparse matrices to give a newly allocated sparse matrix

```
    aoclsparse_matrix  csrA;
    aoclsparse_create_dcsr(&csrA, base, M, K, nnz_A, csr_row_ptr_A.data(), csr_col_ind_A.data(),
 csr_val_A.data());
    aoclsparse_matrix  csrB;
    aoclsparse_create_dcsr(&csrB, base, K, N, nnz_B, csr_row_ptr_B.data(), csr_col_ind_B.data(),
 csr_val_B.data());
aoclsparse_matrix  csrC = NULL;
aoclsparse_int *csr_row_ptr_C = NULL;
aoclsparse_int *csr_col_ind_C = NULL;
double           *csr_val_C = NULL;
aoclsparse_int C_M, C_N;
request =  aoclsparse_stage_nnz_count;
CHECK_AOCLSPARSE_ERROR(aoclsparse_dcsr2m(transA,
    descrA,
    csrA,
    transB,
    descrB,
    csrB,
    request,
    &csrC));
request =  aoclsparse_stage_finalize;
CHECK_AOCLSPARSE_ERROR(aoclsparse_dcsr2m(transA,
    descrA,
    csrA,
    transB,
    descrB,
    csrB,
    request,
    &csrC));
aoclsparse_export_mat_csr(csrC, &base, &C_M, &C_N, &nnz_C, &csr_row_ptr_C, &csr_col_ind_C, (void
 **)&csr_val_C);
```

3.4.2.74 aoclsparse_scsr2m()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsr2m (
        aoclsparse_operation trans_A,
        const aoclsparse_mat_descr descrA,
        const aoclsparse_matrix csrA,
        aoclsparse_operation trans_B,
        const aoclsparse_mat_descr descrB,
        const aoclsparse_matrix csrB,
        const aoclsparse_request request,
        aoclsparse_matrix * csrC )
```

Sparse matrix Sparse matrix multiplication using CSR storage format for single and double precision datatypes.

`aoclsparse_csr2m` multiplies a sparse $m \times k$ matrix A, defined in CSR storage format, and the sparse $k \times n$ matrix B, defined in CSR storage format and stores the result to the sparse $m \times n$ matrix C, such that

$$C := op(A) \cdot op(B),$$

with

$$op(A) = \begin{cases} A, & \text{if trans\_A} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if trans\_A} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if trans\_A} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

and

$$op(B) = \begin{cases} B, & \text{if trans\_B} = \text{aoclsparse\_operation\_none} \\ B^T, & \text{if trans\_B} = \text{aoclsparse\_operation\_transpose} \\ B^H, & \text{if trans\_B} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

Parameters

in	*trans←_A*	matrix A operation type.

Parameters

in	*descrA*	descriptor of the sparse CSR matrix A. Currently, only aoclsparse_matrix_type_general is supported.
in	*csrA*	sparse CSR matrix A structure.
in	*trans←_B*	matrix B operation type.
in	*descrB*	descriptor of the sparse CSR matrix B. Currently, only aoclsparse_matrix_type_general is supported.
in	*csrB*	sparse CSR matrix B structure.
in	*request*	Specifies full computation or two-stage algorithm aoclsparse_stage_nnz_count , Only rowIndex array of the CSR matrix is computed internally. The output sparse CSR matrix can be extracted to measure the memory required for full operation. aoclsparse_stage_finalize . Finalize computation of remaining output arrays (column indices and values of output matrix entries) . Has to be called only after aoclsparse_dcsr2m call with aoclsparse_stage_nnz_count parameter. aoclsparse_stage_full_computation . Perform the entire computation in a single step.
out	**csrC*	Pointer to sparse CSR matrix C structure.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	input parameters contain an invalid value.
aoclsparse_status_invalid_pointer	`descrA`, `csr`, `descrB`, `csrB`, `csrC` is invalid.
aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_general or input matrices `A` or `B` is not in CSR format

Example

Shows multiplication of 2 sparse matrices to give a newly allocated sparse matrix

```
    aoclsparse_matrix  csrA;
    aoclsparse_create_dcsr(&csrA, base, M, K, nnz_A, csr_row_ptr_A.data(), csr_col_ind_A.data(),
 csr_val_A.data());
    aoclsparse_matrix  csrB;
    aoclsparse_create_dcsr(&csrB, base, K, N, nnz_B, csr_row_ptr_B.data(), csr_col_ind_B.data(),
 csr_val_B.data());
aoclsparse_matrix  csrC = NULL;
aoclsparse_int *csr_row_ptr_C = NULL;
aoclsparse_int *csr_col_ind_C = NULL;
double             *csr_val_C = NULL;
aoclsparse_int C_M, C_N;
request =  aoclsparse_stage_nnz_count;
CHECK_AOCLSPARSE_ERROR(aoclsparse_dcsr2m(transA,
    descrA,
    csrA,
    transB,
    descrB,
    csrB,
    request,
    &csrC));
request =  aoclsparse_stage_finalize;
CHECK_AOCLSPARSE_ERROR(aoclsparse_dcsr2m(transA,
    descrA,
    csrA,
    transB,
    descrB,
    csrB,
    request,
    &csrC));
aoclsparse_export_mat_csr(csrC, &base, &C_M, &C_N, &nnz_C, &csr_row_ptr_C, &csr_col_ind_C, (void
 **)&csr_val_C);
```

3.4.2.75 aoclsparse_dilu_smoother()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dilu_smoother (
        aoclsparse_operation op,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
        double ** precond_csr_val,
        const double * approx_inv_diag,
        double * x,
        const double * b )
```

Sparse Iterative solver algorithms for single and double precision datatypes.

`aoclsparse_ilu_smoother` performs Incomplete LU factorization on the sparse matrix `A`, defined in CSR storage format and also does an iterative LU solve to find an approximate `x`

Parameters

in	*op*	matrix `A` operation type. Transpose not yet supported.
in	*A*	sparse matrix handle. Currently ILU functionality is supported only for CSR matrix format.
in	*descr*	descriptor of the sparse matrix handle `A`. Currently, only aoclsparse_matrix_type_symmetric is supported. Both, base-zero and base-one input arrays of CSR matrix are supported
out	*precond_csr_val*	output pointer that contains L and U factors after ILU operation. The original value buffer of matrix `A` is not overwritten with the factors.
in	*approx_inv_diag*	It is unused as of now.
out	*x*	array of `n` element vector found using the known values of CSR matrix `A` and resultant vector product `b` in $Ax = b$. Every call to the API gives an iterative update of `x`, whcih is used to find norm during LU solve phase. Norm and Relative Error % decides the convergence of `x` with respect to `x_ref`
in	*b*	array of `m` elements which is the result of `A` and `x` in $Ax = b$. `b` is calculated using a known reference `x` vector, which is then used to find the norm for iterative `x` during LU solve phase. Norm and Relative Error percentage decides the convergence

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	input parameters contain an invalid value.
aoclsparse_status_invalid_pointer	`descr`, `A` is invalid.
aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_symmetric or input matrix `A` is not in CSR format

For a usage example, see the ILU example in $Ax = b$#95 folder.

3.4.2.76 aoclsparse_silu_smoother()

```
DLL_PUBLIC aoclsparse_status aoclsparse_silu_smoother (
        aoclsparse_operation op,
        aoclsparse_matrix A,
        const aoclsparse_mat_descr descr,
```

```
float ** precond_csr_val,
const float * approx_inv_diag,
float * x,
const float * b )
```

Sparse Iterative solver algorithms for single and double precision datatypes.

