aocl-sparse API Guide

Version v3.1.1.0

Table of Contents

File Index

File List

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

Introduction

aocl-sparse is a library that contains basic linear algebra subroutines for sparse matrices and vectors optimized for AMD EPYC family of processors. It is designed to be used with C and C++.
The current functionality of aocl-sparse is organized in the following categories:

- Sparse Level 3 Functions describe operations between a matrix in sparse format and a matrix in dense/sparse format.
- Sparse Level 2 Functions describe operations between a matrix in sparse format and a vector in dense format.
- Sparse Format Conversion Functions describe operations on a matrix in sparse format to obtain a different matrix format.
- Sparse Auxiliary Functions describe auxiliary functions.

AuxiliaryFunctions

aoclsparse_auxiliary.h provides auxilary functions in aoclsparse

Functions

- **DLL_PUBLIC aoclsparse_status aoclsparse_get_version** (**aoclsparse_int** *version)
  Get aoclsparse 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_mat_csr** (**aoclsparse_mat_csr** &csr, **aoclsparse_index_base** base, **aoclsparse_int** m, **aoclsparse_int** n, **aoclsparse_int** csr_nnz, **aoclsparse_int** *csr_row_ptr, **aoclsparse_int** *csr_col_ind, void *csr_val)
  Update a `CSR` matrix structure.

- **DLL_PUBLIC aoclsparse_status aoclsparse_export_mat_csr** (**aoclsparse_mat_csr** &csr, **aoclsparse_index_base** *base, **aoclsparse_int** *M, **aoclsparse_int** *N, **aoclsparse_int** *csr_nnz, **aoclsparse_int** **csr_row_ptr, **aoclsparse_int** **csr_col_ind, void **csr_val)
  Export a `CSR` matrix structure.

- **DLL_PUBLIC aoclsparse_status aoclsparse_destroy_mat_csr** (**aoclsparse_mat_csr** csr)
  Destroy a `CSR` matrix structure.

Detailed Description

aoclsparse_auxiliary.h provides auxilary functions in aoclsparse

---

Function Documentation

DLL_PUBLIC aoclsparse_status aoclsparse_get_version (aoclsparse_int * *version*)

Get aoclsparse version.

`aoclsparse_get_version` gets the aoclsparse library version number.

- patch = version % 100
- minor = version / 100 % 1000
- major = version / 100000

Parameters

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

Return values

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

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.

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.

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_*	`descr` is invalid.

invalid_pointer	

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.

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**.

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.

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**.

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.

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**.

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** or **aoclsparse_diag_type_non_unit**.

Parameters

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

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.

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**.

DLL_PUBLIC aoclsparse_status aoclsparse_create_mat_csr (aoclsparse_mat_csr & *csr*, aoclsparse_index_base *base*, aoclsparse_int *m*, aoclsparse_int *n*, aoclsparse_int *csr_nnz*, aoclsparse_int * *csr_row_ptr*, aoclsparse_int * *csr_col_ind*, void * *csr_val*)

Update a `CSR` matrix structure.

`aoclsparse_create_mat_csr` updates a structure that holds the matrix in `CSR` storage format. It should be destroyed at the end using **aoclsparse_destroy_mat_csr()**.

Parameters

in,out	*csr*	the pointer to the CSR sparse matrix.
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	*csr_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 of `nnz` elements containing the column indices of the sparse CSR matrix.
in	*csr_val*	array of `nnz` elements of the sparse CSR matrix.

Return values

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

DLL_PUBLIC aoclsparse_status aoclsparse_export_mat_csr (aoclsparse_mat_csr & *csr*, aoclsparse_index_base * *base*, aoclsparse_int * *M*, aoclsparse_int * *N*, aoclsparse_int * *csr_nnz*, aoclsparse_int ** *csr_row_ptr*, aoclsparse_int ** *csr_col_ind*, void ** *csr_val*)

Export a `CSR` matrix structure.

`aoclsparse_export_mat_csr` exports a structure that holds the matrix in `CSR` storage format.

