Sampled Dense Matrix Multiplication

IN3200/IN4200 Obligatory assignment 1, Spring 2026

Note: This is one of the two obligatory assignments that both need to be approved so that a student is allowed to take the final exam. (The grade of the final exam is otherwise *not* related to these two assignments.)

Note: Discussions between the students are allowed, but each student should write her/his own implementations. The details about the submission can be found in Section 4.4.

1 Objectives

This obligatory assignment has the following objectives:

- The student is given a hands-on exercise of implementing a prescribed numerical algorithm.
- The student is asked to implement the needed C functions by strictly following a pre-defined syntax of each function. (It is an important step in writing modular scientific software.)
- The student is asked to learn about dividing an entire code into multiple program files and about compiling a multi-file code.
- The student is given a hands-on exercise of reading from realistic data files.
- The student is given a hands-on exercise of using two different data structures for storing sparse matrices.
- The student is given a hands-on exercise of simple OpenMP parallelization of loops.

2 Introduction

Sampled dense matrix multiplication is an important operation used in machine learning algorithms. In this project, specifically, we are interested in the following computation:

$$C = (A \times B) \odot S, \tag{1}$$

where A, B, C and S are all square matrices with n rows and n columns. Only A and B are dense matrices, whereas C and S are both sparse matrices and have the same sparsity pattern. In (1), the symbol of $\times$ denotes the standard matrix-matrix multiplication, and the symbol of $\odot$ denotes elementwise matrix-matrix multiplication. That is, the resulting value in matrix C on row i and column j will be computed as

$$C_{i,j} = (A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + \ldots + A_{i,n}B_{n,j}) \cdot S_{i,j} = \left(\sum_{k=1}^{n} A_{i,k}B_{k,j} \right) \cdot S_{i,j}. \tag{2}$$

It is very important to notice that computation is only associated with the nonzero values in the sparse matrix S. Matrix C thus has the same sparsity pattern as matrix S.

3 Two storage formats for sparse matrices

The third slide deck for Chapter 3 has described two storage formats for sparse matrices in a C program: *coordinate storage* (COO) and *compressed row storage* (CRS). Both the formats only store the nonzero values of a sparse matrix, but differ in whether the nonzero values are stored in a "random" order or organized "row-after-row".

3.1 COO format

The COO format involves three 1D arrays: `row_idx`, `col_idx` and `val`. All the three arrays are of length `nnz`, which denotes the number of nonzero values in the sparse matrix. Each value of array `val` is of type `double` and contains one nonzero value of the matrix. Arrays `row_idx` and `col_idx` have integer values (of type `int`), each storing the corresponding row or column index for a nonzero value.

In this project, the following data structure should be used to implement the COO storage format (for a square sparse matrix):

```
struct sparse_mat_coo {
  int n, nnz;
  double *val;
  int *row_idx, *col_idx;
};
```

3.2 CRS format

The CRS format also involves three 1D arrays: `row_ptr`, `col_idx` and `val`. The length of array `row_ptr` is however `n+1`, whereas arrays `col_idx` and `val` are still of length `nnz` as in COO. (The textbook gives a brief description of the CRS format on page 87, but has a typo in saying that the length of `row_ptr` is `n`, which should be `n+1`.) Note that the nonzero values from row `i` of the sparse matrix are stored in positions `row_ptr[i]`, `row_ptr[i]+1`, `row_ptr[i]+2`, ..., `row_ptr[i+1]-1` of array `val`.

In this project, the following data structure should be used to implement the CRS storage format:

```
struct sparse_mat_crs {
  int n, nnz;
  double *val;
  int *row_ptr, *col_idx;
};
```

3.3 Matrix market exchange file format

Matrix market exchange format is commonly used to store a sparse matrix in an ASCII formatted data file. This file format matches with the COO data structure. A data file in the matrix market exchange format is assumed to start with a few lines of comments, each preceded by the `%` symbol. After the comment lines, there will always be a text line containing three integers: the matrix's number of rows (`n`), number of columns (`n`), and number of nonzero values (`nnz`). Thereafter, there will be exactly `nnz` text lines, each containing three values that correspond to one nonzero value in the sparse matrix: the row index, the column index and the nonzero value itself.