`aoclsparse_ilu_smoother` performs Incomplete LU factorization on the sparse matrix `A`, defined in CSR storage format and also does an iterative LU solve to find an approximate `x`

Parameters

in	*op*	matrix `A` operation type. Transpose not yet supported.
in	*A*	sparse matrix handle. Currently ILU functionality is supported only for CSR matrix format.
in	*descr*	descriptor of the sparse matrix handle `A`. Currently, only aoclsparse_matrix_type_symmetric is supported. Both, base-zero and base-one input arrays of CSR matrix are supported
out	*precond_csr_val*	output pointer that contains L and U factors after ILU operation. The original value buffer of matrix `A` is not overwritten with the factors.
in	*approx_inv_diag*	It is unused as of now.
out	*x*	array of `n` element vector found using the known values of CSR matrix `A` and resultant vector product `b` in $Ax = b$. Every call to the API gives an iterative update of `x`, whcih is used to find norm during LU solve phase. Norm and Relative Error % decides the convergence of `x` with respect to `x_ref`
in	*b*	array of `m` elements which is the result of `A` and `x` in $Ax = b$. `b` is calculated using a known reference `x` vector, which is then used to find the norm for iterative `x` during LU solve phase. Norm and Relative Error percentage decides the convergence

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_size	input parameters contain an invalid value.
aoclsparse_status_invalid_pointer	`descr`, `A` is invalid.
aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_symmetric or input matrix `A` is not in CSR format

For a usage example, see the ILU example in $Ax = b$#95 folder.

3.4.2.77 aoclsparse_sadd()

```
DLL_PUBLIC aoclsparse_status aoclsparse_sadd (
const aoclsparse_operation op,
const aoclsparse_matrix A,
const float alpha,
const aoclsparse_matrix B,
aoclsparse_matrix * C )
```

Addition of two sparse matrices.

`aoclsparse_(s/d/c/z)add` sums two sparse matrices and returns the result as a newly allocated sparse matrix for real and complex types, respectively. It performs the following operation:

$$C = \alpha * op(A) + B$$

with

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

where A is a $m \times n$ matrix and B is a $m \times n$ matrix if op = aoclsparse_operation_none and $n \times m$ otherwise and the result matrix C has the same dimension as B.

Note

Only matrices in CSR format are supported in this release.

Parameters

in	*op*	matrix A operation type.
in	*alpha*	scalar with same precision as A and B matrix
in	*A*	source sparse matrix A
in	*B*	source sparse matrix B
out	$*C$	pointer to the sparse output matrix C

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	`A` or `B` or `C` are invalid
aoclsparse_status_invalid_size	The dimensions of `A` and `B` are not compatible.
aoclsparse_status_memory_error	Memory allocation failure.
aoclsparse_status_not_implemented	Matrices are not in CSR format.

3.4.2.78 aoclsparse_dadd()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dadd (
        const aoclsparse_operation op,
        const aoclsparse_matrix A,
        const double alpha,
        const aoclsparse_matrix B,
        aoclsparse_matrix * C )
```

Addition of two sparse matrices.

`aoclsparse_(s/d/c/z)add` sums two sparse matrices and returns the result as a newly allocated sparse matrix for real and complex types, respectively. It performs the following operation:

$$C = \alpha * op(A) + B$$

with

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

where A is a $m \times n$ matrix and B is a $m \times n$ matrix if op = aoclsparse_operation_none and $n \times m$ otherwise and the result matrix C has the same dimension as B.

Note

Only matrices in CSR format are supported in this release.

Parameters

in	*op*	matrix A operation type.
in	*alpha*	scalar with same precision as A and B matrix
in	*A*	source sparse matrix A
in	*B*	source sparse matrix B
out	$*C$	pointer to the sparse output matrix C

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	`A` or `B` or `C` are invalid
aoclsparse_status_invalid_size	The dimensions of `A` and `B` are not compatible.
aoclsparse_status_memory_error	Memory allocation failure.
aoclsparse_status_not_implemented	Matrices are not in CSR format.

3.4.2.79 aoclsparse_cadd()

```
DLL_PUBLIC aoclsparse_status aoclsparse_cadd (
    const aoclsparse_operation op,
    const aoclsparse_matrix A,
    const aoclsparse_float_complex alpha,
    const aoclsparse_matrix B,
    aoclsparse_matrix * C )
```

Addition of two sparse matrices.

`aoclsparse_(s/d/c/z)add` sums two sparse matrices and returns the result as a newly allocated sparse matrix for real and complex types, respectively. It performs the following operation:

$$C = \alpha * op(A) + B$$

with

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

where A is a $m \times n$ matrix and B is a $m \times n$ matrix if op = aoclsparse_operation_none and $n \times m$ otherwise and the result matrix C has the same dimension as B.

Note

Only matrices in CSR format are supported in this release.

Parameters

in	*op*	matrix A operation type.
in	*alpha*	scalar with same precision as A and B matrix
in	*A*	source sparse matrix A
in	*B*	source sparse matrix B
out	$*C$	pointer to the sparse output matrix C

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	`A` or `B` or `C` are invalid
aoclsparse_status_invalid_size	The dimensions of `A` and `B` are not compatible.
aoclsparse_status_memory_error	Memory allocation failure.
aoclsparse_status_not_implemented	Matrices are not in CSR format.

3.4.2.80 aoclsparse_zadd()

