frovedis::sliced_colmajor_vector_local<T>

NAME

`frovedis::sliced_colmajor_vector_local<T>` - a data structure containing the row or column wise slicing information of a two-dimensional `frovedis::colmajor_matrix_local<T>`

SYNOPSIS

`#include <frovedis/matrix/sliced_vector.hpp>`

Constructors

sliced_colmajor_vector_local ()
sliced_colmajor_vector_local (const `colmajor_matrix_local<T>`& m)
sliced_colmajor_vector_local (const `std::vector<T>`& v)

Public Member Functions

bool is_valid () const
void debug_print () const

Public Data Members

T* data
size_t size
size_t stride

DESCRIPTION

In order to perform vector operations on some rows or on some columns of a dense matrix, frovedis provides some sliced data structures. `sliced_colmajor_vector_local<T>` is one of them. It is actually not a real vector, rather it only contains some slicing information of a physical `colmajor_matrix_local<T>`. Thus any changes performed on the sliced vector, would actually make changes on the specific row or column of the physical matrix from which the slice was made.

Like `colmajor_matrix_local<T>`, a `sliced_colmajor_vector_local<T>` is also a template based structure with type **"T"**. This has the below structure:

```
template <class T>
struct sliced_colmajor_vector_local {
  T* data;        // pointer pointing to the begining of the row or column to be sliced
  size_t size;    // number of elements in the sliced vector
  size_t stride;  // stride between two consecutive elements in the sliced vector
};
```

E.g., if a physical `colmajor_matrix_local<T>` M has dimensions 4x4 and its 2nd row needs to be sliced, then
"data" will hold the address of M[0][1] (not M[1][0], since matrix is stored in colmajor order),
"size" would be 4 (number of elements in 2nd row),
and "stride" would be 4 (since matrix is stored in colmajor order, the stride between two consecutive elements in a row would be equal to leading dimension of that matrix, i.e., number of rows in that matrix).

On the other hand, if 2nd column needs to be sliced, then
"data" will hold the address of M[1][0] (not M[0][1], since matrix is stored in colmajor order),
"size" would be 4 (number of elements in 2nd column),
and "stride" would be 1 (since matrix is stored in colmajor order, the consecutive elements in a column would be placed one after another).

Such vectors are very useful in operations of external libraries like blas etc.

Constructor Documentation

sliced_colmajor_vector_local ()

This is the default constructor which creates an empty sliced vector with size = stride = 0 and "data" points to NULL. In general of no use, unless it is needed to manipulate the slice information explicitly.

sliced_colmajor_vector_local (const `colmajor_matrix_local<T>`& m)

This is a special constructor for implicit conversion. This constructor treats an entire physical matrix as a sliced vector. Thus the created `sliced_colmajor_vector_local<T>` would have "size" equals to number of rows in the input `colmajor_matrix_local<T>`, "stride" equals to 1 and "data" pointing to the base address of the input `colmajor_matrix_local<T>`. Please note that such conversion can only be posible if the input matrix can be treated as a column vector (a matrix with multiple rows and single column), else it throws an exception.

sliced_colmajor_vector_local (const `std::vector<T>`& v)

This is a special constructor for implicit conversion. This constructor treats an entire physical vector as a sliced vector. Thus the created `sliced_colmajor_vector_local<T>` would have "size" equals to the length of the input `std::vector<T>`, "stride" equals to 1 and "data" pointing to the base address of the input vector.

Public Member Function Documentation

bool is_valid () const

This function returns true, if the caller object is a valid sliced vector, else it returns false.
Kindly note that an empty sliced vector is also an invalid sliced vector, since no valid operation can be performed on its data pointing to NULL.

void debug_print () const

It prints the contents of the sliced row or column of the original (physical) `colmajor_matrix_local<T>` on the user standard output terminal.

Public Data Member Documentation

data

A pointer of type "T" pointing to the beginning of the row or column of a physical `colmajor_matrix_local<T>` from which slice has been made.

size

A size_t attribute to contain the number of elements in the sliced vector.

stride

A size_t attribute to contain the stride between two consecutive elements in a sliced vector.

