

# AOCL-Sparse API Guide

v4.0.0.0

Generated by Doxygen 1.8.17



<b>1 AOCL-Sparse Introduction</b>	<b>1</b>
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 File Documentation</b>	<b>5</b>
3.1 aoclsparse_analysis.h File Reference	5
3.1.1 Detailed Description	5
3.1.2 Function Documentation	5
3.1.2.1 aoclsparse_optimize()	5
3.1.2.2 aoclsparse_set_mv_hint()	6
3.1.2.3 aoclsparse_set_lu_smoother_hint()	6
3.2 aoclsparse_auxiliary.h File Reference	7
3.2.1 Detailed Description	8
3.2.2 Function Documentation	8
3.2.2.1 aoclsparse_get_version()	8
3.2.2.2 aoclsparse_create_mat_descr()	9
3.2.2.3 aoclsparse_copy_mat_descr()	9
3.2.2.4 aoclsparse_destroy_mat_descr()	10
3.2.2.5 aoclsparse_set_mat_index_base()	10
3.2.2.6 aoclsparse_get_mat_index_base()	11
3.2.2.7 aoclsparse_set_mat_type()	11
3.2.2.8 aoclsparse_get_mat_type()	11
3.2.2.9 aoclsparse_set_mat_fill_mode()	12
3.2.2.10 aoclsparse_get_mat_fill_mode()	12
3.2.2.11 aoclsparse_set_mat_diag_type()	13
3.2.2.12 aoclsparse_get_mat_diag_type()	13
3.2.2.13 aoclsparse_create_scsr()	14
3.2.2.14 aoclsparse_create_dcsr()	14
3.2.2.15 aoclsparse_export_mat_csr()	15
3.2.2.16 aoclsparse_destroy()	16
3.3 aoclsparse_convert.h File Reference	16
3.3.1 Detailed Description	18
3.3.2 Function Documentation	18
3.3.2.1 aoclsparse_csr2ell_width()	18
3.3.2.2 aoclsparse_scsr2ell()	18
3.3.2.3 aoclsparse_dcsr2ell()	19
3.3.2.4 aoclsparse_csr2dia_ndiag()	20
3.3.2.5 aoclsparse_scsr2dia()	21
3.3.2.6 aoclsparse_dcsr2dia()	21
3.3.2.7 aoclsparse_csr2bsr_nnz()	22
3.3.2.8 aoclsparse_scsr2bsr()	23
3.3.2.9 aoclsparse_dcsr2bsr()	24

3.3.2.10 aoclsparse_scsr2csc()	24
3.3.2.11 aoclsparse_dcsr2csc()	25
3.3.2.12 aoclsparse_scsr2dense()	26
3.3.2.13 aoclsparse_dcsr2dense()	27
3.4 aoclsparse_functions.h File Reference	28
3.4.1 Detailed Description	30
3.4.2 Function Documentation	30
3.4.2.1 aoclsparse_scsrnmv()	30
3.4.2.2 aoclsparse_dcsrnmv()	32
3.4.2.3 aoclsparse_sellmv()	33
3.4.2.4 aoclsparse_dellmv()	35
3.4.2.5 aoclsparse_sdiamv()	36
3.4.2.6 aoclsparse_ddiamv()	37
3.4.2.7 aoclsparse_sbsrmv()	38
3.4.2.8 aoclsparse_dbsrmv()	40
3.4.2.9 aoclsparse_smv()	41
3.4.2.10 aoclsparse_dmv()	42
3.4.2.11 aoclsparse_scsrsv()	43
3.4.2.12 aoclsparse_dcsrsv()	44
3.4.2.13 aoclsparse_strsv()	45
3.4.2.14 aoclsparse_dtrsv()	47
3.4.2.15 aoclsparse_scsrmm()	48
3.4.2.16 aoclsparse_dcsrmm()	49
3.4.2.17 aoclsparse_dcsr2m()	50
3.4.2.18 aoclsparse_scsr2m()	52
3.4.2.19 aoclsparse_dilu_smoother()	53
3.4.2.20 aoclsparse_silu_smoother()	54
3.5 aoclsparse_solvers.h File Reference	55
3.5.1 Detailed Description	57
3.5.2 Iterative Solver Suite (itsol)	57
3.5.2.1 Introduction	57
3.5.2.2 Forward and Reverse Communication Interfaces	57
3.5.2.3 Recommended Workflow	57
3.5.2.4 Information Array	57
3.5.2.5 Examples	58
3.5.2.6 References	58
3.5.3 Enumeration Type Documentation	58
3.5.3.1 aoclsparse_itsol_rci_job_	58
3.5.4 Function Documentation	59
3.5.4.1 aoclsparse_itsol_handle_prn_options()	59
3.5.4.2 aoclsparse_itsol_option_set()	59
3.5.5 Options	60

3.5.5.1 aoclsparse_itsol_d_init()	61
3.5.5.2 aoclsparse_itsol_s_init()	61
3.5.5.3 aoclsparse_itsol_destroy()	62
3.5.5.4 aoclsparse_itsol_d_rci_input()	62
3.5.5.5 aoclsparse_itsol_s_rci_input()	63
3.5.5.6 aoclsparse_itsol_d_rci_solve()	64
3.5.5.7 aoclsparse_itsol_s_rci_solve()	65
3.5.5.8 aoclsparse_itsol_d_solve()	66
3.5.5.9 aoclsparse_itsol_s_solve()	68
3.6 aoclsparse_types.h File Reference	70
3.6.1 Detailed Description	71
3.6.2 Macro Definition Documentation	71
3.6.2.1 DLL_PUBLIC	71
3.6.3 Typedef Documentation	72
3.6.3.1 aoclsparse_mat_descr	72
3.6.3.2 aoclsparse_csr	72
3.6.3.3 aoclsparse_operation	72
3.6.3.4 aoclsparse_index_base	72
3.6.3.5 aoclsparse_matrix_type	72
3.6.3.6 aoclsparse_matrix_data_type	73
3.6.3.7 aoclsparse_ilu_type	73
3.6.3.8 aoclsparse_matrix_format_type	73
3.6.3.9 aoclsparse_diag_type	73
3.6.3.10 aoclsparse_fill_mode	73
3.6.3.11 aoclsparse_order	73
3.6.3.12 aoclsparse_status	74
3.6.3.13 aoclsparse_request	74
3.6.4 Enumeration Type Documentation	74
3.6.4.1 aoclsparse_operation_	74
3.6.4.2 aoclsparse_index_base_	74
3.6.4.3 aoclsparse_matrix_type_	75
3.6.4.4 aoclsparse_matrix_data_type_	75
3.6.4.5 aoclsparse_ilu_type_	75
3.6.4.6 aoclsparse_matrix_format_type_	77
3.6.4.7 aoclsparse_diag_type_	77
3.6.4.8 aoclsparse_fill_mode_	77
3.6.4.9 aoclsparse_order_	78
3.6.4.10 aoclsparse_status_	78
3.6.4.11 aoclsparse_request_	79
<b>Index</b>	<b>81</b>



## Chapter 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.





## Chapter 2

# File Index

### 2.1 File List

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

<a href="#">aoclsparse_analysis.h</a>	
Aoclsparse_analysis.h provides sparse format analysis subprograms . . . . .	5
<a href="#">aoclsparse_auxiliary.h</a>	
Aoclsparse_auxiliary.h provides auxiliary functions in aoclsparse . . . . .	7
<a href="#">aoclsparse_convert.h</a>	
Aoclsparse_convert.h provides sparse format conversion subprograms . . . . .	16
<a href="#">aoclsparse_functions.h</a>	
Aoclsparse_functions.h provides sparse linear algebra subprograms of level 1, 2 and 3, for AMD CPU hardware . . . . .	28
<a href="#">aoclsparse_solvers.h</a>	
Aoclsparse_solvers.h provides iterative sparse linear system solvers . . . . .	55
<a href="#">aoclsparse_types.h</a>	
Aoclsparse_types.h defines data types used by aoclsparse . . . . .	70



## Chapter 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	<i>mat</i>	sparse matrix in CSR format and sparse format information inside
----	------------	--

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m is invalid.
<i>aoclsparse_status_invalid_pointer</i>	
<i>aoclsparse_status_internal_error</i>	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	<i>mat</i>	sparse matrix in CSR format and sparse format information inside
in	<i>trans</i>	Whether in transposed state or not. Transpose operation is not yet supported.
in	<i>descr</i>	descriptor of the sparse CSR matrix. Currently, only <a href="#">aoclsparse_matrix_type_general</a> and <a href="#">aoclsparse_matrix_type_symmetric</a> is supported.
in	<i>expected_no_of_calls</i>	unused parameter

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m is invalid.
<i>aoclsparse_status_invalid_pointer</i>	
<i>aoclsparse_status_internal_error</i>	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	<i>mat</i>	sparse matrix in CSR format and ILU related information inside
in	<i>trans</i>	Whether in transposed state or not. Transpose operation is not yet supported.
in	<i>descr</i>	descriptor of the sparse CSR matrix. Currently, only <a href="#">aoclsparse_matrix_type_symmetric</a> is supported.
in	<i>expected_no_of_calls</i>	unused parameter

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m is invalid.
<i>aoclsparse_status_invalid_pointer</i>	
<i>aoclsparse_status_internal_error</i>	an internal error occurred.