Parameters

in	*csr*	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	*csr_nnz*	number of non-zero entries of the sparse CSR matrix.
out	*csr_row_ptr*	array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
out	*csr_col_ind*	array of `nnz` elements containing the column indices of the sparse CSR matrix.
out	*csr_val*	array of `nnz` elements of the sparse CSR matrix.

Return values

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

DLL_PUBLIC aoclsparse_status aoclsparse_destroy_mat_csr (aoclsparse_mat_csr *csr*)

Destroy a `CSR` matrix structure.

`aoclsparse_destroy_mat_csr` destroys a structure that holds the matrix in `CSR` storage format.

Parameters

in	*csr*	the pointer to the CSR sparse matrix.

Return values

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

Conversion Functions

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, **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_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_scsr2ell** (**aoclsparse_int** m, 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_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_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_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 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 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_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_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_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.

Detailed Description

aoclsparse_convert.h provides Sparse Format conversion Subprograms

---

Function Documentation

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_*	an internal error occurred.

internal_error	

DLL_PUBLIC aoclsparse_status aoclsparse_scsr2ell (aoclsparse_int *m*, 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 $nnz_{ELL} = m \cdot ell_width$. The number of ELL non-zero elements per row is obtained by **aoclsparse_csr2ell_width()**.

Parameters

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

DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2ell (aoclsparse_int *m*, 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 $nnz_{ELL} = m \cdot ell_width$. The number of ELL non-zero elements per row is obtained by **aoclsparse_csr2ell_width()**.

Parameters

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

DLL_PUBLIC aoclsparse_status aoclsparse_csr2dia_ndiag (aoclsparse_int *m*, aoclsparse_int *n*, 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	*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.

DLL_PUBLIC aoclsparse_status aoclsparse_scsr2dia (aoclsparse_int *m*, aoclsparse_int *n*, 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()**.

Parameters

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

DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2dia (aoclsparse_int *m*, aoclsparse_int *n*, 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()**.

Parameters

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

DLL_PUBLIC aoclsparse_status aoclsparse_csr2bsr_nnz (aoclsparse_int *m*, aoclsparse_int *n*, 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	*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.

DLL_PUBLIC aoclsparse_status aoclsparse_scsr2bsr (aoclsparse_int *m*, aoclsparse_int *n*, 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` .

Parameters

in	*m*	number of rows in the sparse CSR matrix.
in	*n*	number of columns in the sparse CSR matrix.
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_*	`m` or `n` or `block_dim` is invalid.

invalid_size	
aoclsparse_status_invalid_pointer	`bsr_val`,`bsr_row_ptr`,`bsr_col_ind`,`csr_val`,`csr_row_ptr` or `csr_col_ind` pointer is invalid.

DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2bsr (aoclsparse_int *m*, aoclsparse_int *n*, 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` .

Parameters

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

DLL_PUBLIC aoclsparse_status aoclsparse_scsr2csc (aoclsparse_int *m*, aoclsparse_int *n*, aoclsparse_int *nnz*, 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.

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	*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.

DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2csc (aoclsparse_int *m*, aoclsparse_int *n*, aoclsparse_int *nnz*, 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.

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	*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.

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** and also any valid value of the **aoclsparse_index_base**.
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`)
out	*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.

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** and also any valid value of the **aoclsparse_index_base**.
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`)
out	*ld*	leading dimension of dense array `A` .
in	*order*	memory layout of a dense matrix `A` .

Return values

*aoclsparse_status_*	the operation completed successfully.

success	
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.

Sparse Level 2 & 3 Functions

aoclsparse_functions.h provides Sparse Linear Algebra Subprograms of Level 1, 2 and 3, for AMD CPU hardware.