```
DLL_PUBLIC aoclsparse_status aoclsparse_zadd (
        const aoclsparse_operation op,
        const aoclsparse_matrix A,
        const aoclsparse_double_complex alpha,
        const aoclsparse_matrix B,
        aoclsparse_matrix * C )
```

Addition of two sparse matrices.

`aoclsparse_(s/d/c/z)add` sums two sparse matrices and returns the result as a newly allocated sparse matrix for real and complex types, respectively. It performs the following operation:

$$C = \alpha * op(A) + B$$

with

$$op(A) = \begin{cases} A, & \text{if op} = \text{aoclsparse\_operation\_none} \\ A^T, & \text{if op} = \text{aoclsparse\_operation\_transpose} \\ A^H, & \text{if op} = \text{aoclsparse\_operation\_conjugate\_transpose} \end{cases}$$

where A is a $m \times n$ matrix and B is a $m \times n$ matrix if op = aoclsparse_operation_none and $n \times m$ otherwise and the result matrix C has the same dimension as B.

Note

Only matrices in CSR format are supported in this release.

Parameters

`in`	*op*	matrix A operation type.
`in`	*alpha*	scalar with same precision as A and B matrix
`in`	*A*	source sparse matrix A
`in`	*B*	source sparse matrix B
`out`	$*C$	pointer to the sparse output matrix C

Return values

aoclsparse_status_success	The operation completed successfully.
aoclsparse_status_invalid_pointer	`A` or `B` or `C` are invalid
aoclsparse_status_invalid_size	The dimensions of `A` and `B` are not compatible.
aoclsparse_status_memory_error	Memory allocation failure.
aoclsparse_status_not_implemented	Matrices are not in CSR format.

3.5 aoclsparse_solvers.h File Reference

aoclsparse_solvers.h provides iterative sparse linear system solvers.

Typedefs

- typedef enum aoclsparse_itsol_rci_job_ aoclsparse_itsol_rci_job

    Values of `ircomm` *used by the iterative solver reverse communication interface (RCI) aoclsparse_itsol_d_rci_solve and aoclsparse_itsol_s_rci_solve to communicate back to the user which operation is required.*

Enumerations

- enum aoclsparse_itsol_rci_job_ {
aoclsparse_rci_interrupt = -1 , aoclsparse_rci_stop = 0 , aoclsparse_rci_start , aoclsparse_rci_mv ,
aoclsparse_rci_precond , aoclsparse_rci_stopping_criterion }

    Values of `ircomm` *used by the iterative solver reverse communication interface (RCI) aoclsparse_itsol_d_rci_solve and aoclsparse_itsol_s_rci_solve to communicate back to the user which operation is required.*

Functions

- DLL_PUBLIC void aoclsparse_itsol_handle_prn_options (aoclsparse_itsol_handle handle)

    Print options stored in a problem handle.
- DLL_PUBLIC aoclsparse_status aoclsparse_itsol_option_set (aoclsparse_itsol_handle handle, const char *option, const char *value)

    Option Setter.
- DLL_PUBLIC void aoclsparse_itsol_destroy (aoclsparse_itsol_handle *handle)

    Free the memory reserved in a problem `handle` *previously initialized by aoclsparse_itsol_s_init or aoclsparse_itsol_d_init.*

- DLL_PUBLIC aoclsparse_status aoclsparse_itsol_d_init (aoclsparse_itsol_handle *handle)

    Initialize a problem `handle` *(* `aoclsparse_itsol_handle`*) for the iterative solvers suite of the library.*
- DLL_PUBLIC aoclsparse_status aoclsparse_itsol_s_init (aoclsparse_itsol_handle *handle)

    Initialize a problem `handle` *(* `aoclsparse_itsol_handle`*) for the iterative solvers suite of the library.*

- DLL_PUBLIC aoclsparse_status aoclsparse_itsol_d_rci_input (aoclsparse_itsol_handle handle, aoclsparse_int n, const double *b)

    Store partial data of the linear system of equations into the problem `handle`*.*
- DLL_PUBLIC aoclsparse_status aoclsparse_itsol_s_rci_input (aoclsparse_itsol_handle handle, aoclsparse_int n, const float *b)

    Store partial data of the linear system of equations into the problem `handle`*.*

- DLL_PUBLIC aoclsparse_status aoclsparse_itsol_d_rci_solve (aoclsparse_itsol_handle handle, aoclsparse_itsol_rci_job *ircomm, double **u, double **v, double *x, double rinfo[100])

Reverse Communication Interface (RCI) to the iterative solvers (itsol) suite.

- DLL_PUBLIC aoclsparse_status aoclsparse_itsol_s_rci_solve (aoclsparse_itsol_handle handle, aoclsparse_itsol_rci_job ∗ircomm, float ∗∗u, float ∗∗v, float ∗x, float rinfo[100])

  Reverse Communication Interface (RCI) to the iterative solvers (itsol) suite.

- DLL_PUBLIC aoclsparse_status aoclsparse_itsol_d_solve (aoclsparse_itsol_handle handle, aoclsparse_int n, aoclsparse_matrix mat, const aoclsparse_mat_descr descr, const double ∗b, double ∗x, double rinfo[100], aoclsparse_int precond(aoclsparse_int flag, aoclsparse_int n, const double ∗u, double ∗v, void ∗udata), aoclsparse_int monit(aoclsparse_int n, const double ∗x, const double ∗r, double rinfo[100], void ∗udata), void ∗udata)

  Forward communication interface to the iterative solvers suite of the library.

- DLL_PUBLIC aoclsparse_status aoclsparse_itsol_s_solve (aoclsparse_itsol_handle handle, aoclsparse_int n, aoclsparse_matrix mat, const aoclsparse_mat_descr descr, const float ∗b, float ∗x, float rinfo[100], aoclsparse_int precond(aoclsparse_int flag, aoclsparse_int n, const float ∗u, float ∗v, void ∗udata), aoclsparse_int monit(aoclsparse_int n, const float ∗x, const float ∗r, float rinfo[100], void ∗udata), void ∗udata)

  Forward communication interface to the iterative solvers suite of the library.

3.5.1 Detailed Description

aoclsparse_solvers.h provides iterative sparse linear system solvers.

3.5.2 Iterative Solver Suite (itsol)

3.5.2.1 Introduction AOCL Sparse Iterative Solver Suite (itsol) is an iterative framework for solving large-scale sparse linear systems of equations of the form

$$Ax = b,$$

where A is a sparse full-rank square matrix of size n by n, b is a dense n-vector, and x is the vector of unknowns also of size n. The framework solves the previous problem using either the Conjugate Gradient method or GMRES. It supports a variety of preconditioners (*accelerators*) such as Symmetric Gauss-Seidel or Incomplete LU factorization, ILU(0).

Iterative solvers at each step (iteration) find a better approximation to the solution of the linear system of equations in the sense that it reduces an error metric. In contrast, direct solvers only provide a solution once the full algorithm as been executed. A great advantage of iterative solvers is that they can be interrupted once an approximate solution is deemed acceptable.

3.5.2.2 Forward and Reverse Communication Interfaces The suite presents two separate interfaces to all the iterative solvers, a direct one, aoclsparse_itsol_d_rci_solve (aoclsparse_itsol_s_rci_solve), and a reverse communication (RCI) one aoclsparse_itsol_d_rci_solve (aoclsparse_itsol_s_rci_solve). While the underlying algorithms are exactly the same, the difference lies in how data is communicated to the solvers.

The direct communication interface expects to have explicit access to the coefficient matrix A. On the other hand, the reverse communication interface makes no assumption on the matrix storage. Thus when the solver requires some matrix operation such as a matrix-vector product, it returns control to the user and asks the user perform the operation and provide the results by calling again the RCI solver.

3.5.2.3 Recommended Workflow

For solving a linear system of equations, the following workflow is recommended:

- Call aoclsparse_itsol_s_init or aoclsparse_itsol_d_init to initialize aoclsparse_itsol_handle.
- Choose the solver and adjust its behaviour by setting optional parameters with aoclsparse_itsol_option_set, see also Options.
- If the reverse communication interface is desired, define the system's input with aoclsparse_itsol_d_rci_input.
- Solve the system with either using direct interface aoclsparse_itsol_s_solve (or aoclsparse_itsol_d_solve) or reverse communication interface aoclsparse_itsol_s_rci_solve (or aoclsparse_itsol_d_rci_solve)
- Free the memory with aoclsparse_itsol_destroy.

3.5.2.4 Information Array

The array `rinfo`[100] is used by the solvers (e.g. aoclsparse_itsol_d_solve or aoclsparse_itsol_s_rci_solve) to report back useful convergence metrics and other solver statistics. The user callback `monit` is also equipped with this array and can be used to view or monitor the state of the solver. The solver will populate the following entries with the most recent iteration data

Index	Description
0	Absolute residual norm, $r_{\text{abs}} = \|Ax - b\|_2$.
1	Norm of the right-hand side vector b, $\|b\|_2$.
2-29	Reserved for future use.
30	Iteration counter.
31-99	Reserved for future use.

3.5.2.5 Examples

Each iterative solver in the itsol suite is provided with an illustrative example on its usage. The source file for the examples can be found under the `tests/examples/` folder.

Solver	Precision	Filename	Description
itsol forward communication interface	double	`sample_itsol_d_cg.cpp`	Solves a linear system of equations using the Conjugate Gradient method.
	single	`sample_itsol_s_cg.cpp`	
itsol reverse communication interface	double	`sample_itsol_d_cg_rci.cpp`	Solves a linear system of equations using the Conjugate Gradient method.
	single	`sample_itsol_s_cg_rci.cpp`	

3.5.2.6 References

1. Yousef Saad, *Iterative Methods for Sparse Linear Systems*. 2nd ed. 2003. pp xxi + 547.
2. Conjugate gradients, method of. Encyclopedia of Mathematics. URL: `Conjugate Gradients method`.
3. Acceleration methods. Encyclopedia of Mathematics. URL: `Acceleration methods`.

3.5.3 Enumeration Type Documentation

3.5.3.1 aoclsparse_itsol_rci_job_ `enum aoclsparse_itsol_rci_job_`

Values of `ircomm` used by the iterative solver reverse communication interface (RCI) aoclsparse_itsol_d_rci_solve and aoclsparse_itsol_s_rci_solve to communicate back to the user which operation is required.

Enumerator

aoclsparse_rci_interrupt	if set by the user, signals the solver to terminate. This is never set by the solver. Terminate.
aoclsparse_rci_stop	found a solution within specified tolerance (see options "cg rel tolerance", "cg abs tolerance", "gmres rel tolerance", and "gmres abs tolerance" in Options). Terminate, vector `x` contains the solution.
aoclsparse_rci_start	initial value of the `ircomm` flag, no action required. Call solver.
aoclsparse_rci_mv	perform the matrix-vector product $v = Au$. Return control to solver.
aoclsparse_rci_precond	perform a preconditioning step on the vector u and store in v. If the preconditioner M has explicit matrix form, then applying the preconditioner would result in the operations $v = Mu$ or $v = M^{-1}u$. The latter would be performed by solving the linear system of equations $Mv = u$. Return control to solver.
aoclsparse_rci_stopping_criterion	perform a monitoring step and check for custom stopping criteria. If using a positive tolerance value for the convergence options (see aoclsparse_rci_stop), then this step can be ignored and control can be returned to solver.

3.5.4 Function Documentation

3.5.4.1 aoclsparse_itsol_handle_prn_options()