## 3.2 aoclsparse\_auxiliary.h File Reference

[aoclsparse\\_auxiliary.h](#) provides auxiliary functions in aoclsparse

### Functions

- [DLL\\_PUBLIC](#) 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 auxiliary functions in aoclsparse

### 3.2.2 Function Documentation

#### 3.2.2.1 aoclsparse\_get\_version()

```
DLL_PUBLIC 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	<i>version</i>	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	<i>descr</i>	the pointer to the matrix descriptor.
-----	--------------	---------------------------------------

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> 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	<i>dest</i>	the pointer to the destination matrix descriptor.
in	<i>src</i>	the pointer to the source matrix descriptor.

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>src</i> or <i>dest</i> 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	<i>descr</i>	the matrix descriptor.
----	--------------	------------------------

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> 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	<i>descr</i>	the matrix descriptor.
in	<i>base</i>	<a href="#">aoclsparse_index_base_zero</a> or <a href="#">aoclsparse_index_base_one</a> .

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> pointer is invalid.
<i>aoclsparse_status_invalid_value</i>	<i>base</i> 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	<i>descr</i>	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	<i>descr</i>	the matrix descriptor.
in	<i>type</i>	<a href="#">aoclsparse_matrix_type_general</a> , <a href="#">aoclsparse_matrix_type_symmetric</a> , <a href="#">aoclsparse_matrix_type_hermitian</a> or <a href="#">aoclsparse_matrix_type_triangular</a> .

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> pointer is invalid.
<i>aoclsparse_status_invalid_value</i>	<i>type</i> 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	<i>descr</i>	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	<i>descr</i>	the matrix descriptor.
in	<i>fill_mode</i>	<a href="#">aoclsparse_fill_mode_lower</a> or <a href="#">aoclsparse_fill_mode_upper</a> .

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> pointer is invalid.
<i>aoclsparse_status_invalid_value</i>	<i>fill_mode</i> 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	<i>descr</i>	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	<i>descr</i>	the matrix descriptor.
in	<i>diag_type</i>	<a href="#">aoclsparse_diag_type_unit</a> or <a href="#">aoclsparse_diag_type_non_unit</a> .

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> pointer is invalid.
<i>aoclsparse_status_invalid_value</i>	<i>diag_type</i> 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	<i>descr</i>	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	<i>mat</i>	the pointer to the CSR sparse matrix.
in	<i>base</i>	<a href="#">aoclsparse_index_base_zero</a> or <a href="#">aoclsparse_index_base_one</a> .
in	<i>M</i>	number of rows of the sparse CSR matrix.
in	<i>N</i>	number of columns of the sparse CSR matrix.
in	<i>csr_nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ptr</i>	array of nnz elements containing the column indices of the sparse CSR matrix.
in	<i>csr_val</i>	array of nnz elements of the sparse CSR matrix.

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	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	<i>mat</i>	the pointer to the CSR sparse matrix.
in	<i>base</i>	<a href="#">aoclsparse_index_base_zero</a> or <a href="#">aoclsparse_index_base_one</a> .
in	<i>M</i>	number of rows of the sparse CSR matrix.
in	<i>N</i>	number of columns of the sparse CSR matrix.
in	<i>csr_nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ptr</i>	array of nnz elements containing the column indices of the sparse CSR matrix.
in	<i>csr_val</i>	array of nnz elements of the sparse CSR matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	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	<i>csr</i>	the pointer to the CSR sparse matrix.
out	<i>base</i>	<a href="#">aoclsparse_index_base_zero</a> or <a href="#">aoclsparse_index_base_one</a> .
out	<i>M</i>	number of rows of the sparse CSR matrix.
out	<i>N</i>	number of columns of the sparse CSR matrix.
out	<i>csr_nnz</i>	number of non-zero entries of the sparse CSR matrix.
out	<i>csr_row_ptr</i>	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
out	<i>csr_col_ind</i>	array of nnz elements containing the column indices of the sparse CSR matrix.
out	<i>csr_val</i>	array of nnz elements of the sparse CSR matrix.

## Return values

<code>aoclsparse_status_success</code>	the operation completed successfully.
<code>aoclsparse_status_invalid_pointer</code>	<code>csr</code> 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

<code>in</code>	<code>mat</code>	the pointer to the sparse matrix.
-----------------	------------------	-----------------------------------

## Return values

<code>aoclsparse_status_success</code>	the operation completed successfully.
<code>aoclsparse_status_invalid_pointer</code>	<code>matrix</code> 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	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <i>m</i> +1 elements that point to the start of every row of the sparse CSR matrix.
out	<i>ell_width</i>	pointer to the number of non-zero elements per row in ELL storage format.

##### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>csr_row_ptr</i> , or <i>ell_width</i> pointer is invalid.
<i>aoclsparse_status_internal_error</i>	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	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>csr_val</i>	array containing the values of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of $m+1$ elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.
in	<i>ell_width</i>	number of non-zero elements per row in ELL storage format.
out	<i>ell_val</i>	array of $m$ times <code>ell_width</code> elements of the sparse ELL matrix.
out	<i>ell_col_ind</i>	array of $m$ times <code>ell_width</code> elements containing the column indices of the sparse ELL matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	$m$ or <code>ell_width</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>csr_val</code> , <code>csr_row_ptr</code> , <code>csr_col_ind</code> , <code>ell_val</code> or <code>ell_col_ind</code> 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  $\text{nnz}_{\text{ELL}} = m \cdot \text{ell\_width}$ . The number of ELL non-zero elements per row is obtained by `aoclsparse_csr2ell_width()`.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>csr_val</i>	array containing the values of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of $m+1$ elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.

## Parameters

in	<i>ell_width</i>	number of non-zero elements per row in ELL storage format.
out	<i>ell_val</i>	array of <i>m</i> times <i>ell_width</i> elements of the sparse ELL matrix.
out	<i>ell_col_ind</i>	array of <i>m</i> times <i>ell_width</i> elements containing the column indices of the sparse ELL matrix.

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> or <i>ell_width</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>ell_val</i> or <i>ell_col_ind</i> 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	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of cols of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <i>m</i> +1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.
out	<i>dia_num_diag</i>	pointer to the number of diagonals with non-zeroes in DIA storage format.

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>csr_row_ptr</i> , or <i>ell_width</i> pointer is invalid.
<i>aoclsparse_status_internal_error</i>	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_scsr2dia` 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_scsr2dia_ndiag()`.

#### Parameters

in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of cols of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <i>m</i> +1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.
in	<i>csr_val</i>	array containing the values of the sparse CSR matrix.
in	<i>dia_num_diag</i>	number of diagonals in ELL storage format.
out	<i>dia_offset</i>	array of <i>dia_num_diag</i> elements containing the diagonal offsets from main diagonal.
out	<i>dia_val</i>	array of <i>m</i> times <i>dia_num_diag</i> elements of the sparse DIA matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> or <i>ell_width</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>ell_val</i> or <i>ell_col_ind</i> 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	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of cols of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <i>m</i> +1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array containing the column indices of the sparse CSR matrix.
in	<i>csr_val</i>	array containing the values of the sparse CSR matrix.
in	<i>dia_num_diag</i>	number of diagonals in ELL storage format.
out	<i>dia_offset</i>	array of <i>dia_num_diag</i> elements containing the diagonal offsets from main diagonal.
out	<i>dia_val</i>	array of <i>m</i> times <i>dia_num_diag</i> elements of the sparse DIA matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> or <i>ell_width</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>ell_val</i> or <i>ell_col_ind</i> 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	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	integer array containing <i>m</i> +1 elements that point to the start of each row of the CSR matrix
in	<i>csr_col_ind</i>	integer array of the column indices for each non-zero element in the CSR matrix
in	<i>block_dim</i>	the block dimension of the BSR matrix. Between 1 and min( <i>m</i> , <i>n</i> )
out	<i>bsr_row_ptr</i>	integer array containing <i>m</i> +1 elements that point to the start of each block row of the BSR matrix
out	<i>bsr_nnz</i>	total number of nonzero elements in device or host memory.

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m or n or block_dim is invalid.
<i>aoclsparse_status_invalid_pointer</i>	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\_scsr2bsr 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	<i>m</i>	number of rows in the sparse CSR matrix.
in	<i>n</i>	number of columns in the sparse CSR matrix.
in	<i>csr_val</i>	array of nnz elements containing the values of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of m+1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of nnz elements containing the column indices of the sparse CSR matrix.
in	<i>block_dim</i>	size of the blocks in the sparse BSR matrix.
out	<i>bsr_val</i>	array of nnzb*block_dim*block_dim containing the values of the sparse BSR matrix.
out	<i>bsr_row_ptr</i>	array of mb+1 elements that point to the start of every block row of the sparse BSR matrix.
out	<i>bsr_col_ind</i>	array of nnzb elements containing the block column indices of the sparse BSR matrix.

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	m or n or block_dim is invalid.
<i>aoclsparse_status_invalid_pointer</i>	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	<i>m</i>	number of rows in the sparse CSR matrix.
in	<i>n</i>	number of columns in the sparse CSR matrix.
in	<i>csr_val</i>	array of <code>nnz</code> elements containing the values of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <code>m+1</code> elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of <code>nnz</code> elements containing the column indices of the sparse CSR matrix.
in	<i>block_dim</i>	size of the blocks in the sparse BSR matrix.
out	<i>bsr_val</i>	array of <code>nnzb*block_dim*block_dim</code> containing the values of the sparse BSR matrix.
out	<i>bsr_row_ptr</i>	array of <code>mb+1</code> elements that point to the start of every block row of the sparse BSR matrix.
out	<i>bsr_col_ind</i>	array of <code>nnzb</code> elements containing the block column indices of the sparse BSR matrix.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<code>m</code> or <code>n</code> or <code>block_dim</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>bsr_val</code> , <code>bsr_row_ptr</code> , <code>bsr_col_ind</code> , <code>csr_val</code> , <code>csr_row_ptr</code> or <code>csr_col_ind</code> 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	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_val</i>	array of <i>nnz</i> elements of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <i>m</i> +1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of <i>nnz</i> elements containing the column indices of the sparse CSR matrix.
out	<i>csc_val</i>	array of <i>nnz</i> elements of the sparse CSC matrix.
out	<i>csc_row_ind</i>	array of <i>nnz</i> elements containing the row indices of the sparse CSC matrix.
out	<i>csc_col_ptr</i>	array of <i>n</i> +1 elements that point to the start of every column of the sparse CSC matrix. <code>aoclsparse_csr2csc_buffer_size()</code> .

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> , <i>n</i> or <i>nnz</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>csc_val</i> , <i>csc_row_ind</i> , <i>csc_col_ptr</i> 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	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_val</i>	array of <i>nnz</i> elements of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <i>m</i> +1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of <i>nnz</i> elements containing the column indices of the sparse CSR matrix.
out	<i>csc_val</i>	array of <i>nnz</i> elements of the sparse CSC matrix.
out	<i>csc_row_ind</i>	array of <i>nnz</i> elements containing the row indices of the sparse CSC matrix.
out	<i>csc_col_ptr</i>	array of <i>n</i> +1 elements that point to the start of every column of the sparse CSC matrix. <code>aoclsparse_csr2csc_buffer_size()</code> .

**Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> , <i>n</i> or <i>nnz</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>csc_val</i> , <i>csc_row_ind</i> , <i>csc_col_ptr</i> 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	<i>m</i>	number of rows of the dense matrix A.
in	<i>n</i>	number of columns of the dense matrix A.
in	<i>descr</i>	the descriptor of the dense matrix A, the supported matrix type is <a href="#">aoclsparse_matrix_type_general</a> and also any valid value of the <a href="#">aoclsparse_index_base</a> .
in	<i>csr_val</i>	array of <i>nnz</i> ( = <i>csr_row_ptr</i> [ <i>m</i> ] - <i>csr_row_ptr</i> [0] ) nonzero elements of matrix A.
in	<i>csr_row_ptr</i>	integer array of <i>m</i> +1 elements that contains the start of every row and the end of the last row plus one.



## Parameters

in	<i>csr_col_ind</i>	integer array of nnz ( = <i>csr_row_ptr</i> [ <i>m</i> ] - <i>csr_row_ptr</i> [0] ) column indices of the non-zero elements of matrix A.
out	<i>A</i>	array of dimensions ( <i>ld</i> , <i>n</i> )
out	<i>ld</i>	leading dimension of dense array A.
in	<i>order</i>	memory layout of a dense matrix A.

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> or <i>n</i> or <i>ld</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	A or <i>csr_val</i> <i>csr_row_ptr</i> or <i>csr_col_ind</i> 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	<i>m</i>	number of rows of the dense matrix A.
in	<i>n</i>	number of columns of the dense matrix A.
in	<i>descr</i>	the descriptor of the dense matrix A, the supported matrix type is <a href="#">aoclsparse_matrix_type_general</a> and also any valid value of the <a href="#">aoclsparse_index_base</a> .
in	<i>csr_val</i>	array of nnz ( = <i>csr_row_ptr</i> [ <i>m</i> ] - <i>csr_row_ptr</i> [0] ) nonzero elements of matrix A.
in	<i>csr_row_ptr</i>	integer array of <i>m</i> +1 elements that contains the start of every row and the end of the last row plus one.
in	<i>csr_col_ind</i>	integer array of nnz ( = <i>csr_row_ptr</i> [ <i>m</i> ] - <i>csr_row_ptr</i> [0] ) column indices of the non-zero elements of matrix A.
out	<i>A</i>	array of dimensions ( <i>ld</i> , <i>n</i> )
out	<i>ld</i>	leading dimension of dense array A.
in	<i>order</i>	memory layout of a dense matrix A.

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
----------------------------------	---------------------------------------

## Return values

<code>aoclsparse_status_invalid_size</code>	<code>m</code> or <code>n</code> or <code>ld</code> is invalid.
<code>aoclsparse_status_invalid_pointer</code>	A or <code>csr_val</code> <code>csr_row_ptr</code> or <code>csr_col_ind</code> 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_dcsmv` (`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_scsrv` (`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_dcsrv` (`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_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\_scsrmmv()

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

#### Return values

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

### 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\_dcscrmv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcscrmv (
    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_cscrmv` multiplies the scalar  $\alpha$  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  $\beta$ , 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	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse CSR matrix.
in	<i>n</i>	number of columns of the sparse CSR matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse CSR matrix.
in	<i>csr_val</i>	array of <i>nnz</i> elements of the sparse CSR matrix.
in	<i>csr_col_ind</i>	array of <i>nnz</i> elements containing the column indices of the sparse CSR matrix.
in	<i>csr_row_ptr</i>	array of <i>m</i> +1 elements that point to the start of every row of the sparse CSR matrix.
in	<i>descr</i>	descriptor of the sparse CSR matrix. Currently, only <a href="#">aoclsparse_matrix_type_general</a> and <a href="#">aoclsparse_matrix_type_symmetric</a> is supported.
in	<i>x</i>	array of <i>n</i> elements ( $op(A) = A$ ) or <i>m</i> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in, out	<i>y</i>	array of <i>m</i> elements ( $op(A) = A$ ) or <i>n</i> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	<i>m</i> , <i>n</i> or <i>nnz</i> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> , <i>alpha</i> , <i>csr_val</i> , <i>csr_row_ptr</i> , <i>csr_col_ind</i> , <i>x</i> , <i>beta</i> or <i>y</i> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<i>trans</i> is not <a href="#">aoclsparse_operation_none</a> or <a href="#">aoclsparse_matrix_type</a> is not <a href="#">aoclsparse_matrix_type_general</a> , or <a href="#">aoclsparse_matrix_type</a> is not <a href="#">aoclsparse_matrix_type_symmetric</a> .

## Example

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