Functions

- **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 & 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 & 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_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_scsrmm** (**aoclsparse_operation** trans_A, const float *alpha, const **aoclsparse_mat_csr** csr, 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** trans_A, const double *alpha, const **aoclsparse_mat_csr** csr, 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_dcsr2m** (**aoclsparse_operation** trans_A, const **aoclsparse_mat_descr** descrA, const **aoclsparse_mat_csr** csrA, **aoclsparse_operation** trans_B, const **aoclsparse_mat_descr** descrB, const **aoclsparse_mat_csr** csrB, const **aoclsparse_request** request, **aoclsparse_mat_csr** *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_mat_csr** csrA, **aoclsparse_operation** trans_B, const **aoclsparse_mat_descr** descrB, const **aoclsparse_mat_csr** csrB, const **aoclsparse_request** request, **aoclsparse_mat_csr** *csrC)
  Sparse matrix Sparse matrix multiplication using CSR storage format for single and double precision datatypes.

Detailed Description

aoclsparse_functions.h provides Sparse Linear Algebra Subprograms of Level 1, 2 and 3, for AMD CPU hardware.

Function Documentation

**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 & 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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

```
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];

    for(j = csr_row_ptr[i]; j < csr_row_ptr[i + 1]; ++j)
    {
        y[i] = y[i] + alpha * csr_val[j] * x[csr_col_ind[j]];
    }
}
```

Note

Currently, only `trans` == **aoclsparse_operation_none** is supported. Currently, for **aoclsparse_matrix_type** == **aoclsparse_matrix_type_symmetric**, only lower triangular matrices are supported.

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_*	`trans` != **aoclsparse_operation_none** or **aoclsparse_matrix_type** !=

not_implemented	**aoclsparse_matrix_type_general**. **aoclsparse_matrix_type** != **aoclsparse_matrix_type_symmetric**.

Example

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

```
// Compute y = Ax
aoclsparse_scsrmv(aoclsparse_operation_none,
                  &alpha,
                  m,
                  n,
                  nnz,
                  csr_val,
                  csr_col_ind,
                  csr_row_ptr,
                  descr,
                  x,
                  &beta,
                  y);

// Do more work
// ...
```

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 & 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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

```
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];

    for(j = csr_row_ptr[i]; j < csr_row_ptr[i + 1]; ++j)
    {
        y[i] = y[i] + alpha * csr_val[j] * x[csr_col_ind[j]];
    }
}
```

Note

Currently, only `trans` == **aoclsparse_operation_none** is supported. Currently, for **aoclsparse_matrix_type** == **aoclsparse_matrix_type_symmetric**, only lower triangular matrices are supported.

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` != **aoclsparse_operation_none** or **aoclsparse_matrix_type** != **aoclsparse_matrix_type_general**. **aoclsparse_matrix_type** != **aoclsparse_matrix_type_symmetric**.

Example

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

```
// Compute y = Ax
aoclsparse_scsrmv(aoclsparse_operation_none,
                  &alpha,
                  m,
                  n,
                  nnz,
                  csr_val,
                  csr_col_ind,
                  csr_row_ptr,
                  descr,
                  x,
                  &beta,
                  y);

// Do more work
// ...
```

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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

```
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];

    for(p = 0; p < ell_width; ++p)
    {
        idx = p * m + i;

        if((ell_col_ind[idx] >= 0) && (ell_col_ind[idx] < n))
        {
            y[i] = y[i] + alpha * ell_val[idx] * x[ell_col_ind[idx]];
        }
    }
}
```

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. Currently, only **aoclsparse_matrix_type_general** is 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**.

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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

```
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];

    for(p = 0; p < ell_width; ++p)
    {
        idx = p * m + i;

        if((ell_col_ind[idx] >= 0) && (ell_col_ind[idx] < n))
        {
            y[i] = y[i] + alpha * ell_val[idx] * x[ell_col_ind[idx]];
        }
    }
}
```

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. Currently, only **aoclsparse_matrix_type_general** is 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**.

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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

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 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. Currently, only **aoclsparse_matrix_type_general** is supported.
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**.

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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

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 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. Currently, only **aoclsparse_matrix_type_general** is supported.
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**.

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 $(mb \cdot bsr_dim) \times (nb \cdot bsr_dim)$ 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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

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. Currently, only **aoclsparse_matrix_type_general** is 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**.

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 $(mb \cdot bsr_dim) \times (nb \cdot bsr_dim)$ 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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

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. Currently, only **aoclsparse_matrix_type_general** is 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**.

