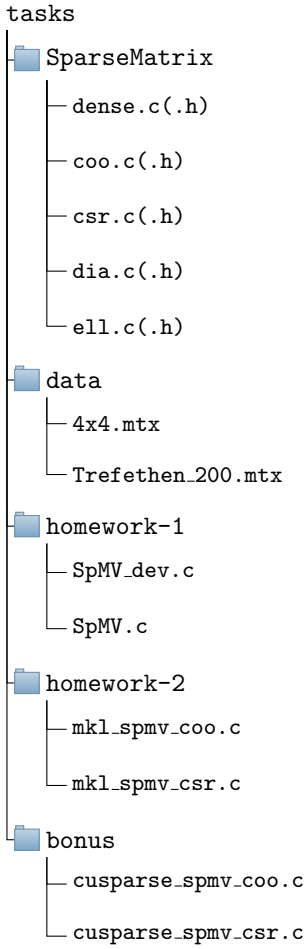
High-Performance (Sparse) Matrix Computations
Jan. 24 2022, RWTH Aachen

Homework

1. Directory Tree and Workflow

The directory tree for the homework is listed as follows:



1.1. **compile the code.** Here is the instruction to compile the code by `CMake` with the command line tool.

```
cd tasks #go into the main directory of homework
mkdir build
cd build
cmake .. #use CMake compile the codes
make
```

Listing 1. How to build the code

1.2. **available matrices.** Several sparse matrices from applications have been provided accompanying with the codes. These matrices can be used for the initial tests. The example listed below is a 4×4 test matrices with 9 entries.

```
%%MatrixMarket matrix coordinate real general
%-------------------------------------------------------------------------------
% TEST
%-------------------------------------------------------------------------------
4 4 9
1 1 1
3 1 5
1 2 7
2 2 2
4 2 6
2 3 8
3 3 3
3 4 9
4 4 4
```

Listing 2. Example of MatrixMarket format

The matrices are stored in MatrixMarket format[1]. More matrices can be downloaded from the MatrixMarket or SuiteSparse Matrix Collection.

1.3. **run executables.** Here are examples showing the way to run compiled executables:

```
#homework 1
## for development purpose
./homework-1/spmv_dev.exe
## for benchmark purpose
./homework-1/spmv.exe ../data/${NAME_OF_MATRIX_FOR_TEST}

#homework 2
## MKL with CSR format
./homework-2/mkl_spmv_csr.exe ../data/${NAME_OF_MATRIX_FOR_TEST}
## MKL with COO format
./homework-2/mkl_spmv_coo.exe ../data/${NAME_OF_MATRIX_FOR_TEST}

#bonus
## cuSPARSE with CSR format
./bonus/cusparse_spmv_csr.exe ../data/${NAME_OF_MATRIX_FOR_TEST}
## cuSPARSE with COO format
./bonus/cusparse_spmv_csr.exe ../data/${NAME_OF_MATRIX_FOR_TEST}
```

Listing 3. How to run the code

2. homework-1

The objective of this homework is to implement SpMV for different sparse matrices format: `COO`, `CSR`, `DIA` and `ELL` which have been shown in the slides of this module.

For each sparse matrix format, a header file and a related source file have already been implemented within the directory `SparseMatrix`. Below is an example header file `coo.h`, which provides a `struct` for the `COO` format.

```
/*in file SparseMatrix/coo.h*/
typedef struct coo
{
    int m; //size of matrix
    int nnz; //number of non-zero elements in the sparse matrix

    int *rowind; //a pointer storing the row indices of non-zeros elements
    int *colind; //a pointer storing the column indices of non-zeros elements
    double *val; //a pointer storing the non-zeros elements

} coo_t;
```

Listing 4. COO header file

Similarly, the `structs` for `CSR`, `DIA` and `ELL` can be found within `csr.h`, `dia.h` and `ell.h`.