```
DLL_PUBLIC void aoclsparse_itsol_handle_prn_options (
            aoclsparse_itsol_handle handle )
```

Print options stored in a problem handle.

This function prints to the standard output a list of available options stored in a problem handle and their current value. For available options, see Options in aoclsparse_itsol_option_set.

Parameters

in	*handle*	pointer to the iterative solvers' data structure.

3.5.4.2 aoclsparse_itsol_option_set()

```
DLL_PUBLIC aoclsparse_status aoclsparse_itsol_option_set (
            aoclsparse_itsol_handle handle,
            const char * option,
            const char * value )
```

Option Setter.

This function sets the value to a given option inside the provided problem handle. Handle options can be printed using aoclsparse_itsol_handle_prn_options. Available options are listed in Options.

Parameters

`in,out`	*handle*	pointer to the iterative solvers' data structure.
`in`	*option*	string specifying the name of the option to set.
`in`	*value*	string providing the value to set the option to.

3.5.5 Options

The iterative solver framework has the following options.

Option name	Type	Default value
cg iteration limit	integer	$i = 500$
Set CG iteration limit		
Valid values: $1 \leq i$.		
gmres iteration limit	integer	$i = 150$
Set GMRES iteration limit		
Valid values: $1 \leq i$.		
gmres restart iterations	integer	$i = 20$
Set GMRES restart iterations		
Valid values: $1 \leq i$.		
cg rel tolerance	real	$r = 1.08735e-06$
Set relative convergence tolerance for cg method		
Valid values: $0 \leq r$.		
cg abs tolerance	real	$r = 0$
Set absolute convergence tolerance for cg method		
Valid values: $0 \leq r$.		
gmres rel tolerance	real	$r = 1.08735e-06$
Set relative convergence tolerance for gmres method		
Valid values: $0 \leq r$.		
gmres abs tolerance	real	$r = 1e-06$
Set absolute convergence tolerance for gmres method		
Valid values: $0 \leq r$.		
iterative method	string	$s =$ `cg`
Choose solver to use		
Valid values: $s =$ `cg`, `gm res`, `gmres`, or `pcg`.		
cg preconditioner	string	$s =$ `none`
Choose preconditioner to use with cg method		
Valid values: $s =$ `gs`, `none`, `sgs`, `symgs`, or `user`.		
gmres preconditioner	string	$s =$ `none`
Choose preconditioner to use with gmres method		
Valid values: $s =$ `ilu0`, `none`, or `user`.		

Note

It is worth noting that only some options apply to each specific solver, e.g. name of options that begin with "cg" affect the behaviour of the CG solver.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_invalid_value	either the option name was not found or the provided option value is out of the valid range.
aoclsparse_status_invalid_pointer	the pointer to the problem handle is invalid.
aoclsparse_status_internal_error	an unexpected error occurred.

3.5.5.1 aoclsparse_itsol_d_init() `DLL_PUBLIC aoclsparse_status aoclsparse_itsol_d_init ( aoclsparse_itsol_handle * handle )`

Initialize a problem `handle` (`aoclsparse_itsol_handle`) for the iterative solvers suite of the library.

aoclsparse_itsol_s_init and aoclsparse_itsol_d_init initialize a data structure referred to as problem `handle`. This `handle` is used by iterative solvers (itsol) suite to setup options, define which solver to use, etc.

Parameters

`in,out`	*handle*	the pointer to the problem handle data structure.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_memory_error	internal memory allocation error.
aoclsparse_status_invalid_pointer	the pointer to the problem handle is invalid.
aoclsparse_status_internal_error	an unexpected error occurred.

Note

Once the `handle` is no longer needed, it can be destroyed and the memory released by calling aoclsparse_itsol_destroy.

3.5.5.2 aoclsparse_itsol_s_init() `DLL_PUBLIC aoclsparse_status aoclsparse_itsol_s_init ( aoclsparse_itsol_handle * handle )`

Initialize a problem `handle` (`aoclsparse_itsol_handle`) for the iterative solvers suite of the library.

aoclsparse_itsol_s_init and aoclsparse_itsol_d_init initialize a data structure referred to as problem `handle`. This `handle` is used by iterative solvers (itsol) suite to setup options, define which solver to use, etc.

Parameters

in,out	*handle*	the pointer to the problem handle data structure.

Return values

aoclsparse_status_success	the operation completed successfully.
aoclsparse_status_memory_error	internal memory allocation error.
aoclsparse_status_invalid_pointer	the pointer to the problem handle is invalid.
aoclsparse_status_internal_error	an unexpected error occurred.

Note

Once the `handle` is no longer needed, it can be destroyed and the memory released by calling aoclsparse_itsol_destroy.

3.5.5.3 aoclsparse_itsol_destroy()

```
DLL_PUBLIC void aoclsparse_itsol_destroy (
        aoclsparse_itsol_handle * handle )
```

Free the memory reserved in a problem `handle` previously initialized by aoclsparse_itsol_s_init or aoclsparse_itsol_d_init.

Once the problem handle is no longer needed, calling this function to deallocate the memory is advisable to avoid memory leaks.

Note

Passing a `handle` that has not been initialized by aoclsparse_itsol_s_init or aoclsparse_itsol_d_init may have unpredictable results.

Parameters

in,out	*handle*	pointer to a problem handle.

3.5.5.4 aoclsparse_itsol_d_rci_input()

```
DLL_PUBLIC aoclsparse_status aoclsparse_itsol_d_rci_input
(
        aoclsparse_itsol_handle handle,
        aoclsparse_int n,
        const double * b )
```

Store partial data of the linear system of equations into the problem `handle`.

This function needs to be called before the reverse communication interface iterative solver is called. It registers the linear system's dimension `n`, and stores the right-hand side vector `b`.

Note

This function does not need to be called if the forward communication interface is used.

Parameters

`in,out`	*handle*	problem `handle`. Needs to be initialized by calling aoclsparse_itsol_s_init or aoclsparse_itsol_d_init.
`in`	*n*	the number of columns of the (square) linear system matrix.
`in`	*b*	the right hand side of the linear system. Must be a vector of size `n`.

Return values

aoclsparse_status_success	initialization completed successfully.
aoclsparse_status_invalid_pointer	one or more of the pointers `handle`, and `b` are invalid.
aoclsparse_status_wrong_type	`handle` was initialized with a different floating point precision than requested here, e.g. aoclsparse_itsol_d_init (double precision) was used to initialize `handle` but aoclsparse_itsol_s_rci_input (single precision) is being called instead of the correct double precision one, aoclsparse_itsol_d_rci_input.
aoclsparse_status_invalid_value	`n` was set to a negative value.
aoclsparse_status_memory_error	internal memory allocation error.

3.5.5.5 aoclsparse_itsol_s_rci_input()

```
DLL_PUBLIC aoclsparse_status aoclsparse_itsol_s_rci_input (
            aoclsparse_itsol_handle handle,
            aoclsparse_int n,
            const float * b )
```

Store partial data of the linear system of equations into the problem `handle`.

This function needs to be called before the reverse communication interface iterative solver is called. It registers the linear system's dimension `n`, and stores the right-hand side vector `b`.

Note

This function does not need to be called if the forward communication interface is used.

Parameters

`in,out`	*handle*	problem `handle`. Needs to be initialized by calling aoclsparse_itsol_s_init or aoclsparse_itsol_d_init.
`in`	*n*	the number of columns of the (square) linear system matrix.
`in`	*b*	the right hand side of the linear system. Must be a vector of size `n`.

Return values

aoclsparse_status_success	initialization completed successfully.
aoclsparse_status_invalid_pointer	one or more of the pointers `handle`, and `b` are invalid.

Return values

aoclsparse_status_wrong_type	`handle` was initialized with a different floating point precision than requested here, e.g. aoclsparse_itsol_d_init (double precision) was used to initialize `handle` but aoclsparse_itsol_s_rci_input (single precision) is being called instead of the correct double precision one, aoclsparse_itsol_d_rci_input.
aoclsparse_status_invalid_value	`n` was set to a negative value.
aoclsparse_status_memory_error	internal memory allocation error.

3.5.5.6 aoclsparse_itsol_d_rci_solve()