```
// Compute y = Ax
aoclsparse_scsr_mv(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  $\alpha$  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  $\beta$ , 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	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse ELL matrix.
in	<i>n</i>	number of columns of the sparse ELL matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse ELL matrix.
in	<i>descr</i>	descriptor of the sparse ELL matrix. Currently, only <a href="#">aoclsparse_matrix_type_general</a> is supported.
in	<i>ell_val</i>	array that contains the elements of the sparse ELL matrix. Padded elements should be zero.
in	<i>ell_col_ind</i>	array that contains the column indices of the sparse ELL matrix. Padded column indices should be -1.
in	<i>ell_width</i>	number of non-zero elements per row of the sparse ELL matrix.
in	<i>x</i>	array of $n$ elements ( $op(A) = A$ ) or $m$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in, out	<i>y</i>	array of $m$ elements ( $op(A) = A$ ) or $n$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).



## Return values

<code>aoclsparse_status_success</code>	the operation completed successfully.
<code>aoclsparse_status_invalid_size</code>	<code>m</code> , <code>n</code> or <code>ell_width</code> is invalid.
<code>aoclsparse_status_invalid_pointer</code>	<code>descr</code> , <code>alpha</code> , <code>ell_val</code> , <code>ell_col_ind</code> , <code>x</code> , <code>beta</code> or <code>y</code> pointer is invalid.
<code>aoclsparse_status_not_implemented</code>	<code>trans</code> != <a href="#">aoclsparse_operation_none</a> or <code>aoclsparse_matrix_type</code> != <a href="#">aoclsparse_matrix_type_general</a> .

## 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  $\alpha$  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  $\beta$ , 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

<code>in</code>	<code>trans</code>	matrix operation type.
-----------------	--------------------	------------------------

## Parameters

in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse ELL matrix.
in	<i>n</i>	number of columns of the sparse ELL matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse ELL matrix.
in	<i>descr</i>	descriptor of the sparse ELL matrix. Currently, only <a href="#">aoclsparse_matrix_type_general</a> is supported.
in	<i>ell_val</i>	array that contains the elements of the sparse ELL matrix. Padded elements should be zero.
in	<i>ell_col_ind</i>	array that contains the column indices of the sparse ELL matrix. Padded column indices should be -1.
in	<i>ell_width</i>	number of non-zero elements per row of the sparse ELL matrix.
in	<i>x</i>	array of $n$ elements ( $op(A) = A$ ) or $m$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in, out	<i>y</i>	array of $m$ elements ( $op(A) = A$ ) or $n$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m$ , $n$ or $ell\_width$ is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> , <i>alpha</i> , <i>ell_val</i> , <i>ell_col_ind</i> , <i>x</i> , <i>beta</i> or <i>y</i> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	$trans \neq$ <a href="#">aoclsparse_operation_none</a> or <a href="#">aoclsparse_matrix_type</a> $\neq$ <a href="#">aoclsparse_matrix_type_general</a> .

## 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  $\alpha$  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  $\beta$ , 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	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse DIA matrix.
in	<i>n</i>	number of columns of the sparse DIA matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse DIA matrix.
in	<i>descr</i>	descriptor of the sparse DIA matrix. Currently, only <a href="#">aoclsparse_matrix_type_general</a> is supported.
in	<i>dia_val</i>	array that contains the elements of the sparse DIA matrix. Padded elements should be zero.
in	<i>dia_offset</i>	array that contains the offsets of each diagonal of the sparse DIA matrix.
in	<i>dia_num_diag</i>	number of diagonals in the sparse DIA matrix.
in	<i>x</i>	array of $n$ elements ( $op(A) = A$ ) or $m$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in, out	<i>y</i>	array of $m$ elements ( $op(A) = A$ ) or $n$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m, n$ or <code>ell_width</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> , <code>alpha</code> , <code>ell_val</code> , <code>ell_col_ind</code> , <code>x</code> , <code>beta</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<code>trans</code> != <a href="#">aoclsparse_operation_none</a> or <code>aoclsparse_matrix_type</code> != <a href="#">aoclsparse_matrix_type_general</a> .

## 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  $\alpha$  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  $\beta$ , 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	<i>trans</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>m</i>	number of rows of the sparse DIA matrix.
in	<i>n</i>	number of columns of the sparse DIA matrix.
in	<i>nnz</i>	number of non-zero entries of the sparse DIA matrix.
in	<i>descr</i>	descriptor of the sparse DIA matrix. Currently, only <a href="#">aoclsparse_matrix_type_general</a> is supported.
in	<i>dia_val</i>	array that contains the elements of the sparse DIA matrix. Padded elements should be zero.
in	<i>dia_offset</i>	array that contains the offsets of each diagonal of the sparse DIA matrix.
in	<i>dia_num_diag</i>	number of diagonals in the sparse DIA matrix.
in	<i>x</i>	array of $n$ elements ( $op(A) = A$ ) or $m$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in, out	<i>y</i>	array of $m$ elements ( $op(A) = A$ ) or $n$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m$ , $n$ or <code>ell_width</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> , <code>alpha</code> , <code>ell_val</code> , <code>ell_col_ind</code> , <code>x</code> , <code>beta</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<code>trans</code> != <a href="#">aoclsparse_operation_none</a> or <code>aoclsparse_matrix_type</code> != <a href="#">aoclsparse_matrix_type_general</a> .