2.1. **Task 1: sequential SpMV.** This task is to implement sequential `SpMV` for all available sparse matrix formats.

Requirement:

- `CMake version > 3.18`
- a `C` compiler

The implementation should take place with the related source file. for example, for `COO` format, the sequential SpMV should be implemented in `coo.c` in the `coo_spmv` function.

```
/*in file SparseMatrix/coo.c*/
void coo_spmv(coo_t *a, double *x, double *y){

    /* put your code here*/

}
```

Listing 5. SpMV function in COO source file

The correctness of implementations can be checked by running

```
./homework-1/spmv_dev.exe
```

Listing 6. How to run the code

It will compare the results obtained for each format with the one of a dense matrix-vector operation. For the validation purpose, the implementation of SpMV should be in the order as: `COO`, `CSR`, `DIA` and `ELL`. The matrix used for verification is `4x4.mtx` in the directory `data`.

[1] https://math.nist.gov/MatrixMarket/formats.html

2.2. **Task 2: Parallelize SpMV with OpenMP.** Try to parallelize the implementations of SpMV for four different sparse matrix storage formats.

Requirement:

- `CMake version > 3.18`
- a `C` compiler with `OpenMP support`

2.3. **Task 3: Test with different matrices and threads.** The objective of this task is to test the implemented sequential and parallel SpMV with different matrices and different threads number.

Requirement:

- `CMake version > 3.18`
- a `C` compiler with `OpenMP support`

```
#Set this environment variable to the number of threads you want to use
export OMP_NUM_THREADS=4

./homework-1/spmv.exe ../data/${NAME_OF_MATRIX_FOR_TEST}
```

Listing 7. Run SpMV tests with input matrices and multi-threading

3. Homework-2

The objective of this homework is to complete the implementation of SpMV based on MKL for both `COO` and `CSR` formats. Most parts of implementation have already been available within the directory `homework-2`. You need to fill the missing input arguments when calling the MKL routines.

Requirement:

- `CMake version > 3.18`
- a `C` compiler
- `Intel MKL` (on `CLAIX`)

```
/*homework-2/mkl_spmv_coo.c*/
/*Begin: complete arguments of the following lines*/
mkl_sparse_d_create_coo(&mkl_coo_handle, SPARSE_INDEX_BASE_ZERO, , , , , , );
/*End*/

/*homework-2/mkl_spmv_csr.c*/
/*Begin: complete arguments of the following lines*/
mkl_sparse_d_create_csr(&mkl_csr_handle, SPARSE_INDEX_BASE_ZERO, , , , , , );
/*End*/

/*Both homework-2/mkl_spmv_coo.c and homework-2/mkl_spmv_csr.c*/
/*Begin: complete arguments of the following lines*/
mkl_sparse_d_mv(SPARSE_OPERATION_NON_TRANSPOSE, , , , , , );
/*End*/
```

Listing 8. Code to be completed in `homework-2/mkl_spmv_csr.c`

The documentation about the three API can be found:

- mkl_sparse_?_create_coo
- mkl_sparse_?_create_csr
- mkl_sparse_?_mv

Attention: For the compilation reason, the `build` of the source codes of `homework-2` is disabled in default. So if you want to work on this task, please re-run the `CMake` command as follows with the flag option `-DBUILD_HOMEWORK2=ON`.

```
cmake .. -DBUILD_HOMEWORK2=ON
make
```

Listing 9. Re-run `CMake` to enable the compilation of `homework-2`

4. Bonus: try SpMV in cuSPARSE on GPU

Try with the SpMV provided by `cuSPARSE` library on single GPU for both `COO` and `CSR`.

Requirement:

- `CMake version > 3.18`
- a `C` compiler
- `Nvidia CUDA installation` (on `CLAIX`)

The codes are already completed, the tasks are:

- Run them on supercomputer `CLAIX` with different matrices (sparsity pattern, size, etc)
- Identify the time cost of memory transfers between GPU and CPU as a fraction of total time of execution