```
DLL_PUBLIC aoclsparse_status aoclsparse_itsol_d_rci_solve (
        aoclsparse_itsol_handle handle,
        aoclsparse_itsol_rci_job * ircomm,
        double ** u,
        double ** v,
        double * x,
        double rinfo[100] )
```

Reverse Communication Interface (RCI) to the iterative solvers (itsol) suite.

This function solves the linear system of equations

$$Ax = b,$$

where the matrix of coefficients A is not required to be provided explicitly. The right hand-side is the dense vector `b` and the vector of unknowns is `x`. If A is symmetric and positive definite then set the option "iterative method" to "cg" to solve the problem using the `Conjugate Gradient method`, alternatively set the option to "gmres" to solve using `GMRes`. See the Options for a list of available options to modify the behaviour of each solver.

The reverse communication interface (RCI), also know as *matrix-free* interface does not require the user to explicitly provide the matrix A. During the solve process whenever the algorithm requires a matrix operation (matrix-vector or transposed matrix-vector products), it returns control to the user with a flag `ircomm` indicating what operation is requested. Once the user performs the requested task it must call this function again to resume the solve.

The expected workflow is as follows:

1. Call aoclsparse_itsol_s_init or aoclsparse_itsol_d_init to initialize the problem `handle` (aoclsparse_itsol↩_handle)
2. Choose the solver and adjust its behaviour by setting optional parameters with aoclsparse_itsol_option_set, see also Options.
3. Define the problem size and right-hand side vector b with aoclsparse_itsol_d_rci_input.
4. Solve the system with either aoclsparse_itsol_s_rci_solve or aoclsparse_itsol_d_rci_solve.
5. If there is another linear system of equations to solve with the same matrix but a different right-hand side b, then repeat from step 3.
6. If solver terminated successfully then vector `x` contains the solution.
7. Free the memory with aoclsparse_itsol_destroy.

These reverse communication interfaces complement the *forward communication* interfaces aoclsparse_itsol_d_rci_solve and aoclsparse_itsol_s_rci_solve.

Parameters

in,out	*handle*	problem `handle`. Needs to be previously initialized by aoclsparse_itsol_s_init or aoclsparse_itsol_d_init and then populated using either aoclsparse_itsol_s_rci_input or aoclsparse_itsol_d_rci_input, as appropriate.
in,out	*ircomm*	pointer to the reverse communication instruction flag and defined in aoclsparse_itsol_rci_job_.
in,out	*u*	pointer to a generic vector of data. The solver will point to the data on which the operation defined by `ircomm` needs to be applied.
in,out	*v*	pointer to a generic vector of data. The solver will ask that the result of the operation defined by `ircomm` be stored in `v`.
in,out	*x*	dense vector of unknowns. On input, it should contain the initial guess from which to start the iterative process. If there is no good initial estimate guess then any arbitrary but finite values can be used. On output, it contains an estimate to the solution of the linear system of equations up to the requested tolerance, e.g. see "cg rel tolerance" or "cg abs tolerance" in Options.
out	*rinfo*	vector containing information and stats related to the iterative solve, see Information Array. This parameter can be used to monitor progress and define a custom stopping criterion when the solver returns control to user with `ircomm` = aoclsparse_rci_stopping_criterion.

Note

This function returns control back to the user under certain circumstances. The table in aoclsparse_itsol_rci_job_ indicates what actions are required to be performed by the user.

For an illustrative example see Examples.

3.5.5.7 aoclsparse_itsol_s_rci_solve()

```
DLL_PUBLIC aoclsparse_status aoclsparse_itsol_s_rci_solve
(
            aoclsparse_itsol_handle handle,
            aoclsparse_itsol_rci_job * ircomm,
            float ** u,
            float ** v,
            float * x,
            float rinfo[100] )
```

Reverse Communication Interface (RCI) to the iterative solvers (itsol) suite.

This function solves the linear system of equations

$$Ax = b,$$

where the matrix of coefficients A is not required to be provided explicitly. The right hand-side is the dense vector `b` and the vector of unknowns is `x`. If A is symmetric and positive definite then set the option "iterative method" to "cg" to solve the problem using the `Conjugate Gradient method`, alternatively set the option to "gmres" to solve using `GMRes`. See the Options for a list of available options to modify the behaviour of each solver.

The reverse communication interface (RCI), also know as *matrix-free* interface does not require the user to explicitly provide the matrix A. During the solve process whenever the algorithm requires a matrix operation (matrix-vector or transposed matrix-vector products), it returns control to the user with a flag `ircomm` indicating what operation is requested. Once the user performs the requested task it must call this function again to resume the solve.

The expected workflow is as follows:

1. Call aoclsparse_itsol_s_init or aoclsparse_itsol_d_init to initialize the problem `handle` (aoclsparse_itsol↩_handle)
2. Choose the solver and adjust its behaviour by setting optional parameters with aoclsparse_itsol_option_set, see also Options.
3. Define the problem size and right-hand side vector b with aoclsparse_itsol_d_rci_input.
4. Solve the system with either aoclsparse_itsol_s_rci_solve or aoclsparse_itsol_d_rci_solve.
5. If there is another linear system of equations to solve with the same matrix but a different right-hand side b, then repeat from step 3.
6. If solver terminated successfully then vector `x` contains the solution.
7. Free the memory with aoclsparse_itsol_destroy.

These reverse communication interfaces complement the *forward communication* interfaces aoclsparse_itsol_d_rci_solve and aoclsparse_itsol_s_rci_solve.

Parameters

`in,out`	*handle*	problem `handle`. Needs to be previously initialized by aoclsparse_itsol_s_init or aoclsparse_itsol_d_init and then populated using either aoclsparse_itsol_s_rci_input or aoclsparse_itsol_d_rci_input, as appropriate.
`in,out`	*ircomm*	pointer to the reverse communication instruction flag and defined in aoclsparse_itsol_rci_job_.
`in,out`	*u*	pointer to a generic vector of data. The solver will point to the data on which the operation defined by `ircomm` needs to be applied.
`in,out`	*v*	pointer to a generic vector of data. The solver will ask that the result of the operation defined by `ircomm` be stored in `v`.
`in,out`	*x*	dense vector of unknowns. On input, it should contain the initial guess from which to start the iterative process. If there is no good initial estimate guess then any arbitrary but finite values can be used. On output, it contains an estimate to the solution of the linear system of equations up to the requested tolerance, e.g. see "cg rel tolerance" or "cg abs tolerance" in Options.
`out`	*rinfo*	vector containing information and stats related to the iterative solve, see Information Array. This parameter can be used to monitor progress and define a custom stopping criterion when the solver returns control to user with `ircomm` = aoclsparse_rci_stopping_criterion.

Note

This function returns control back to the user under certain circumstances. The table in aoclsparse_itsol_rci_job_ indicates what actions are required to be performed by the user.

For an illustrative example see Examples.

3.5.5.8 aoclsparse_itsol_d_solve()

```
DLL_PUBLIC aoclsparse_status aoclsparse_itsol_d_solve (
        aoclsparse_itsol_handle handle,
        aoclsparse_int n,
        aoclsparse_matrix mat,
        const aoclsparse_mat_descr descr,
        const double * b,
```

