Appendix: Examples of writing test main files

The use of the sparse matrix algorithm library is divided into three stages. The first stage is matrix generation, which includes conventional matrix file reading and sparse matrix generation in the corresponding format. The second stage is the conversion of the matrix format, which will convert the matrix to a sparse matrix in the CSR storage format. If the sparse matrix is in the CSR format, this stage can be ignored. The third stage is the use of specific functions.

In the first stage, you need to perform targeted read and write operations for the content of the matrix file, and declare the necessary parameter variables. The matrix data structure provided by the sparse matrix algorithm library is mal_sparse_matrix_t, and different storage formats can be obtained according to different matrix creation functions. At this stage, the algorithm library supports the use of mal_sparse_?_create_coo to create COO sparse matrix. The following will briefly introduce the use of the algorithm library with the use of the function mal_sparse_s_mm as an example.

To use the algorithm library, you need to include the necessary header file (line 1). In the main function, first analyze the function execution parameters of the program (line 7), and then supply the function of reading the matrix file. In this example, the mal_read_coo function reads the matrix file in COO format (line 13), the data type of the non-zero element is single-precision real numbers, reads the matrix file, and obtains the corresponding COO sparse matrix, and then analyzes the given matrix parameters (line 14-16 lines), including whether to transpose the matrix, matrix storage characteristics and matrix element characteristics. Use mal_sparse_s_create_coo to create the COO matrix (line 19), and use mal_sparse_convert_csr to convert the matrix format from COO to CSR (line 20). Before performing the operation with mal_sparse_s_mm, use mal_set_thread_num(int thread_num) to set the number of worker threads (line 22). After the function is executed (line 23), the created matrix is destroyed (line 25, 26), and the memory is released (line 27-29).

Pseudocode: the use of mal_sparse_s_mm

```
#include <mal_spblas.h>
int main()
{
    const char *file; //path of matrix file
    int thread_num;
    MAL_INT m, n, nnz;
  arg_parse_common(argc, argv, &file, &thread_num, &check);//Parsing parameters
    MAL_INT *row_index, *col_index;
    float *values;
    const float alpha = 3.;
    const float beta = 2.;

  mal_read_coo(file, &m, &n, &nnz, &row_index, &col_index, &values);
  mal_sparse_operation_t transA = mal_args_get_transA(argc, argv);
   mal_sparse_layout_t layout = mal_args_get_layout(argc, argv);
  struct mal_matrix_descr descr = mal_args_get_matrix_descrA(argc, argv);

    mal_sparse_matrix_t coo, csr;

 mal_sparse_s_create_coo(&coo, MAL_SPARSE_INDEX_BASE_ZERO, m, n, nnz, row_index, col_index, val
ues);
    mal_sparse_convert_csr(coo, MAL_SPARSE_OPERATION_NON_TRANSPOSE, &csr);

    mal_set_thread_num(int thread_num);
    mal_sparse_s_mm(transA, alpha, csr, descr, layout, x, columns, ldx, beta, y, ldy);

    mal_sparse_destroy(coo);
    mal_sparse_destroy(csr);
  mal_free(row_index);
  mal_free(col_index);
  mal_free(values);
}
```