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_csrsv` 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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

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** != **aoclsparse_matrix_type_general**.

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_csrsv` 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

$$op(A) = \begin{cases} A, & if trans == aoclsparse_operation_none \\ A^T, & if trans == aoclsparse_operation_transpose \\ A^H, & if trans == aoclsparse_operation_conjugate_transpose \end{cases}$$

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** != **aoclsparse_matrix_type_general**.

DLL_PUBLIC aoclsparse_status aoclsparse_scsrmm (aoclsparse_operation *trans_A*, const float * *alpha*, const aoclsparse_mat_csr *csr*, 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_csrmm` multiplies the scalar α with a sparse $m \times k$ matrix A, defined in CSR storage format, and the dense $k \times n$ matrix B and adds the result to the dense $m \times n$ matrix C that is multiplied by the scalar β, such that

$$C := \alpha \cdot op(A) \cdot B + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & if trans_A == aoclsparse_operation_none \\ A^T, & if trans_A == aoclsparse_operation_transpose \\ A^H, & if trans_A == aoclsparse_operation_conjugate_transpose \end{cases}$$

```
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	*trans_A*	matrix A operation type.
in	*alpha*	scalar α.
in	*csr*	sparse CSR matrix A structure.
in	*descr*	descriptor of the sparse CSR matrix A. Currently, only **aoclsparse_matrix_type_general** is 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	`m`, `n`, `k`, `nnz`, `ldb` or `ldc` is invalid.
aoclsparse_status_invalid_pointer	`descr`, `alpha`, `csr`, `B`, `beta` or `C` pointer is invalid.
aoclsparse_status_not_implemented	**aoclsparse_matrix_type** != **aoclsparse_matrix_type_general**.

DLL_PUBLIC aoclsparse_status aoclsparse_dcsrmm (aoclsparse_operation *trans_A*, const double * *alpha*, const aoclsparse_mat_csr *csr*, 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_csrmm` multiplies the scalar α with a sparse $m \times k$ matrix A, defined in CSR storage format, and the dense $k \times n$ matrix B and adds the result to the dense $m \times n$ matrix C that is multiplied by the scalar β, such that

$$C := \alpha \cdot op(A) \cdot B + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & if trans_A == aoclsparse_operation_none \\ A^T, & if trans_A == aoclsparse_operation_transpose \\ A^H, & if trans_A == aoclsparse_operation_conjugate_transpose \end{cases}$$

```
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	*trans_A*	matrix A operation type.
in	*alpha*	scalar α.
in	*csr*	sparse CSR matrix A structure.
in	*descr*	descriptor of the sparse CSR matrix A. Currently, only **aoclsparse_matrix_type_general** is 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	`m` , `n` , `k` , `nnz` , `ldb` or `ldc` is invalid.
aoclsparse_status_invalid_pointer	`descr` , `alpha` , `csr` , `B` , `beta` or `C` pointer is invalid.
aoclsparse_status_not_implemented	**aoclsparse_matrix_type** != **aoclsparse_matrix_type_general**.

DLL_PUBLIC aoclsparse_status aoclsparse_dcsr2m (aoclsparse_operation *trans_A*, const aoclsparse_mat_descr *descrA*, const aoclsparse_mat_csr *csrA*, aoclsparse_operation *trans_B*, const aoclsparse_mat_descr *descrB*, const aoclsparse_mat_csr *csrB*, const aoclsparse_request *request*, aoclsparse_mat_csr * *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

$$op(A) = \begin{cases} A, & if trans_A == aoclsparse_operation_none \\ A^T, & if trans_A == aoclsparse_operation_transpose \\ A^H, & if trans_A == aoclsparse_operation_conjugate_transpose \end{cases}$$

and

$$op(B) = \begin{cases} B, & if trans_B == aoclsparse_operation_none \\ B^T, & if trans_B == aoclsparse_operation_transpose \\ B^H, & if trans_B == aoclsparse_operation_conjugate_transpose \end{cases}$$