#### 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  $\alpha$  with a sparse  $(mb \cdot bsr\_dim) \times (nb \cdot bsr\_dim)$  matrix, defined in BSR storage format, and the dense vector  $x$  and adds the result to the dense vector  $y$  that is multiplied by the scalar  $\beta$ , 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	<i>trans</i>	matrix operation type.
in	<i>mb</i>	number of block rows of the sparse BSR matrix.
in	<i>nb</i>	number of block columns of the sparse BSR matrix.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>descr</i>	descriptor of the sparse BSR matrix. Currently, only <a href="#">aoclsparse_matrix_type_general</a> is supported.
in	<i>bsr_val</i>	array of <code>nnzb</code> blocks of the sparse BSR matrix.
in	<i>bsr_row_ptr</i>	array of <code>mb+1</code> elements that point to the start of every block row of the sparse BSR matrix.
in	<i>bsr_col_ind</i>	array of <code>nnz</code> containing the block column indices of the sparse BSR matrix.
in	<i>bsr_dim</i>	block dimension of the sparse BSR matrix.
in	<i>x</i>	array of <code>mb*bsr_dim</code> elements ( $op(A) = A$ ) or <code>mb*bsr_dim</code> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in, out	<i>y</i>	array of <code>mb*bsr_dim</code> elements ( $op(A) = A$ ) or <code>nb*bsr_dim</code> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_handle</i>	the library context was not initialized.
<i>aoclsparse_status_invalid_size</i>	<code>mb</code> , <code>nb</code> , <code>nnzb</code> or <code>bsr_dim</code> is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<code>descr</code> , <code>alpha</code> , <code>bsr_val</code> , <code>bsr_row_ptr</code> , <code>bsr_col_ind</code> , <code>x</code> , <code>beta</code> or <code>y</code> pointer is invalid.
<i>aoclsparse_status_arch_mismatch</i>	the device is not supported.

## Return values

<code>aoclsparse_status_not_implemented</code>	<code>trans</code> != <code>aoclsparse_operation_none</code> or <code>aoclsparse_matrix_type</code> != <code>aoclsparse_matrix_type_general</code> .
--	--

## 3.4.2.8 aoclsparse\_dbstrmv()

```

DLL_PUBLIC aoclsparse_status aoclsparse_dbstrmv (
    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_bstrmv` multiplies the scalar  $\alpha$  with a sparse  $(mb \cdot bsr\_dim) \times (nb \cdot bsr\_dim)$  matrix, defined in BSR storage format, and the dense vector  $x$  and adds the result to the dense vector  $y$  that is multiplied by the scalar  $\beta$ , 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	<i>trans</i>	matrix operation type.
in	<i>mb</i>	number of block rows of the sparse BSR matrix.
in	<i>nb</i>	number of block columns of the sparse BSR matrix.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>descr</i>	descriptor of the sparse BSR matrix. Currently, only <code>aoclsparse_matrix_type_general</code> is supported.
in	<i>bsr_val</i>	array of nnzb blocks of the sparse BSR matrix.
in	<i>bsr_row_ptr</i>	array of mb+1 elements that point to the start of every block row of the sparse BSR matrix.
in	<i>bsr_col_ind</i>	array of nnz containing the block column indices of the sparse BSR matrix.
in	<i>bsr_dim</i>	block dimension of the sparse BSR matrix.
in	<i>x</i>	array of nb*bsr_dim elements ( $op(A) = A$ ) or mb*bsr_dim elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ . <span style="float: right;">Generated by Doxygen</span>
in, out	<i>y</i>	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

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

## 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  $\alpha$  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  $\beta$ , 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	<code>op</code>	matrix operation type.
in	<code>alpha</code>	scalar $\alpha$ .
in	<code>A</code>	the sparse matrix structure that is created using <a href="#">aoclsparse_create_dcscr</a> .
in	<code>descr</code>	descriptor of the sparse CSR matrix. Currently, only <a href="#">aoclsparse_matrix_type_general</a> and <a href="#">aoclsparse_matrix_type_symmetric</a> is supported.
in	<code>x</code>	array of <code>n</code> elements ( $op(A) = A$ ) or <code>m</code> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<code>beta</code>	scalar $\beta$ .
in, out	<code>y</code>	array of <code>m</code> elements ( $op(A) = A$ ) or <code>n</code> elements ( $op(A) = A^T$ or $op(A) = A^H$ ).

## Return values

<code>aoclsparse_status_success</code>	the operation completed successfully.
<code>aoclsparse_status_invalid_size</code>	<code>m</code> , <code>n</code> or <code>nnz</code> is invalid.
<code>aoclsparse_status_invalid_pointer</code>	<code>descr</code> , <code>alpha</code> , <code>internal</code> structures related to the sparse matrix <code>A</code> , <code>x</code> , <code>beta</code> or <code>y</code> has an invalid pointer.
<code>aoclsparse_status_not_implemented</code>	<code>trans</code> != <code>aoclsparse_operation_none</code> or <code>aoclsparse_matrix_type</code> != <code>aoclsparse_matrix_type_general</code> . <code>aoclsparse_matrix_type</code> != <code>aoclsparse_matrix_type_symmetric</code> .

## 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  $\alpha$  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  $\beta$ , 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	<i>op</i>	matrix operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>A</i>	the sparse matrix structure that is created using <a href="#">aoclsparse_create_dcsr</a> .
in	<i>descr</i>	descriptor of the sparse CSR matrix. Currently, only <a href="#">aoclsparse_matrix_type_general</a> and <a href="#">aoclsparse_matrix_type_symmetric</a> is supported.
in	<i>x</i>	array of $n$ elements ( $op(A) = A$ ) or $m$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).
in	<i>beta</i>	scalar $\beta$ .
in, out	<i>y</i>	array of $m$ elements ( $op(A) = A$ ) or $n$ elements ( $op(A) = A^T$ or $op(A) = A^H$ ).

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m$ , $n$ or $nnz$ is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> , <i>alpha</i> , <i>internal</i> structures related to the sparse matrix <i>A</i> , <i>x</i> , <i>beta</i> or <i>y</i> has an invalid pointer.
<i>aoclsparse_status_not_implemented</i>	<i>trans</i> != <a href="#">aoclsparse_operation_none</a> or <i>aoclsparse_matrix_type</i> != <a href="#">aoclsparse_matrix_type_general</a> . <i>aoclsparse_matrix_type</i> != <a href="#">aoclsparse_matrix_type_symmetric</a> .

## 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  $\alpha$ , such that

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

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.

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

#### Return values

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

#### 3.4.2.12 aoclsparse\_dcsrcsv()

```
DLL_PUBLIC aoclsparse_status aoclsparse_dcsrcsv (
    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_?strsv` 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  $\alpha$ , such that

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

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.

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

#### Return values

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

#### 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  $\alpha$ . 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 as 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 optimization for the matrix will be done by the solver on entry.

#### Parameters

in	<i>trans</i>	matrix operation type, either <code>aoclsparse_operation_none</code> or <code>aoclsparse_operation_transpose</code> .
in	<i>alpha</i>	scalar $\alpha$ , used to premultiply right-hand side vector $b$ .
in, out	<i>A</i>	matrix data. $A$ is modified only if solver requires to optimize matrix data.
in	<i>descr</i>	matrix descriptor.
in	<i>b</i>	array of $m$ elements, storing the right-hand side.
out	<i>x</i>	array of $m$ elements, storing the solution if solver returns <code>aoclsparse_status_success</code> .

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully and $x$ contains the solution to the linear system of equations.
<i>aoclsparse_status_invalid_size</i>	matrix $A$ or $op(A)$ is invalid.

## Return values

<code>aoclsparse_status_invalid_pointer</code>	One or more of <code>A</code> , <code>descr</code> , <code>x</code> , <code>b</code> are invalid pointers.
<code>aoclsparse_status_internal_error</code>	an internal error occurred.
<code>aoclsparse_status_not_implemented</code>	the requested operation is not yet implemented.
<i>other</i>	possible failure values from a call to <a href="#">aoclsparse_optimize</a> .

## 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  $\alpha$ . 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 as 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 optimization for the matrix will be done by the solver on entry.

## Parameters

in	<i>trans</i>	matrix operation type, either <a href="#">aoclsparse_operation_none</a> or <a href="#">aoclsparse_operation_transpose</a> .
in	<i>alpha</i>	scalar $\alpha$ , used to premultiply right-hand side vector $b$ .
in, out	<i>A</i>	matrix data. $A$ is modified only if solver requires to optimize matrix data.
in	<i>descr</i>	matrix descriptor.
in	<i>b</i>	array of $m$ elements, storing the right-hand side.
out	<i>x</i>	array of $m$ elements, storing the solution if solver returns <a href="#">aoclsparse_status_success</a> .

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully and $x$ contains the solution to the linear system of equations.
<i>aoclsparse_status_invalid_size</i>	matrix $A$ or $op(A)$ is invalid.
<i>aoclsparse_status_invalid_pointer</i>	One or more of $A$ , $descr$ , $x$ , $b$ are invalid pointers.
<i>aoclsparse_status_internal_error</i>	an internal error occurred.
<i>aoclsparse_status_not_implemented</i>	the requested operation is not yet implemented.
<i>other</i>	possible failure values from a call to <a href="#">aoclsparse_optimize</a> .

