AOCL-Sparse API Guide

version 4.1.0.0 (22 May 2023)

Generated by Doxygen 1.9.1

1 AOCL-Sparse 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:

1. Sparse Level 3 functions describe the operations between a matrix in sparse format and a matrix in dense/sparse format.
2. Sparse Level 2 functions describe the operations between a matrix in sparse format and a vector in dense format.
3. Sparse Solver functions that perform matrix factorization and solution phases.

4. Analysis and execute functionalities for performing optimized Sparse Matrix-Dense Vector multiplication and Sparse Solver.
5. Sparse Format Conversion functions describe operations on a matrix in sparse format to obtain a different matrix format.
6. Sparse Auxiliary Functions describe auxiliary functions.

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.

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*	sparse matrix in CSR format 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 CSR matrix. Currently, only 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.

3.2 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_export_mat_csr (aoclsparse_matrix &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_create_scsr (aoclsparse_matrix &mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int csr_nnz, aoclsparse_int ∗csr_row_ptr, aoclsparse_int ∗csr_col_ptr, float ∗csr_val)

  Update a CSR matrix structure.
- DLL_PUBLIC aoclsparse_status aoclsparse_create_dcsr (aoclsparse_matrix &mat, aoclsparse_index_base base, aoclsparse_int M, aoclsparse_int N, aoclsparse_int csr_nnz, aoclsparse_int ∗csr_row_ptr, aoclsparse_int ∗csr_col_ptr, double ∗csr_val)

  Update a CSR matrix structure.

- DLL_PUBLIC aoclsparse_status aoclsparse_destroy (aoclsparse_matrix &mat)

  Destroy a sparse matrix structure.

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 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.

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.

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 csr_nnz,
```

```
            aoclsparse_int * csr_row_ptr,
            aoclsparse_int * csr_col_ptr,
            float * csr_val )
```

Update a CSR matrix structure.

aoclsparse_create_(s/d)csr updates a structure that holds the matrix in CSR storage format. It should be destroyed at the end using aoclsparse_destroy().

Parameters

in,out	*mat*	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_ptr*	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.

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 csr_nnz,
            aoclsparse_int * csr_row_ptr,
            aoclsparse_int * csr_col_ptr,
            double * csr_val )
```

Update a CSR matrix structure.

aoclsparse_create_(s/d)csr updates a structure that holds the matrix in CSR storage format. It should be destroyed at the end using aoclsparse_destroy().

Parameters

in,out	*mat*	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_ptr*	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.

3.2.2.15 aoclsparse_export_mat_csr()

```
DLL_PUBLIC aoclsparse_status aoclsparse_export_mat_csr (
    aoclsparse_matrix & 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.

3.2.2.16 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.

3.3 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, 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.

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_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().

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.

3.3.2.3 aoclsparse_dcsr2ell()

```
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 $\text{nnz}_{\text{ELL}} = m \cdot$ 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.

3.3.2.4 aoclsparse_csr2dia_ndiag()

```
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.

3.3.2.5 aoclsparse_scsr2dia()

```
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.

Return values

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

3.3.2.6 aoclsparse_dcsr2dia()

```
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.

3.3.2.7 aoclsparse_csr2bsr_nnz()

```
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.

3.3.2.8 aoclsparse_scsr2bsr()

```
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.

Parameters

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 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.

3.3.2.10 aoclsparse_scsr2csc()

```
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.

3.3.2.11 aoclsparse_dcsr2csc() 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.

3.3.2.12 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 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.

3.3.2.13 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 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.

3.4 aoclsparse_functions.h File Reference

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_smv (aoclsparse_operation op, const float *alpha, aoclsparse↩ _matrix A, const aoclsparse_mat_descr descr, const float *x, const float *beta, float *y)

  Single & Double precision sparse matrix vector multiplication using optimized mv routines.
- 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)

  Single & Double precision sparse matrix vector multiplication using optimized mv routines.

- 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 solve for 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 solve for 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 solve for 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 solve for single and double data precisions.

- DLL_PUBLIC aoclsparse_status aoclsparse_scsrmm (aoclsparse_operation trans_A, const float *alpha, const aoclsparse_matrix 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_matrix 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_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.

3.4.1 Detailed Description

aoclsparse_functions.h provides sparse linear algebra subprograms of level 1, 2 and 3, for AMD CPU hardware.

3.4.2 Function Documentation

3.4.2.1 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 & 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

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

```
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` is not aoclsparse_operation_none or 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.

```
// 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
// ...
```

3.4.2.2 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 & 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

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

```
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` is not aoclsparse_operation_none or 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.

```
// 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
// ...
```

3.4.2.3 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

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

```
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.

3.4.2.4 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

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

```
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.

Parameters

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.

3.4.2.5 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. 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.

3.4.2.6 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. 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.

3.4.2.7 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. 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.

3.4.2.8 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. 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.