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_*	**aoclsparse_matrix_type** != **aoclsparse_matrix_type_general**.

not_implemented	

Example

Shows multiplication of 2 sparse matrices to give a newly allocated sparse matrix

```
    aoclsparse_mat_csr csrA;
    aoclsparse_create_mat_csr(csrA, base, M, K, nnz_A, csr_row_ptr_A.data(),
csr_col_ind_A.data(), csr_val_A.data());
    aoclsparse_mat_csr csrB;
    aoclsparse_create_mat_csr(csrB, base, K, N, nnz_B, csr_row_ptr_B.data(),
csr_col_ind_B.data(), csr_val_B.data());

aoclsparse_mat_csr 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);
```

DLL_PUBLIC aoclsparse_status aoclsparse_scsr2m (aoclsparse_operation *trans_A*, const aoclsparse_mat_descr *descrA*, const aoclsparse_mat_csr *csrA*, aoclsparse_operation *trans_B*, const aoclsparse_mat_descr *descrB*, const aoclsparse_mat_csr *csrB*, const aoclsparse_request *request*, aoclsparse_mat_csr * *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

$$op(A) = \begin{cases} A, & if trans_A == aoclsparse_operation_none \\ A^T, & if trans_A == aoclsparse_operation_transpose \\ A^H, & if trans_A == aoclsparse_operation_conjugate_transpose \end{cases}$$

and

$$op(B) = \begin{cases} B, & if trans_B == aoclsparse_operation_none \\ B^T, & if trans_B == aoclsparse_operation_transpose \\ B^H, & if trans_B == aoclsparse_operation_conjugate_transpose \end{cases}$$

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** != **aoclsparse_matrix_type_general**.

Example

Shows multiplication of 2 sparse matrices to give a newly allocated sparse matrix

```
    aoclsparse_mat_csr csrA;
    aoclsparse_create_mat_csr(csrA, base, M, K, nnz_A, csr_row_ptr_A.data(),
csr_col_ind_A.data(), csr_val_A.data());
    aoclsparse_mat_csr csrB;
    aoclsparse_create_mat_csr(csrB, base, K, N, nnz_B, csr_row_ptr_B.data(),
csr_col_ind_B.data(), csr_val_B.data());

aoclsparse_mat_csr 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);
```

aoclsparse_types.h

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_mat_csr * **aoclsparse_mat_csr**
  CSR matrix storage format.

- 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_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_diag_type_** { **aoclsparse_diag_type_non_unit** = 0, **aoclsparse_diag_type_unit** = 1 }
  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 }
  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.*

Detailed Description

aoclsparse_types.h defines data types used by aoclsparse

Macro Definition Documentation

#define DLL_PUBLIC __attribute__((__visibility__("default")))

Macro for function attribute.

The macro specifies visibility attribute of public functions

Typedef Documentation

typedef struct _aoclsparse_mat_descr* aoclsparse_mat_descr

Descriptor of the matrix.

The aoclSPARSE matrix descriptor 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()**.

typedef struct _aoclsparse_mat_csr* aoclsparse_mat_csr

CSR matrix storage format.

The aoclSPARSE CSR matrix structure holds the CSR matrix. It must be initialized using **aoclsparse_create_mat_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_mat_csr()**.

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.

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()**.

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()**.

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()**.

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()**.

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

typedef enum aoclsparse_status_ aoclsparse_status

List of aoclsparse status codes definition.

This is a list of the **aoclsparse_status** types that are used by the aoclSPARSE library.

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.

Enumeration Type Documentation

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.

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.

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.

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

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.

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.

enum aoclsparse_status_

List of aoclsparse status codes definition.

This is a list of the **aoclsparse_status** types that are used by the aoclSPARSE library.

Enumerator:

aoclsparse_status_success	success.
aoclsparse_status_not_implemented	function is not implemented.
aoclsparse_status_invalid_pointer	invalid pointer parameter.
aoclsparse_status_invalid_size	invalid size parameter.
aoclsparse_status_internal_error	other internal library failure.
aoclsparse_status_invalid_value	invalid value parameter.

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.