## 3.4.2.15 aoclsparse\_scsrmm()

```
DL_L_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_scsrmm` multiplies the scalar  $\alpha$  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  $\beta$ , such that

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

with

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

```
for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];
        for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
        {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}
```

## Parameters

in	<i>trans_A</i>	matrix $A$ operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>csr</i>	sparse CSR matrix $A$ structure.
in	<i>descr</i>	descriptor of the sparse CSR matrix $A$ . Currently, only <a href="#">aoclsparse_matrix_type_general</a> is supported.
in	<i>order</i>	aoclsparse_order_row/aoclsparse_order_column for dense matrix
in	<i>B</i>	array of dimension $ldb \times n$ or $ldb \times k$ .
in	<i>n</i>	number of columns of the dense matrix $B$ and $C$ .
in	<i>ldb</i>	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	<i>beta</i>	scalar $\beta$ .
in, out	<i>C</i>	array of dimension $ldc \times n$ .
in	<i>ldc</i>	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

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m, n, k, nnz, ldb$ or $ldc$ is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> , <i>alpha</i> , <i>csr</i> , <i>B</i> , <i>beta</i> or <i>C</i> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<a href="#">aoclsparse_matrix_type</a> is not <a href="#">aoclsparse_matrix_type_general</a> .

## 3.4.2.16 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  $\alpha$  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  $\beta$ , such that

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

with

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

```

for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];
        for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
        {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}

```

#### Parameters

in	<i>trans_A</i>	matrix $A$ operation type.
in	<i>alpha</i>	scalar $\alpha$ .
in	<i>csr</i>	sparse CSR matrix $A$ structure.
in	<i>descr</i>	descriptor of the sparse CSR matrix $A$ . Currently, only <a href="#">aoclsparse_matrix_type_general</a> is supported.
in	<i>order</i>	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	<i>beta</i>	scalar $\beta$ .
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

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	$m, n, k, nnz, ldb$ or $ldc$ is invalid.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> , <i>alpha</i> , <i>csr</i> , <i>B</i> , <i>beta</i> or <i>C</i> pointer is invalid.
<i>aoclsparse_status_not_implemented</i>	<a href="#">aoclsparse_matrix_type</a> is not <a href="#">aoclsparse_matrix_type_general</a> .

### 3.4.2.17 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 multiplication using CSR storage format for single and double precision datatypes.

`aoclsparse_dcsr2m` 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

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

and

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

#### Parameters

in	<i>trans↔ _A</i>	matrix <i>A</i> operation type.
in	<i>descrA</i>	descriptor of the sparse CSR matrix <i>A</i> . Currently, only <a href="#">aoclsparse_matrix_type_general</a> is supported.
in	<i>csrA</i>	sparse CSR matrix <i>A</i> structure.
in	<i>trans↔ _B</i>	matrix <i>B</i> operation type.
in	<i>descrB</i>	descriptor of the sparse CSR matrix <i>B</i> . Currently, only <a href="#">aoclsparse_matrix_type_general</a> is supported.
in	<i>csrB</i>	sparse CSR matrix <i>B</i> structure.
in	<i>request</i>	Specifies full computation or two-stage algorithm <a href="#">aoclsparse_stage_nnz_count</a> , 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. <a href="#">aoclsparse_stage_finalize</a> . 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. <a href="#">aoclsparse_stage_full_computation</a> . Perform the entire computation in a single step.
out	<i>*csrC</i>	Pointer to sparse CSR matrix <i>C</i> structure.

#### Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	input parameters contain an invalid value.
<i>aoclsparse_status_invalid_pointer</i>	descrA, csr, descrB, csrB, csrC is invalid.
<i>aoclsparse_status_not_implemented</i>	<a href="#">aoclsparse_matrix_type</a> is not <a href="#">aoclsparse_matrix_type_general</a> .

#### 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);

```

### 3.4.2.18 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

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

and

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

#### Parameters

in	<i>trans_↔ _A</i>	matrix $A$ operation type.
in	<i>descrA</i>	descriptor of the sparse CSR matrix $A$ . Currently, only <code>aoclsparse_matrix_type_general</code> is supported.
in	<i>csrA</i>	sparse CSR matrix $A$ structure.
in	<i>trans_↔ _B</i>	matrix $B$ operation type.
in	<i>descrB</i>	descriptor of the sparse CSR matrix $B$ . Currently, only <code>aoclsparse_matrix_type_general</code> is supported.
in	<i>csrB</i>	sparse CSR matrix $B$ structure.
in	<i>request</i>	Specifies full computation or two-stage algorithm <code>aoclsparse_stage_nnz_count</code> , 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. <code>aoclsparse_stage_finalize</code> . Finalize computation of remaining output arrays ( column indices and values of output matrix entries) . Has to be called only after <code>aoclsparse_dcsr2m</code> call with <code>aoclsparse_stage_nnz_count</code> parameter. <code>aoclsparse_stage_full_computation</code> Perform the entire computation in a single step.
out	<i>*csrC</i>	Pointer to sparse CSR matrix $C$ structure.

## Return values

<code>aoclsparse_status_success</code>	the operation completed successfully.
<code>aoclsparse_status_invalid_size</code>	input parameters contain an invalid value.
<code>aoclsparse_status_invalid_pointer</code>	<code>descrA</code> , <code>csr</code> , <code>descrB</code> , <code>csrB</code> , <code>csrC</code> is invalid.
<code>aoclsparse_status_not_implemented</code>	<code>aoclsparse_matrix_type</code> is not <code>aoclsparse_matrix_type_general</code> .