```
            double * x,
            double rinfo[100],
            aoclsparse_int  precondaoclsparse_int flag, aoclsparse_int n, const double *u,
double *v, void *udata,
            aoclsparse_int  monitaoclsparse_int n, const double *x, const double *r, double
rinfo[100], void *udata,
            void * udata )
```

Forward communication interface to the iterative solvers suite of the library.

This function solves the linear system of equations

$$Ax = b,$$

where the matrix of coefficients A is defined by `mat`. The right hand-side is the dense vector `b` and the vector of unknowns is `x`. If A is symmetric and positive definite then set the option "iterative method" to "cg" to solve the problem using the `Conjugate Gradient method`, alternatively set the option to "gmres" to solve using `GMRes`. See the Options for a list of available options to modify the behaviour of each solver.

The expected workflow is as follows:

1. Call aoclsparse_itsol_s_init or aoclsparse_itsol_d_init to initialize the problem `handle` (aoclsparse_itsol_handle).
2. Choose the solver and adjust its behaviour by setting optional parameters with aoclsparse_itsol_option_set, see also Options.
3. Solve the system by calling aoclsparse_itsol_s_solve or aoclsparse_itsol_d_solve.
4. If there is another linear system of equations to solve with the same matrix but a different right-hand side b, then repeat from step 3.
5. If solver terminated successfully then vector `x` contains the solution.
6. Free the memory with aoclsparse_itsol_destroy.

This interface requires to explicitly provide the matrix A and its descriptor `descr`, this kind of interface is also known as *forward communication* which contrasts with *reverse communication* in which case the matrix A and its descriptor `descr` need not be explicitly available. For more details on the latter, see aoclsparse_itsol_d_rci_solve or aoclsparse_itsol_s_rci_solve.

Parameters

in,out	*handle*	a valid problem handle, previously initialized by calling aoclsparse_itsol_s_init or aoclsparse_itsol_d_init.
in	*n*	the size of the square matrix `mat`.
in,out	*mat*	coefficient matrix A.
in,out	*descr*	matrix descriptor for `mat`.
in	*b*	right-hand side dense vector b.
in,out	*x*	dense vector of unknowns. On input, it should contain the initial guess from which to start the iterative process. If there is no good initial estimate guess then any arbitrary but finite values can be used. On output, it contains an estimate to the solution of the linear system of equations up to the requested tolerance, e.g. see "cg rel tolerance" or "cg abs tolerance" in Options.
out	*rinfo*	vector containing information and stats related to the iterative solve, see Information Array.

Parameters

in	*precond*	(optional, can be nullptr) function pointer to a user routine that applies the preconditioning step $$v = Mu\text{or } v = M^{-1}u,$$ where v is the resulting vector of applying a preconditioning step on the vector u and M refers to the user specified preconditioner in matrix form and need not be explicitly available. The void pointer udata, is a convenience pointer that can be used by the user to point to user data and is not used by the itsol framework. If the user requests to use a predefined preconditioner already available in the suite (refer to e.g. "cg preconditioner" or "gmres preconditioner" in Options), then this parameter need not be provided.
in	*monit*	(optional, can be nullptr) function pointer to a user monitoring routine. If provided, then at each iteration, the routine is called and can be used to define a custom stopping criteria or to oversee the convergence process. In general, this function need not be provided. If provided then the solver provides `n` the problem size, `x` the current iterate, `r` the current residual vector ($r = Ax - b$), `rinfo` the current solver's stats, see Information Array, and `udata` a convenience pointer that can be used by the user to point to arbitrary user data and is not used by the itsol framework.
in,out	*udata*	(optional, can be nullptr) user convenience pointer, it can be used by the user to pass a pointer to user data. It is not modified by the solver.

Note

For an illustrative example see Examples.

3.5.5.9 aoclsparse_itsol_s_solve()