Public Global Function Documentation

make_row_vector (mat, row_index)

Parameters
mat: An object of either `colmajor_matrix_local<T>` or `sliced_colmajor_matrix_local<T>` type.
row_index: A size_t attribute to indicate the row index to be sliced.

Purpose
This function accepts a valid `colmajor_matrix_local<T>` or `sliced_colmajor_matrix_local<T>` with the row index to be sliced (index starts with 0). On receiving valid inputs, it outputs a `sliced_colmajor_vector_local<T>` object containing the slicing information, else it throws an exception.

Example:
If a physical `colmajor_matrix_local<T>` "mat" has the dimensions 4x4 and its 2nd row needs to be sliced, then this function should be called like:

```
auto rvec = make_row_vector(mat,1); // row index of second row is 1

Input (mat):        Output (rvec):
------------        --------------
1 2 3 4       =>    5 6 7 8
5 6 7 8
8 7 6 5
4 3 2 1
```

Now if it is needed to slice the 2nd row from its 4th block (sub-matrix), then the operations can be performed as per the code below:

```
auto smat   = make_sliced_colmajor_matrix_local(mat,2,2,2,2);
auto s_rvec = make_row_vector(smat,1);
```

First the original matrix needs to be sliced to get its 4th block (3rd row and 3rd column till 4th row and 4th column) and then 2nd row is to be sliced from the sub-matrix.

Kindly note that 2nd row of “smat” is actually the 4th row of the physical matrix “mat”, but this function takes care of it internally. Thus user only needs to take care of the index of the input sliced matrix, not the actual physical matrix.

```
Input (mat):        Output (smat):      Output (s_rvec):
-------------       --------------      ----------------
1 2 3 4             6 5             =>  2 1
5 6 7 8       =>    2 1
8 7 6 5
4 3 2 1
```

Return Value
On success, this function returns the sliced row vector of the type `sliced_colmajor_vector_local<T>`. Otherwise it throws an exception.

make_col_vector (mat, col_index)

Parameters
mat: An object of either `colmajor_matrix_local<T>` or `sliced_colmajor_matrix_local<T>` type.
col_index: A size_t attribute to indicate the column index needs to be sliced.

Purpose
This function accepts a valid `colmajor_matrix_local<T>` or `sliced_colmajor_matrix_local<T>` with the column index to be sliced (index starts with 0). On receiving the valid inputs, it outputs a `sliced_colmajor_vector_local<T>` object containing the slicing information, else it throws an exception.

Example:
If a physical `colmajor_matrix_local<T>` “mat” has the dimensions 4x4 and its 2nd column needs to be sliced, then this function should be called like:

```
auto cvec = make_col_vector(mat,1); // column index of second column is 1
```

```
Input (mat):         Output (cvec):
------------         --------------
1 2 3 4       =>     2 6 7 3
5 6 7 8
8 7 6 5
4 3 2 1
```

Now if it is needed to slice the 2nd column from its 4th block (sub-matrix), then the operations can be performed as per the code below:

```
auto smat   = make_sliced_colmajor_matrix_local(mat,2,2,2,2);
auto s_cvec = make_col_vector(smat,1);
```

First the original matrix needs to be sliced to get its 4th block (3rd row and 3rd column till 4th row and 4th column) and then 2nd column is to be sliced from the sub-matrix.

Kindly note that 2nd column of “smat” is actually the 4th column of the physical matrix “mat”, but this function takes care of it internally. Thus user only needs to take care of the index of the input sliced matrix, not the actual physical matrix.

```
Input (mat):        Output (smat):      Output (s_cvec):
-------------       --------------      ----------------
1 2 3 4             6 5            =>   5 1
5 6 7 8      =>     2 1
8 7 6 5
4 3 2 1
```

Return Value
On success, it returns the sliced column vector of the type `sliced_colmajor_vector_local<T>`. Otherwise it throws an exception.

SEE ALSO

colmajor_matrix, sliced_colmajor_matrix_local