## 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);

```

## 3.4.2.19 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	<code>op</code>	matrix <code>A</code> operation type. Transpose not yet supported.
----	-----------------	--

## Parameters

in	<i>A</i>	sparse matrix handle. Currently ILU functionality is supported only for CSR matrix format.
in	<i>descr</i>	descriptor of the sparse matrix handle <i>A</i> . Currently, only <a href="#">aoclsparse_matrix_type_symmetric</a> is supported.
out	<i>precond_csr_val</i>	output pointer that contains L and U factors after ILU operation. The original value buffer of matrix <i>A</i> is not overwritten with the factors.
in	<i>approx_inv_diag</i>	It is unused as of now.
out	<i>x</i>	array of <i>n</i> element vector found using the known values of CSR matrix <i>A</i> and resultant vector product <i>b</i> in $Ax = b$ . Every call to the API gives an iterative update of <i>x</i> , which is used to find norm during LU solve phase. Norm and Relative Error % decides the convergence of <i>x</i> with respect to <i>x_ref</i>
in	<i>b</i>	array of <i>m</i> elements which is the result of <i>A</i> and <i>x</i> in $Ax = b$ . <i>b</i> is calculated using a known reference <i>x</i> vector, which is then used to find the norm for iterative <i>x</i> during LU solve phase. Norm and Relative Error percentage decides the convergence

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	input parameters contain an invalid value.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> , <i>A</i> is invalid.
<i>aoclsparse_status_not_implemented</i>	<a href="#">aoclsparse_matrix_type</a> is not <a href="#">aoclsparse_matrix_type_symmetric</a> .

Refer to ILU Example from tests/include.

## 3.4.2.20 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	<i>op</i>	matrix <i>A</i> operation type. Transpose not yet supported.
in	<i>A</i>	sparse matrix handle. Currently ILU functionality is supported only for CSR matrix format.

## Parameters

in	<i>descr</i>	descriptor of the sparse matrix handle <i>A</i> . Currently, only <a href="#">aoclsparse_matrix_type_symmetric</a> is supported.
out	<i>precond_csr_val</i>	output pointer that contains L and U factors after ILU operation. The original value buffer of matrix <i>A</i> is not overwritten with the factors.
in	<i>approx_inv_diag</i>	It is unused as of now.
out	<i>x</i>	array of <i>n</i> element vector found using the known values of CSR matrix <i>A</i> and resultant vector product <i>b</i> in $Ax = b$ . Every call to the API gives an iterative update of <i>x</i> , which is used to find norm during LU solve phase. Norm and Relative Error % decides the convergence of <i>x</i> with respect to <i>x_ref</i>
in	<i>b</i>	array of <i>m</i> elements which is the result of <i>A</i> and <i>x</i> in $Ax = b$ . <i>b</i> is calculated using a known reference <i>x</i> vector, which is then used to find the norm for iterative <i>x</i> during LU solve phase. Norm and Relative Error percentage decides the convergence

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_size</i>	input parameters contain an invalid value.
<i>aoclsparse_status_invalid_pointer</i>	<i>descr</i> , <i>A</i> is invalid.
<i>aoclsparse_status_not_implemented</i>	<a href="#">aoclsparse_matrix_type</a> is not <a href="#">aoclsparse_matrix_type_symmetric</a> .

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](#) precondition([aoclsparse\\_int](#) flag, const double \*u, double \*v, void \*udata), [aoclsparse\\_int](#) monit(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](#) precondition([aoclsparse\\_int](#) flag, const float \*u, float \*v, void \*udata), [aoclsparse\\_int](#) monit(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 G↔MRES. 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 has 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 <a href="#">Conjugate Gradient method</a> .
	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 <a href="#">Conjugate Gradient method</a> .
	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

<code>aoclsparse_rci_interrupt</code>	if set by the user, signals the solver to terminate. This is never set by the solver. Terminate.
---------------------------------------	--



## Enumerator

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 <a href="#">Options</a> ). Terminate, vector $x$ contains the solution.
aoclsparse_rci_start	initial value of the <a href="#">ircomm</a> 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 <a href="#">aoclsparse_rci_stop</a> ), 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	<i>handle</i>	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
<b>cg iteration limit</b>	integer	$i = 500$
Set CG iteration limit		
Valid values: $1 \leq i$ .		
<b>gmres iteration limit</b>	integer	$i = 150$
Set GMRES iteration limit		
Valid values: $1 \leq i$ .		
<b>gmres restart iterations</b>	integer	$i = 20$
Set GMRES restart iterations		
Valid values: $1 \leq i$ .		
<b>cg rel tolerance</b>	real	$r = 1.08735e - 06$
Set relative convergence tolerance for cg method		
Valid values: $0 \leq r$ .		
<b>cg abs tolerance</b>	real	$r = 0$
Set absolute convergence tolerance for cg method		
Valid values: $0 \leq r$ .		
<b>gmres rel tolerance</b>	real	$r = 1.08735e - 06$
Set relative convergence tolerance for gmres method		
Valid values: $0 \leq r$ .		
<b>gmres abs tolerance</b>	real	$r = 1e - 06$
Set absolute convergence tolerance for gmres method		
Valid values: $0 \leq r$ .		
<b>iterative method</b>	string	$s = \text{cg}$
Choose solver to use		
Valid values: $s = \text{cg}, \text{gm res}, \text{gmres}, \text{or pcg}$ .		
<b>cg preconditioner</b>	string	$s = \text{none}$
Choose preconditioner to use with cg method		
Valid values: $s = \text{gs}, \text{none}, \text{sgs}, \text{symgs}, \text{or user}$ .		
<b>gmres preconditioner</b>	string	$s = \text{none}$
Choose preconditioner to use with gmres method		
Valid values: $s = \text{ilu0}, \text{none}, \text{or user}$ .		

**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**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_invalid_value</i>	either the option name was not found or the provided option value is out of the valid range.
<i>aoclsparse_status_invalid_pointer</i>	the pointer to the problem handle is invalid.
<i>aoclsparse_status_internal_error</i>	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**

<i>in, out</i>	<i>handle</i>	the pointer to the problem handle data structure.
----------------	---------------	---

**Return values**

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_memory_error</i>	internal memory allocation error.
<i>aoclsparse_status_invalid_pointer</i>	the pointer to the problem handle is invalid.
<i>aoclsparse_status_internal_error</i>	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

<i>in, out</i>	<i>handle</i>	the pointer to the problem handle data structure.
----------------	---------------	---

## Return values

<i>aoclsparse_status_success</i>	the operation completed successfully.
<i>aoclsparse_status_memory_error</i>	internal memory allocation error.
<i>aoclsparse_status_invalid_pointer</i>	the pointer to the problem handle is invalid.
<i>aoclsparse_status_internal_error</i>	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

<i>in, out</i>	<i>handle</i>	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	<i>handle</i>	problem <i>handle</i> . Needs to be initialized by calling <a href="#">aoclsparse_itsol_s_init</a> or <a href="#">aoclsparse_itsol_d_init</a> .
in	<i>n</i>	the number of columns of the (square) linear system matrix.
in	<i>b</i>	the right hand side of the linear system. Must be a vector of size <i>n</i> .

**Return values**

<i>aoclsparse_status_success</i>	initialization completed successfully.
<i>aoclsparse_status_invalid_pointer</i>	one or more of the pointers <i>handle</i> , and <i>b</i> are invalid.
<i>aoclsparse_status_wrong_type</i>	<i>handle</i> was initialized with a different floating point precision than requested here, e.g. <a href="#">aoclsparse_itsol_d_init</a> (double precision) was used to initialize <i>handle</i> but <a href="#">aoclsparse_itsol_s_rci_input</a> (single precision) is being called instead of the correct double precision one, <a href="#">aoclsparse_itsol_d_rci_input</a> .
<i>aoclsparse_status_invalid_value</i>	<i>n</i> was set to a negative value.
<i>aoclsparse_status_memory_error</i>	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	<i>handle</i>	problem <i>handle</i> . Needs to be initialized by calling <a href="#">aoclsparse_itsol_s_init</a> or <a href="#">aoclsparse_itsol_d_init</a> .
in	<i>n</i>	the number of columns of the (square) linear system matrix.
in	<i>b</i>	the right hand side of the linear system. Must be a vector of size <i>n</i> .

**Return values**

<i>aoclsparse_status_success</i>	initialization completed successfully.
----------------------------------	--

## Return values

<code>aoclsparse_status_invalid_pointer</code>	one or more of the pointers <code>handle</code> , and <code>b</code> are invalid.
<code>aoclsparse_status_wrong_type</code>	<code>handle</code> was initialized with a different floating point precision than requested here, e.g. <code>aoclsparse_itsol_d_init</code> (double precision) was used to initialize <code>handle</code> but <code>aoclsparse_itsol_s_rci_input</code> (single precision) is being called instead of the correct double precision one, <code>aoclsparse_itsol_d_rci_input</code> .
<code>aoclsparse_status_invalid_value</code>	<code>n</code> was set to a negative value.
<code>aoclsparse_status_memory_error</code>	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	<i>handle</i>	problem <i>handle</i> . Needs to be previously initialized by <a href="#">aoclsparse_itsol_s_init</a> or <a href="#">aoclsparse_itsol_d_init</a> and then populated using either <a href="#">aoclsparse_itsol_s_rci_input</a> or <a href="#">aoclsparse_itsol_d_rci_input</a> , as appropriate.
in, out	<i>ircomm</i>	pointer to the reverse communication instruction flag and defined in <a href="#">aoclsparse_itsol_rci_job_</a> .
in, out	<i>u</i>	pointer to a generic vector of data. The solver will point to the data on which the operation defined by <a href="#">ircomm</a> needs to be applied.
in, out	<i>v</i>	pointer to a generic vector of data. The solver will ask that the result of the operation defined by <a href="#">ircomm</a> be stored in <i>v</i> .
in, out	<i>x</i>	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 <a href="#">Options</a> .
out	<i>rinfo</i>	(optional, can be nullptr) vector containing information and stats related to the iterative solve, see <a href="#">Information Array</a> . This parameter can be used to monitor progress and define a custom stopping criterion when the solver returns control to user with <a href="#">ircomm</a> = <a href="#">aoclsparse_rci_stopping_criterion</a> .

## 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	<i>handle</i>	problem <code>handle</code> . Needs to be previously initialized by <a href="#">aoclsparse_itsol_s_init</a> or <a href="#">aoclsparse_itsol_d_init</a> and then populated using either <a href="#">aoclsparse_itsol_s_rci_input</a> or <a href="#">aoclsparse_itsol_d_rci_input</a> , as appropriate.
in, out	<i>ircomm</i>	pointer to the reverse communication instruction flag and defined in <a href="#">aoclsparse_itsol_rci_job_</a> .
in, out	<i>u</i>	pointer to a generic vector of data. The solver will point to the data on which the operation defined by <code>ircomm</code> needs to be applied.
in, out	<i>v</i>	pointer to a generic vector of data. The solver will ask that the result of the operation defined by <code>ircomm</code> be stored in <code>v</code> .
in, out	<i>x</i>	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 <a href="#">Options</a> .
out	<i>rinfo</i>	(optional, can be <code>nullptr</code> ) vector containing information and stats related to the iterative solve, see <a href="#">Information Array</a> . This parameter can be used to monitor progress and define a custom stopping criterion when the solver returns control to user with <code>ircomm = aoclsparse_rci_stopping_criterion</code> .

#### 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 precond aoclsparse_int flag, const double *u, double *v, void *udata,
aoclsparse_int monit 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	<i>handle</i>	a valid problem handle, previously initialized by calling <a href="#">aoclsparse_itsol_s_init</a> or <a href="#">aoclsparse_itsol_d_init</a> .
in	<i>n</i>	the size of the square matrix <code>mat</code> .
in, out	<i>mat</i>	coefficient matrix $A$ .
in, out	<i>descr</i>	matrix descriptor for <code>mat</code> .
in	<i>b</i>	right-hand side dense vector $b$ .
in, out	<i>x</i>	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 <a href="#">Options</a> .
out	<i>rinfo</i>	(optional, can be nullptr) vector containing information and stats related to the iterative solve, see <a href="#">Information Array</a> .

## Parameters

in	<i>precond</i>	(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 <i>udata</i> , 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 <a href="#">Options</a> ), then this parameter need not be provided.
in	<i>monit</i>	(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 will pass $x$ containing the current iterate, $r$ stores the current residual vector ( $r = Ax - b$ ), <i>rinfo</i> contains the current stats, see <a href="#">Information Array</a> , and <i>udata</i> is a convenience pointer that can be used by the user to point to user data and is not used by the itsol framework.
in, out	<i>udata</i>	(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, const float *u, float *v, void *udata,
    aoclsparse_int monitconst 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	<i>handle</i>	a valid problem handle, previously initialized by calling <a href="#">aoclsparse_itsol_s_init</a> or <a href="#">aoclsparse_itsol_d_init</a> .
in	<i>n</i>	the size of the square matrix <code>mat</code> .
in, out	<i>mat</i>	coefficient matrix $A$ .
in, out	<i>descr</i>	matrix descriptor for <code>mat</code> .
in	<i>b</i>	right-hand side dense vector $b$ .
in, out	<i>x</i>	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 <a href="#">Options</a> .
out	<i>rinfo</i>	(optional, can be nullptr) vector containing information and stats related to the iterative solve, see <a href="#">Information Array</a> .
in	<i>precond</i>	(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 <code>udata</code> , 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 <a href="#">Options</a> ), then this parameter need not be provided.
in	<i>monit</i>	(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 will pass $x$ containing the current iterate, $r$ stores the current residual vector ( $r = Ax - b$ ), <code>rinfo</code> contains the current stats, see <a href="#">Information Array</a> , and <code>udata</code> is a convenience pointer that can be used by the user to point to user data and is not used by the itsol framework.
in, out	<i>udata</i>	(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` indicates 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_type
```

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

<a href="#">aoclsparse_operation_none</a>	Operate with matrix.
<a href="#">aoclsparse_operation_transpose</a>	Operate with transpose.
<a href="#">aoclsparse_operation_conjugate_transpose</a>	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 [aoclspase\\_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

<code>aoclsparse_fill_mode_lower</code>	lower triangular part is stored.
<code>aoclsparse_fill_mode_upper</code>	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

<code>aoclsparse_order_row</code>	Row major.
<code>aoclsparse_order_column</code>	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

<code>aoclsparse_status_success</code>	success.
<code>aoclsparse_status_not_implemented</code>	functionality is not implemented.
<code>aoclsparse_status_invalid_pointer</code>	invalid pointer parameter.
<code>aoclsparse_status_invalid_size</code>	invalid size parameter.
<code>aoclsparse_status_internal_error</code>	internal library failure.
<code>aoclsparse_status_invalid_value</code>	invalid parameter value.
<code>aoclsparse_status_invalid_index_value</code>	invalid index value.
<code>aoclsparse_status_maxit</code>	function stopped after reaching number of iteration limit.
<code>aoclsparse_status_user_stop</code>	user requested termination.
<code>aoclsparse_status_wrong_type</code>	function called on the wrong type (double/float).
<code>aoclsparse_status_memory_error</code>	memory allocation failure.
<code>aoclsparse_status_numerical_error</code>	numerical error, e.g., matrix is not positive definite, divide-by-zero error
<code>aoclsparse_status_invalid_operation</code>	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` function.