3.4.2.9 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 )
```

Single & Double precision sparse matrix vector multiplication using optimized mv routines.

`aoclsparse_`?mv 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

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

Note

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

Parameters

`in`	*op*	matrix operation type.
`in`	*alpha*	scalar α.
`in`	*A*	the sparse matrix structure that is created using aoclsparse_create_dcsr.
`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`, `internal` structures related to the sparse matrix `A`, `x`, `beta` or `y` has an invalid pointer.
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_dmv(trans,
```

```
                &alpha,
                A,
                descr,
                x,
                &beta,
                y);
// Do more work
// ...
```

3.4.2.10 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 )
```

Single & Double precision sparse matrix vector multiplication using optimized mv routines.

aoclsparse_?mv 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

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

Note

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

Parameters

in	*op*	matrix operation type.
in	*alpha*	scalar α.
in	*A*	the sparse matrix structure that is created using aoclsparse_create_dcsr.
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, internal structures related to the sparse matrix A, x, beta or y has an invalid pointer.

Return values

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_dmv(trans,
               &alpha,
               A,
               descr,
               x,
               &beta,
               y);
// Do more work
// ...
```

3.4.2.11 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.

Parameters

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.

3.4.2.12 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.

3.4.2.13 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 solve for single and double data precisions.

aoclsparse_strsv and aoclsparse_dtrsv solve a 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 transposition operation,

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

Note

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 considered to all be unitary.

The input matrix need not be (upper or lower) triangular matrix, `descr fill_mode` 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 optimiziation for the matrix will be done by the solver on entry.

There is `_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 256bit implementation. `kid=2` Kernel Templated version using AVX/AVX2 extensions (analog to `kid=1`). `kid=3` Kernel Templated version using AVX512F/AVX512VL and AXV512DQ extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

`in`	*trans*	matrix operation type, either aoclsparse_operation_none or aoclsparse_operation_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.
`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 opteration is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.14 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 solve for single and double data precisions.

`aoclsparse_strsv` and `aoclsparse_dtrsv` solve a 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 transposition operation,

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

Note

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 considered to all be unitary.

The input matrix need not be (upper or lower) triangular matrix, `descr` `fill_mode` 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 optimiziation for the matrix will be done by the solver on entry.

There is `_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 256bit implementation. `kid=2` Kernel Templated version using AVX/AVX2 extensions (analog to `kid=1`). `kid=3` Kernel Templated version using AVX512F/AVX512VL and AXV512DQ extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

`in`	*trans*	matrix operation type, either aoclsparse_operation_none or aoclsparse_operation_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.
`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.

Return values

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 opteration is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.15 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 solve for single and double data precisions.

`aoclsparse_strsv` and `aoclsparse_dtrsv` solve a 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 transposition operation,

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

Note

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 considered to all be unitary.

The input matrix need not be (upper or lower) triangular matrix, `descr` `fill_mode` 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 optimiziation for the matrix will be done by the solver on entry.

There is `_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 256bit implementation. `kid=2` Kernel Templated version using AVX/AVX2 extensions (analog to `kid=1`). `kid=3` Kernel Templated version using AVX512F/AVX512VL and AXV512DQ extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

in	*trans*	matrix operation type, either aoclsparse_operation_none or aoclsparse_operation_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.
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 opteration is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.16 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 solve for single and double data precisions.

aoclsparse_strsv and aoclsparse_dtrsv solve a 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 transposition operation,

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

Note

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 considered to all be unitary.

The input matrix need not be (upper or lower) triangular matrix, `descr` `fill_mode` 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 optimiziation for the matrix will be done by the solver on entry.

There is `_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 256bit implementation. `kid=2` Kernel Templated version using AVX/AVX2 extensions (analog to `kid=1`). `kid=3` Kernel Templated version using AVX512F/AVX512VL and AXV512DQ extensions. Any other Kernel ID value will default to `kid=0`.

Parameters

`in`	*trans*	matrix operation type, either aoclsparse_operation_none or aoclsparse_operation_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.
`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 opteration is not yet implemented.
other	possible failure values from a call to aoclsparse_optimize.

3.4.2.17 aoclsparse_scsrmm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_scsrmm (
        aoclsparse_operation trans_A,
        const float * alpha,
        const aoclsparse_matrix 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

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

```
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	*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 is not aoclsparse_matrix_type_general.

3.4.2.18 aoclsparse_dcsrmm()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsrmm (
        aoclsparse_operation trans_A,
        const double * alpha,
        const aoclsparse_matrix 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

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

```
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	*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.

Return values

aoclsparse_status_not_implemented	aoclsparse_matrix_type is not aoclsparse_matrix_type_general.

3.4.2.19 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.

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);
```

3.4.2.20 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.
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.

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);
```

3.4.2.21 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.
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.

Refer to ILU Example from tests/include.

3.4.2.22 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.
`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.

Refer to ILU Example from tests/include.

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	$i = 20$

Option name	Type	Default value
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.

Return values

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.
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.
`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`.

Parameters

`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 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 }

  Specify the matrix storage format 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 , 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_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.

3.6.3.4 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().

3.6.3.5 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().

3.6.3.6 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.

3.6.3.7 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.

3.6.3.8 aoclsparse_matrix_format_type `typedef enum aoclsparse_matrix_format_type_ aoclsparse_matrix_format_typ`

Specify the matrix storage format type.

The aoclsparse_matrix_format_type indices the storage format of a sparse matrix.

3.6.3.9 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().

3.6.3.10 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().

3.6.3.11 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

3.6.3.12 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.

3.6.3.13 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.

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

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