```
DLL_PUBLIC aoclsparse_status aoclsparse_itsol_s_solve (
            aoclsparse_itsol_handle handle,
            aoclsparse_int n,
            aoclsparse_matrix mat,
            const aoclsparse_mat_descr descr,
            const float * b,
            float * x,
            float rinfo[100],
            aoclsparse_int  precondaoclsparse_int flag, aoclsparse_int n, const float *u,
float *v, void *udata,
            aoclsparse_int  monitaoclsparse_int n, const float *x, const float *r, float rinfo[100],
void *udata,
            void * udata )
```

Forward communication interface to the iterative solvers suite of the library.

This function solves the linear system of equations

$$Ax = b,$$

where the matrix of coefficients A is defined by `mat`. The right hand-side is the dense vector `b` and the vector of unknowns is `x`. If A is symmetric and positive definite then set the option "iterative method" to "cg" to solve the problem using the `Conjugate Gradient method`, alternatively set the option to "gmres" to solve using `GMRes`. See the Options for a list of available options to modify the behaviour of each solver.

The expected workflow is as follows:

1. Call aoclsparse_itsol_s_init or aoclsparse_itsol_d_init to initialize the problem `handle` (aoclsparse_itsol←_handle).
2. Choose the solver and adjust its behaviour by setting optional parameters with aoclsparse_itsol_option_set, see also Options.
3. Solve the system by calling aoclsparse_itsol_s_solve or aoclsparse_itsol_d_solve.
4. If there is another linear system of equations to solve with the same matrix but a different right-hand side b, then repeat from step 3.
5. If solver terminated successfully then vector `x` contains the solution.
6. Free the memory with aoclsparse_itsol_destroy.

This interface requires to explicitly provide the matrix A and its descriptor `descr`, this kind of interface is also known as *forward communication* which contrasts with *reverse communication* in which case the matrix A and its descriptor `descr` need not be explicitly available. For more details on the latter, see aoclsparse_itsol_d_rci_solve or aoclsparse_itsol_s_rci_solve.

Parameters

`in,out`	*handle*	a valid problem handle, previously initialized by calling aoclsparse_itsol_s_init or aoclsparse_itsol_d_init.
`in`	*n*	the size of the square matrix `mat`.
`in,out`	*mat*	coefficient matrix A.
`in,out`	*descr*	matrix descriptor for `mat`.
`in`	*b*	right-hand side dense vector b.
`in,out`	*x*	dense vector of unknowns. On input, it should contain the initial guess from which to start the iterative process. If there is no good initial estimate guess then any arbitrary but finite values can be used. On output, it contains an estimate to the solution of the linear system of equations up to the requested tolerance, e.g. see "cg rel tolerance" or "cg abs tolerance" in Options.
`out`	*rinfo*	vector containing information and stats related to the iterative solve, see Information Array.
`in`	*precond*	(optional, can be nullptr) function pointer to a user routine that applies the preconditioning step $$v = Mu \text{or } v = M^{-1}u,$$ where v is the resulting vector of applying a preconditioning step on the vector u and M refers to the user specified preconditioner in matrix form and need not be explicitly available. The void pointer udata, is a convenience pointer that can be used by the user to point to user data and is not used by the itsol framework. If the user requests to use a predefined preconditioner already available in the suite (refer to e.g. "cg preconditioner" or "gmres preconditioner" in Options), then this parameter need not be provided.
`in`	*monit*	(optional, can be nullptr) function pointer to a user monitoring routine. If provided, then at each iteration, the routine is called and can be used to define a custom stopping criteria or to oversee the convergence process. In general, this function need not be provided. If provided then the solver provides `n` the problem size, `x` the current iterate, `r` the current residual vector ($r = Ax - b$), `rinfo` the current solver's stats, see Information Array, and `udata` a convenience pointer that can be used by the user to point to arbitrary user data and is not used by the itsol framework.
`in,out`	*udata*	(optional, can be nullptr) user convenience pointer, it can be used by the user to pass a pointer to user data. It is not modified by the solver.

Note

For an illustrative example see Examples.

3.6 aoclsparse_types.h File Reference

aoclsparse_types.h defines data types used by aoclsparse

Macros

- #define DLL_PUBLIC __attribute__((__visibility__("default")))
  Macro for function attribute.

Typedefs

- typedef int32_t aoclsparse_int
  Specifies whether int32 or int64 is used.
- typedef struct _aoclsparse_mat_descr ∗ aoclsparse_mat_descr
  Descriptor of the matrix.
- typedef struct _aoclsparse_csr ∗ aoclsparse_csr
  CSR matrix storage format.
- typedef struct _aoclsparse_matrix ∗ aoclsparse_matrix
  AOCL sparse matrix.
- typedef enum aoclsparse_operation_ aoclsparse_operation
  Specify whether the matrix is to be transposed or not.
- typedef enum aoclsparse_index_base_ aoclsparse_index_base
  Specify the matrix index base.
- typedef enum aoclsparse_matrix_type_ aoclsparse_matrix_type
  Specify the matrix type.
- typedef enum aoclsparse_matrix_data_type_ aoclsparse_matrix_data_type
  Specify the matrix data type.
- typedef enum aoclsparse_ilu_type_ aoclsparse_ilu_type
  Specify the type of ILU factorization.
- typedef enum aoclsparse_matrix_format_type_ aoclsparse_matrix_format_type
  Specify the matrix storage format type.
- typedef enum aoclsparse_diag_type_ aoclsparse_diag_type
  Indicates if the diagonal entries are unity.
- typedef enum aoclsparse_fill_mode_ aoclsparse_fill_mode
  Specify the matrix fill mode.
- typedef enum aoclsparse_order_ aoclsparse_order
  List of dense matrix ordering.
- typedef enum aoclsparse_status_ aoclsparse_status
  List of aoclsparse status codes definition.
- typedef enum aoclsparse_request_ aoclsparse_request
  List of request stages for sparse matrix ∗ sparse matrix.

Enumerations

- enum aoclsparse_operation_ { aoclsparse_operation_none = 111 , aoclsparse_operation_transpose = 112 , aoclsparse_operation_conjugate_transpose = 113 }

    Specify whether the matrix is to be transposed or not.
- enum aoclsparse_index_base_ { aoclsparse_index_base_zero = 0 , aoclsparse_index_base_one = 1 }

    Specify the matrix index base.
- enum aoclsparse_matrix_type_ { aoclsparse_matrix_type_general = 0 , aoclsparse_matrix_type_symmetric = 1 , aoclsparse_matrix_type_hermitian = 2 , aoclsparse_matrix_type_triangular = 3 }

    Specify the matrix type.
- enum aoclsparse_matrix_data_type_ { aoclsparse_dmat = 0 , aoclsparse_smat = 1 , aoclsparse_cmat = 2 , aoclsparse_zmat = 3 }

    Specify the matrix data type.
- enum aoclsparse_ilu_type_ { aoclsparse_ilu0 = 0 , aoclsparse_ilup = 1 }

    Specify the type of ILU factorization.
- enum aoclsparse_matrix_format_type_ {
  aoclsparse_csr_mat = 0 , aoclsparse_ell_mat = 1 , aoclsparse_ellt_mat = 2 , aoclsparse_ellt_csr_hyb_mat = 3 ,
  aoclsparse_ell_csr_hyb_mat = 4 , aoclsparse_dia_mat = 5 , aoclsparse_csr_mat_br4 = 6 , aoclsparse_csc_mat = 7 ,
  aoclsparse_coo_mat = 8 }

    Specify the matrix storage format type.
- enum aoclsparse_diag_type_ { aoclsparse_diag_type_non_unit = 0 , aoclsparse_diag_type_unit = 1 , aoclsparse_diag_type_zero = 2 }

    Indicates if the diagonal entries are unity.
- enum aoclsparse_fill_mode_ { aoclsparse_fill_mode_lower = 0 , aoclsparse_fill_mode_upper = 1 }

    Specify the matrix fill mode.
- enum aoclsparse_order_ { aoclsparse_order_row = 0 , aoclsparse_order_column = 1 }

    List of dense matrix ordering.
- enum aoclsparse_status_ {
  aoclsparse_status_success = 0 , aoclsparse_status_not_implemented = 1 , aoclsparse_status_invalid_pointer = 2 , aoclsparse_status_invalid_size = 3 ,
  aoclsparse_status_internal_error = 4 , aoclsparse_status_invalid_value = 5 , aoclsparse_status_invalid_index_value = 6 , aoclsparse_status_maxit = 7 ,
  aoclsparse_status_user_stop = 8 , aoclsparse_status_wrong_type = 9 , aoclsparse_status_memory_error = 10 , aoclsparse_status_numerical_error = 11 ,
  aoclsparse_status_invalid_operation = 12 }

    List of aoclsparse status codes definition.
- enum aoclsparse_request_ { aoclsparse_stage_nnz_count = 0 , aoclsparse_stage_finalize = 1 , aoclsparse_stage_full_computation = 2 }

    List of request stages for sparse matrix ∗ sparse matrix.

3.6.1 Detailed Description

aoclsparse_types.h defines data types used by aoclsparse

3.6.2 Macro Definition Documentation

3.6.2.1 DLL_PUBLIC `#define DLL_PUBLIC __attribute__((__visibility__("default")))`

Macro for function attribute.

The macro specifies visibility attribute of public functions

3.6.3 Typedef Documentation

3.6.3.1 aoclsparse_mat_descr `typedef struct _aoclsparse_mat_descr* aoclsparse_mat_descr`

Descriptor of the matrix.

The aoclsparse_mat_descr is a structure holding all properties of a matrix. It must be initialized using aoclsparse_create_mat_descr() and the returned descriptor must be passed to all subsequent library calls that involve the matrix. It should be destroyed at the end using aoclsparse_destroy_mat_descr().

3.6.3.2 aoclsparse_csr `typedef struct _aoclsparse_csr* aoclsparse_csr`

CSR matrix storage format.

The aoclsparse CSR matrix structure holds the CSR matrix. It must be initialized using aoclsparse_create_(d/s)csr() and the returned CSR matrix must be passed to all subsequent library calls that involve the matrix. It should be destroyed at the end using aoclsparse_destroy().