##### Enumerator

<code>aoclsparse_stage_nnz_count</code>	Only rowIndex array of the CSR matrix is computed internally.
<code>aoclsparse_stage_finalize</code>	Finalize computation. Has to be called only after <code>csr2m</code> call with <code>aoclsparse_stage_nnz_count</code> parameter.
<code>aoclsparse_stage_full_computation</code>	Perform the entire computation in a single step.



# Index

- aoclsparse\_analysis.h, 5
  - aoclsparse\_optimize, 5
  - aoclsparse\_set\_lu\_smoother\_hint, 6
  - aoclsparse\_set\_mv\_hint, 6
- aoclsparse\_auxiliary.h, 7
  - aoclsparse\_copy\_mat\_descr, 9
  - aoclsparse\_create\_dcsr, 14
  - aoclsparse\_create\_mat\_descr, 9
  - aoclsparse\_create\_scsr, 14
  - aoclsparse\_destroy, 16
  - aoclsparse\_destroy\_mat\_descr, 10
  - aoclsparse\_export\_mat\_csr, 15
  - aoclsparse\_get\_mat\_diag\_type, 13
  - aoclsparse\_get\_mat\_fill\_mode, 12
  - aoclsparse\_get\_mat\_index\_base, 10
  - aoclsparse\_get\_mat\_type, 11
  - aoclsparse\_get\_version, 8
  - aoclsparse\_set\_mat\_diag\_type, 13
  - aoclsparse\_set\_mat\_fill\_mode, 12
  - aoclsparse\_set\_mat\_index\_base, 10
  - aoclsparse\_set\_mat\_type, 11
- aoclsparse\_cmat
  - aoclsparse\_types.h, 75
- aoclsparse\_convert.h, 16
  - aoclsparse\_csr2bsr\_nnz, 22
  - aoclsparse\_csr2dia\_ndiag, 20
  - aoclsparse\_csr2ell\_width, 18
  - aoclsparse\_dcsr2bsr, 23
  - aoclsparse\_dcsr2csc, 25
  - aoclsparse\_dcsr2dense, 27
  - aoclsparse\_dcsr2dia, 21
  - aoclsparse\_dcsr2ell, 19
  - aoclsparse\_scsr2bsr, 23
  - aoclsparse\_scsr2csc, 24
  - aoclsparse\_scsr2dense, 26
  - aoclsparse\_scsr2dia, 20
  - aoclsparse\_scsr2ell, 18
- aoclsparse\_copy\_mat\_descr
  - aoclsparse\_auxiliary.h, 9
- aoclsparse\_create\_dcsr
  - aoclsparse\_auxiliary.h, 14
- aoclsparse\_create\_mat\_descr
  - aoclsparse\_auxiliary.h, 9
- aoclsparse\_create\_scsr
  - aoclsparse\_auxiliary.h, 14
- aoclsparse\_csr
  - aoclsparse\_types.h, 72
- aoclsparse\_csr2bsr\_nnz
  - aoclsparse\_convert.h, 22
- aoclsparse\_csr2dia\_ndiag
  - aoclsparse\_convert.h, 20
- aoclsparse\_csr2ell\_width
  - aoclsparse\_convert.h, 18
- aoclsparse\_csr\_mat
  - aoclsparse\_types.h, 77
- aoclsparse\_csr\_mat\_br4
  - aoclsparse\_types.h, 77
- aoclsparse\_dbarmv
  - aoclsparse\_functions.h, 40
- aoclsparse\_dcsr2bsr
  - aoclsparse\_convert.h, 23
- aoclsparse\_dcsr2csc
  - aoclsparse\_convert.h, 25
- aoclsparse\_dcsr2dense
  - aoclsparse\_convert.h, 27
- aoclsparse\_dcsr2dia
  - aoclsparse\_convert.h, 21
- aoclsparse\_dcsr2ell
  - aoclsparse\_convert.h, 19
- aoclsparse\_dcsr2m
  - aoclsparse\_functions.h, 50
- aoclsparse\_dcsrmm
  - aoclsparse\_functions.h, 49
- aoclsparse\_dcsrsv
  - aoclsparse\_functions.h, 32
- aoclsparse\_dcsrsv
  - aoclsparse\_functions.h, 44
- aoclsparse\_ddiamv
  - aoclsparse\_functions.h, 37
- aoclsparse\_dellmv
  - aoclsparse\_functions.h, 35
- aoclsparse\_destroy
  - aoclsparse\_auxiliary.h, 16
- aoclsparse\_destroy\_mat\_descr
  - aoclsparse\_auxiliary.h, 10
- aoclsparse\_dia\_mat
  - aoclsparse\_types.h, 77
- aoclsparse\_diag\_type
  - aoclsparse\_types.h, 73
- aoclsparse\_diag\_type\_
  - aoclsparse\_types.h, 77
- aoclsparse\_diag\_type\_non\_unit
  - aoclsparse\_types.h, 77
- aoclsparse\_diag\_type\_unit
  - aoclsparse\_types.h, 77
- aoclsparse\_dilu\_smoother
  - aoclsparse\_functions.h, 53
- aoclsparse\_dmat

- aoclsparse\_types.h, 75
- aoclsparse\_dmv
  - aoclsparse\_functions.h, 42
- aoclsparse\_dtrsv
  - aoclsparse\_functions.h, 47
- aoclsparse\_ell\_csr\_hyb\_mat
  - aoclsparse\_types.h, 77
- aoclsparse\_ell\_mat
  - aoclsparse\_types.h, 77
- aoclsparse\_ellt\_csr\_hyb\_mat
  - aoclsparse\_types.h, 77
- aoclsparse\_ellt\_mat
  - aoclsparse\_types.h, 77
- aoclsparse\_export\_mat\_csr
  - aoclsparse\_auxiliary.h, 15
- aoclsparse\_fill\_mode
  - aoclsparse\_types.h, 73
- aoclsparse\_fill\_mode\_
  - aoclsparse\_types.h, 77
- aoclsparse\_fill\_mode\_lower
  - aoclsparse\_types.h, 78
- aoclsparse\_fill\_mode\_upper
  - aoclsparse\_types.h, 78
- aoclsparse\_functions.h, 28
  - aoclsparse\_dbstrmv, 40
  - aoclsparse\_dcsr2m, 50
  - aoclsparse\_dcsrmm, 49
  - aoclsparse\_dcsrsv, 32
  - aoclsparse\_dcsrsv, 44
  - aoclsparse\_ddiamv, 37
  - aoclsparse\_dellmv, 35
  - aoclsparse\_dilu\_smoother, 53
  - aoclsparse\_dmv, 42
  - aoclsparse\_dtrsv, 47
  - aoclsparse\_sbsrmv, 38
  - aoclsparse\_scsr2m, 52
  - aoclsparse\_scsrmm, 48
  - aoclsparse\_scsrsv, 30
  - aoclsparse\_scsrsv, 43
  - aoclsparse\_sdiamv, 36
  - aoclsparse\_sellmv, 33
  - aoclsparse\_silu\_smoother, 54
  - aoclsparse\_smv, 41
  - aoclsparse\_strsv, 45
- aoclsparse\_get\_mat\_diag\_type
  - aoclsparse\_auxiliary.h, 13
- aoclsparse\_get\_mat\_fill\_mode
  - aoclsparse\_auxiliary.h, 12
- aoclsparse\_get\_mat\_index\_base
  - aoclsparse\_auxiliary.h, 10
- aoclsparse\_get\_mat\_type
  - aoclsparse\_auxiliary.h, 11
- aoclsparse\_get\_version
  - aoclsparse\_auxiliary.h, 8
- aoclsparse\_ilu0
  - aoclsparse\_types.h, 77
- aoclsparse\_ilu\_type
  - aoclsparse\_types.h, 73
- aoclsparse\_ilu\_type\_
  - aoclsparse\_types.h, 75
- aoclsparse\_ilup
  - aoclsparse\_types.h, 77
- aoclsparse\_index\_base
  - aoclsparse\_types.h, 72
- aoclsparse\_index\_base\_
  - aoclsparse\_types.h, 74
- aoclsparse\_index\_base\_one
  - aoclsparse\_types.h, 75
- aoclsparse\_index\_base\_zero
  - aoclsparse\_types.h, 75
- aoclsparse\_itsol\_d\_init
  - aoclsparse\_solvers.h, 61
- aoclsparse\_itsol\_d\_rci\_input
  - aoclsparse\_solvers.h, 62
- aoclsparse\_itsol\_d\_rci\_solve
  - aoclsparse\_solvers.h, 64
- aoclsparse\_itsol\_d\_solve
  - aoclsparse\_solvers.h, 66
- aoclsparse\_itsol\_destroy
  - aoclsparse\_solvers.h, 62
- aoclsparse\_itsol\_handle\_prn\_options
  - aoclsparse\_solvers.h, 59
- aoclsparse\_itsol\_option\_set
  - aoclsparse\_solvers.h, 59
- aoclsparse\_itsol\_rci\_job\_
  - aoclsparse\_solvers.h, 58
- aoclsparse\_itsol\_s\_init
  - aoclsparse\_solvers.h, 61
- aoclsparse\_itsol\_s\_rci\_input
  - aoclsparse\_solvers.h, 63
- aoclsparse\_itsol\_s\_rci\_solve
  - aoclsparse\_solvers.h, 65
- aoclsparse\_itsol\_s\_solve
  - aoclsparse\_solvers.h, 68
- aoclsparse\_mat\_descr
  - aoclsparse\_types.h, 72
- aoclsparse\_matrix\_data\_type
  - aoclsparse\_types.h, 72
- aoclsparse\_matrix\_data\_type\_
  - aoclsparse\_types.h, 75
- aoclsparse\_matrix\_format\_type
  - aoclsparse\_types.h, 73
- aoclsparse\_matrix\_format\_type\_
  - aoclsparse\_types.h, 77
- aoclsparse\_matrix\_type
  - aoclsparse\_types.h, 72
- aoclsparse\_matrix\_type\_
  - aoclsparse\_types.h, 75
- aoclsparse\_matrix\_type\_general
  - aoclsparse\_types.h, 75
- aoclsparse\_matrix\_type\_hermitian
  - aoclsparse\_types.h, 75
- aoclsparse\_matrix\_type\_symmetric
  - aoclsparse\_types.h, 75
- aoclsparse\_matrix\_type\_triangular
  - aoclsparse\_types.h, 75



- aoclspase\_operation
  - aoclspase\_types.h, 72
- aoclspase\_operation\_
  - aoclspase\_types.h, 74
- aoclspase\_operation\_conjugate\_transpose
  - aoclspase\_types.h, 74
- aoclspase\_operation\_none
  - aoclspase\_types.h, 74
- aoclspase\_operation\_transpose
  - aoclspase\_types.h, 74
- aoclspase\_optimize
  - aoclspase\_analysis.h, 5
- aoclspase\_order
  - aoclspase\_types.h, 73
- aoclspase\_order\_
  - aoclspase\_types.h, 78
- aoclspase\_order\_column
  - aoclspase\_types.h, 78
- aoclspase\_order\_row
  - aoclspase\_types.h, 78
- aoclspase\_rci\_interrupt
  - aoclspase\_solvers.h, 58
- aoclspase\_rci\_mv
  - aoclspase\_solvers.h, 59
- aoclspase\_rci\_precond
  - aoclspase\_solvers.h, 59
- aoclspase\_rci\_start
  - aoclspase\_solvers.h, 59
- aoclspase\_rci\_stop
  - aoclspase\_solvers.h, 59
- aoclspase\_rci\_stopping\_criterion
  - aoclspase\_solvers.h, 59
- aoclspase\_request
  - aoclspase\_types.h, 74
- aoclspase\_request\_
  - aoclspase\_types.h, 79
- aoclspase\_sbsrmv
  - aoclspase\_functions.h, 38
- aoclspase\_scsr2bsr
  - aoclspase\_convert.h, 23
- aoclspase\_scsr2csc
  - aoclspase\_convert.h, 24
- aoclspase\_scsr2dense
  - aoclspase\_convert.h, 26
- aoclspase\_scsr2dia
  - aoclspase\_convert.h, 20
- aoclspase\_scsr2ell
  - aoclspase\_convert.h, 18
- aoclspase\_scsr2m
  - aoclspase\_functions.h, 52
- aoclspase\_scsrmm
  - aoclspase\_functions.h, 48
- aoclspase\_scsrmlv
  - aoclspase\_functions.h, 30
- aoclspase\_scsrsv
  - aoclspase\_functions.h, 43
- aoclspase\_sdiamv
  - aoclspase\_functions.h, 36
- aoclspase\_sellmv
  - aoclspase\_functions.h, 33
- aoclspase\_set\_lu\_smoother\_hint
  - aoclspase\_analysis.h, 6
- aoclspase\_set\_mat\_diag\_type
  - aoclspase\_auxiliary.h, 13
- aoclspase\_set\_mat\_fill\_mode
  - aoclspase\_auxiliary.h, 12
- aoclspase\_set\_mat\_index\_base
  - aoclspase\_auxiliary.h, 10
- aoclspase\_set\_mat\_type
  - aoclspase\_auxiliary.h, 11
- aoclspase\_set\_mv\_hint
  - aoclspase\_analysis.h, 6
- aoclspase\_silu\_smoother
  - aoclspase\_functions.h, 54
- aoclspase\_smat
  - aoclspase\_types.h, 75
- aoclspase\_smv
  - aoclspase\_functions.h, 41
- aoclspase\_solvers.h, 55
  - aoclspase\_itsol\_d\_init, 61
  - aoclspase\_itsol\_d\_rci\_input, 62
  - aoclspase\_itsol\_d\_rci\_solve, 64
  - aoclspase\_itsol\_d\_solve, 66
  - aoclspase\_itsol\_destroy, 62
  - aoclspase\_itsol\_handle\_prn\_options, 59
  - aoclspase\_itsol\_option\_set, 59
  - aoclspase\_itsol\_rci\_job\_, 58
  - aoclspase\_itsol\_s\_init, 61
  - aoclspase\_itsol\_s\_rci\_input, 63
  - aoclspase\_itsol\_s\_rci\_solve, 65
  - aoclspase\_itsol\_s\_solve, 68
  - aoclspase\_rci\_interrupt, 58
  - aoclspase\_rci\_mv, 59
  - aoclspase\_rci\_precond, 59
  - aoclspase\_rci\_start, 59
  - aoclspase\_rci\_stop, 59
  - aoclspase\_rci\_stopping\_criterion, 59
- aoclspase\_stage\_finalize
  - aoclspase\_types.h, 79
- aoclspase\_stage\_full\_computation
  - aoclspase\_types.h, 79
- aoclspase\_stage\_nnz\_count
  - aoclspase\_types.h, 79
- aoclspase\_status
  - aoclspase\_types.h, 73
- aoclspase\_status\_
  - aoclspase\_types.h, 78
- aoclspase\_status\_internal\_error
  - aoclspase\_types.h, 78
- aoclspase\_status\_invalid\_index\_value
  - aoclspase\_types.h, 78
- aoclspase\_status\_invalid\_operation
  - aoclspase\_types.h, 78
- aoclspase\_status\_invalid\_pointer
  - aoclspase\_types.h, 78
- aoclspase\_status\_invalid\_size

- aoclsparse\_types.h, 78
- aoclsparse\_status\_invalid\_value
  - aoclsparse\_types.h, 78
- aoclsparse\_status\_maxit
  - aoclsparse\_types.h, 78
- aoclsparse\_status\_memory\_error
  - aoclsparse\_types.h, 78
- aoclsparse\_status\_not\_implemented
  - aoclsparse\_types.h, 78
- aoclsparse\_status\_numerical\_error
  - aoclsparse\_types.h, 78
- aoclsparse\_status\_success
  - aoclsparse\_types.h, 78
- aoclsparse\_status\_user\_stop
  - aoclsparse\_types.h, 78
- aoclsparse\_status\_wrong\_type
  - aoclsparse\_types.h, 78
- aoclsparse\_strsv
  - aoclsparse\_functions.h, 45
- aoclsparse\_types.h, 70
  - aoclsparse\_cmat, 75
  - aoclsparse\_csr, 72
  - aoclsparse\_csr\_mat, 77
  - aoclsparse\_csr\_mat\_br4, 77
  - aoclsparse\_dia\_mat, 77
  - aoclsparse\_diag\_type, 73
  - aoclsparse\_diag\_type\_, 77
  - aoclsparse\_diag\_type\_non\_unit, 77
  - aoclsparse\_diag\_type\_unit, 77
  - aoclsparse\_dmat, 75
  - aoclsparse\_ell\_csr\_hyb\_mat, 77
  - aoclsparse\_ell\_mat, 77
  - aoclsparse\_ellt\_csr\_hyb\_mat, 77
  - aoclsparse\_ellt\_mat, 77
  - aoclsparse\_fill\_mode, 73
  - aoclsparse\_fill\_mode\_, 77
  - aoclsparse\_fill\_mode\_lower, 78
  - aoclsparse\_fill\_mode\_upper, 78
  - aoclsparse\_ilu0, 77
  - aoclsparse\_ilu\_type, 73
  - aoclsparse\_ilu\_type\_, 75
  - aoclsparse\_ilup, 77
  - aoclsparse\_index\_base, 72
  - aoclsparse\_index\_base\_, 74
  - aoclsparse\_index\_base\_one, 75
  - aoclsparse\_index\_base\_zero, 75
  - aoclsparse\_mat\_descr, 72
  - aoclsparse\_matrix\_data\_type, 72
  - aoclsparse\_matrix\_data\_type\_, 75
  - aoclsparse\_matrix\_format\_type, 73
  - aoclsparse\_matrix\_format\_type\_, 77
  - aoclsparse\_matrix\_type, 72
  - aoclsparse\_matrix\_type\_, 75
  - aoclsparse\_matrix\_type\_general, 75
  - aoclsparse\_matrix\_type\_hermitian, 75
  - aoclsparse\_matrix\_type\_symmetric, 75
  - aoclsparse\_matrix\_type\_triangular, 75
  - aoclsparse\_operation, 72
  - aoclsparse\_operation\_, 74
  - aoclsparse\_operation\_conjugate\_transpose, 74
  - aoclsparse\_operation\_none, 74
  - aoclsparse\_operation\_transpose, 74
  - aoclsparse\_order, 73
  - aoclsparse\_order\_, 78
  - aoclsparse\_order\_column, 78
  - aoclsparse\_order\_row, 78
  - aoclsparse\_request, 74
  - aoclsparse\_request\_, 79
  - aoclsparse\_smat, 75
  - aoclsparse\_stage\_finalize, 79
  - aoclsparse\_stage\_full\_computation, 79
  - aoclsparse\_stage\_nnz\_count, 79
  - aoclsparse\_status, 73
  - aoclsparse\_status\_, 78
  - aoclsparse\_status\_internal\_error, 78
  - aoclsparse\_status\_invalid\_index\_value, 78
  - aoclsparse\_status\_invalid\_operation, 78
  - aoclsparse\_status\_invalid\_pointer, 78
  - aoclsparse\_status\_invalid\_size, 78
  - aoclsparse\_status\_invalid\_value, 78
  - aoclsparse\_status\_maxit, 78
  - aoclsparse\_status\_memory\_error, 78
  - aoclsparse\_status\_not\_implemented, 78
  - aoclsparse\_status\_numerical\_error, 78
  - aoclsparse\_status\_success, 78
  - aoclsparse\_status\_user\_stop, 78
  - aoclsparse\_status\_wrong\_type, 78
  - aoclsparse\_zmat, 75
  - DLL\_PUBLIC, 71
- aoclsparse\_zmat
  - aoclsparse\_types.h, 75
- DLL\_PUBLIC
  - aoclsparse\_types.h, 71