3.6.3.3 aoclsparse_matrix `typedef struct _aoclsparse_matrix* aoclsparse_matrix`

AOCL sparse matrix.

The aoclsparse matrix structure holds the all matrix storage format supported. It must be initialized using `aoclsparse_create`_(s/d/c/z)(csr/csc/coo) and the returned matrix must be passed to all subsequent library calls that involve the matrix. It should be destroyed at the end using aoclsparse_destroy().

3.6.3.4 aoclsparse_operation `typedef enum aoclsparse_operation_ aoclsparse_operation`

Specify whether the matrix is to be transposed or not.

The aoclsparse_operation indicates the operation performed with the given matrix.

3.6.3.5 aoclsparse_index_base `typedef enum aoclsparse_index_base_ aoclsparse_index_base`

Specify the matrix index base.

The aoclsparse_index_base indicates the index base of the indices. For a given aoclsparse_mat_descr, the aoclsparse_index_base can be set using aoclsparse_set_mat_index_base(). The current aoclsparse_index_base of a matrix can be obtained by aoclsparse_get_mat_index_base().

3.6.3.6 aoclsparse_matrix_type `typedef enum aoclsparse_matrix_type_ aoclsparse_matrix_type`

Specify the matrix type.

The aoclsparse_matrix_type indices the type of a matrix. For a given aoclsparse_mat_descr, the aoclsparse_matrix_type can be set using aoclsparse_set_mat_type(). The current aoclsparse_matrix_type of a matrix can be obtained by aoclsparse_get_mat_type().

3.6.3.7 aoclsparse_matrix_data_type `typedef enum aoclsparse_matrix_data_type_ aoclsparse_matrix_data_type`

Specify the matrix data type.

The aoclsparse_matrix_data_type indices the data-type of a matrix.

3.6.3.8 aoclsparse_ilu_type `typedef enum aoclsparse_ilu_type_ aoclsparse_ilu_type`

Specify the type of ILU factorization.

The aoclsparse_ilu_type indicates the type of ILU factorization like ILU0, ILU(p) etc.

3.6.3.9 aoclsparse_matrix_format_type `typedef enum aoclsparse_matrix_format_type_ aoclsparse_matrix_format_type`

Specify the matrix storage format type.

The aoclsparse_matrix_format_type indices the storage format of a sparse matrix.

3.6.3.10 aoclsparse_diag_type `typedef enum aoclsparse_diag_type_ aoclsparse_diag_type`

Indicates if the diagonal entries are unity.

The aoclsparse_diag_type indicates whether the diagonal entries of a matrix are unity or not. If aoclsparse_diag_type_unit is specified, all present diagonal values will be ignored. For a given aoclsparse_mat_descr, the aoclsparse_diag_type can be set using aoclsparse_set_mat_diag_type(). The current aoclsparse_diag_type of a matrix can be obtained by aoclsparse_get_mat_diag_type().

3.6.3.11 aoclsparse_fill_mode `typedef enum aoclsparse_fill_mode_ aoclsparse_fill_mode`

Specify the matrix fill mode.

The aoclsparse_fill_mode indicates whether the lower or the upper part is stored in a sparse triangular matrix. For a given aoclsparse_mat_descr, the aoclsparse_fill_mode can be set using aoclsparse_set_mat_fill_mode(). The current aoclsparse_fill_mode of a matrix can be obtained by aoclsparse_get_mat_fill_mode().

3.6.3.12 aoclsparse_order `typedef enum aoclsparse_order_ aoclsparse_order`

List of dense matrix ordering.

This is a list of supported aoclsparse_order types that are used to describe the memory layout of a dense matrix

3.6.3.13 aoclsparse_status `typedef enum aoclsparse_status_ aoclsparse_status`

List of aoclsparse status codes definition.

List of aoclsparse_status values returned by the functions in the library.

3.6.3.14 aoclsparse_request `typedef enum aoclsparse_request_ aoclsparse_request`

List of request stages for sparse matrix $*$ sparse matrix.

This is a list of the aoclsparse_request types that are used by the aoclsparse_csr2m funtion.

3.6.4 Enumeration Type Documentation

3.6.4.1 aoclsparse_operation_ `enum aoclsparse_operation_`

Specify whether the matrix is to be transposed or not.

The aoclsparse_operation indicates the operation performed with the given matrix.

Enumerator

aoclsparse_operation_none	Operate with matrix.
aoclsparse_operation_transpose	Operate with transpose.
aoclsparse_operation_conjugate_transpose	Operate with conj. transpose.

3.6.4.2 aoclsparse_index_base_ `enum aoclsparse_index_base_`

Specify the matrix index base.

The aoclsparse_index_base indicates the index base of the indices. For a given aoclsparse_mat_descr, the aoclsparse_index_base can be set using aoclsparse_set_mat_index_base(). The current aoclsparse_index_base of a matrix can be obtained by aoclsparse_get_mat_index_base().

Enumerator

aoclsparse_index_base_zero	zero based indexing.
aoclsparse_index_base_one	one based indexing.

3.6.4.3 aoclsparse_matrix_type_ `enum aoclsparse_matrix_type_`

Specify the matrix type.

The aoclsparse_matrix_type indices the type of a matrix. For a given aoclsparse_mat_descr, the aoclsparse_matrix_type can be set using aoclsparse_set_mat_type(). The current aoclsparse_matrix_type of a matrix can be obtained by aoclsparse_get_mat_type().

Enumerator

aoclsparse_matrix_type_general	general matrix type.
aoclsparse_matrix_type_symmetric	symmetric matrix type.
aoclsparse_matrix_type_hermitian	hermitian matrix type.
aoclsparse_matrix_type_triangular	triangular matrix type.

3.6.4.4 aoclsparse_matrix_data_type_ `enum aoclsparse_matrix_data_type_`

Specify the matrix data type.

The aoclsparse_matrix_data_type indices the data-type of a matrix.

Enumerator

aoclsparse_dmat	double precision data.
aoclsparse_smat	single precision data.
aoclsparse_cmat	single precision complex data.
aoclsparse_zmat	double precision complex data.

3.6.4.5 aoclsparse_ilu_type_ `enum aoclsparse_ilu_type_`

Specify the type of ILU factorization.

The aoclsparse_ilu_type indicates the type of ILU factorization like ILU0, ILU(p) etc.

Enumerator

aoclsparse_ilu0	ILU0.
aoclsparse_ilup	ILU(p).

3.6.4.6 aoclsparse_matrix_format_type_ `enum aoclsparse_matrix_format_type_`

Specify the matrix storage format type.

The aoclsparse_matrix_format_type indices the storage format of a sparse matrix.

Enumerator

aoclsparse_csr_mat	CSR format.
aoclsparse_ell_mat	ELLPACK format.
aoclsparse_ellt_mat	ELLPACK format stored as transpose format.
aoclsparse_ellt_csr_hyb_mat	ELLPACK transpose + CSR hybrid format.
aoclsparse_ell_csr_hyb_mat	ELLPACK + CSR hybrid format.
aoclsparse_dia_mat	diag format.
aoclsparse_csr_mat_br4	Modified CSR format for AVX2 double.
aoclsparse_csc_mat	CSC format.
aoclsparse_coo_mat	COO format.

3.6.4.7 aoclsparse_diag_type_ `enum aoclsparse_diag_type_`

Indicates if the diagonal entries are unity.

The aoclsparse_diag_type indicates whether the diagonal entries of a matrix are unity or not. If aoclsparse_diag_type_unit is specified, all present diagonal values will be ignored. For a given aoclsparse_mat_descr, the aoclsparse_diag_type can be set using aoclsparse_set_mat_diag_type(). The current aoclsparse_diag_type of a matrix can be obtained by aoclsparse_get_mat_diag_type().

Enumerator

aoclsparse_diag_type_non_unit	diagonal entries are non-unity.
aoclsparse_diag_type_unit	diagonal entries are unity
aoclsparse_diag_type_zero	ignore diagonal entries: for strict L/U matrices

3.6.4.8 aoclsparse_fill_mode_ `enum aoclsparse_fill_mode_`

Specify the matrix fill mode.

The aoclsparse_fill_mode indicates whether the lower or the upper part is stored in a sparse triangular matrix. For a given aoclsparse_mat_descr, the aoclsparse_fill_mode can be set using aoclsparse_set_mat_fill_mode(). The current aoclsparse_fill_mode of a matrix can be obtained by aoclsparse_get_mat_fill_mode().

Enumerator

aoclsparse_fill_mode_lower	lower triangular part is stored.
aoclsparse_fill_mode_upper	upper triangular part is stored.

3.6.4.9 aoclsparse_order_ `enum aoclsparse_order_`

List of dense matrix ordering.

This is a list of supported aoclsparse_order types that are used to describe the memory layout of a dense matrix

Enumerator

aoclsparse_order_row	Row major.
aoclsparse_order_column	Column major.

3.6.4.10 aoclsparse_status_ `enum aoclsparse_status_`

List of aoclsparse status codes definition.

List of aoclsparse_status values returned by the functions in the library.

Enumerator

aoclsparse_status_success	success.
aoclsparse_status_not_implemented	functionality is not implemented.
aoclsparse_status_invalid_pointer	invalid pointer parameter.
aoclsparse_status_invalid_size	invalid size parameter.
aoclsparse_status_internal_error	internal library failure.
aoclsparse_status_invalid_value	invalid parameter value.
aoclsparse_status_invalid_index_value	invalid index value.
aoclsparse_status_maxit	function stopped after reaching number of iteration limit.
aoclsparse_status_user_stop	user requested termination.
aoclsparse_status_wrong_type	function called on the wrong type (double/float).
aoclsparse_status_memory_error	memory allocation failure.
aoclsparse_status_numerical_error	numerical error, e.g., matrix is not positive definite, divide-by-zero error
aoclsparse_status_invalid_operation	cannot proceed with the request at this point.

3.6.4.11 aoclsparse_request_ `enum aoclsparse_request_`

List of request stages for sparse matrix $*$ sparse matrix.

This is a list of the aoclsparse_request types that are used by the aoclsparse_csr2m funtion.

Enumerator

aoclsparse_stage_nnz_count	Only rowIndex array of the CSR matrix is computed internally.
aoclsparse_stage_finalize	Finalize computation. Has to be called only after csr2m call with aoclsparse_stage_nnz_count parameter.
aoclsparse_stage_full_computation	Perform the entire computation in a single step.

Index