The following is an example of a sparse matrix stored in the matrix market exchange file format:[1]

```
%%MatrixMarket matrix coordinate real general
%=================================================================================
%
```

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

```
% Indices are 1-based, i.e. A(1,1) is the first element.
%
%==============================================================================
  5  5  8
    1     1   1.000e+00
    2     2   1.050e+01
    3     3   1.500e-02
    1     4   6.000e+00
    4     2   2.505e+02
    4     4  -2.800e+02
    4     5   3.332e+01
    5     5   1.200e+01
```

The corresponding sparse matrix is

$$\begin{bmatrix} 1 & 0 & 0 & 6 & 0 \\ 0 & 10.5 & 0 & 0 & 0 \\ 0 & 0 & 0.015 & 0 & 0 \\ 0 & 250.5 & 0 & -280 & 33.32 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}$$

Some realistic sparse matrices in the matrix market exchange format can be found at the following open collection: http://sparse.tamu.edu/HB. Note: Some of the matrices there are rectangular (not of square shape). Moreover, most of the square-shaped matrices in the collection are meant to be symmetric, but it is OK to only use the lower triangular part of these matrices, as stored in the data files.

4 Required work of the project

4.1 Four serial functions

The following serial functions should be implemented:

1. `void read_sparse_matrix_from_file (char *filename, struct sparse_mat_coo *matrix);` This function should read a sparse matrix from a data file in the matrix market exchange file format (the name of the file is given in argument `filename`). The read-in sparse matrix is to be stored in a COO structure (such that the structure's values of `n` and `nnz` are updated, its arrays of `row_idx`, `col_idx` and `val` are allocated with the correct length `nnz`, and then filled with values that are read from the data file).

2. `void sampled_matrix_multiplication_coo (struct sparse_mat_coo *C, double **A, double **B, struct sparse_mat_coo *S);` This function should compute the sampled dense matrix multiplication: $C = (A \times B) \odot S$ as defined in (2), with sparse matrices C and S stored in the COO format, while dense matrices A and B are stored as row-major 2D arrays.

3. `void translate_coo_to_crs (struct sparse_mat_coo *mat_coo, struct sparse_mat_crs *mat_crs);` This function should, based on an existing COO data structure of a sparse matrix, fill correct values in arrays `row_ptr`, `col_idx` and `val` of the equivalent CRS data structure.

4. `void sampled_matrix_multiplication_crs (struct sparse_mat_crs *C, double **A, double **B, struct sparse_mat_crs *S);` This function should also compute the sampled dense matrix multiplication: $C = (A \times B) \odot S$, now with sparse matrices C and S in the CRS format.

Each serial function should be placed in a separate file with the same name. (For example, the `read_sparse_matrix_from_file` function is contained in a file named `read_sparse_matrix_from_file.c`.)

4.2 The `main` program

The following is a skeleton of the `main` program (placed in a file named `main.c`) that needs to be completed by the student:

```
int main (int nargs, char **args)
{
  char filename[80];
  double **A, **B;
  struct sparse_mat_coo S_coo, C_coo;
  struct sparse_mat_crs S_crs, C_crs;

  // read from command line the name of a data file in the matrix market exchange format
  // ....

  // read a sparse matrix from the matrix market data file, and store it in S_coo
  read_sparse_matrix_from_file (filename, &S_coo);

  // create the corresponding COO data structure C_coo
  C_coo.n = S_coo.n; C_coo.nnz = S_coo.nnz;
  C_coo.row_idx = S_coo.row_idx;
  C_coo.col_idx = S_coo.col_idx;
  C_coo.val = (double*)malloc(C_coo.nnz * sizeof(double));

  // allocate A and B as 2D n x n arrays and assign suitable numerical values
  // ....

  // computation 1
  sampled_matrix_multiplication_coo (&C_coo, A, B, &S_coo);

  // allocate CRS data structure S_crs
  S_crs.n = S_coo.n; S_crs.nnz = S_coo.nnz;
  S_crs.row_ptr = (int*)malloc((S_crs.n+1) * sizeof(int));
  S_crs.col_idx = (int*)malloc(S_crs.nnz * sizeof(int));
  S_crs.val = (double*)malloc(S_crs.nnz * sizeof(double));

  // translate from S_coo to S_crs
  translate_coo_to_crs (&S_coo, &S_crs);

  // allocate CRS data structure C_crs
  C_crs.n = S_crs.n; C_crs.nnz = S_crs.nnz;
  C_crs.row_ptr = S_crs.row_ptr;
  C_crs.col_idx = S_crs.col_idx;
  C_crs.val = (double*)malloc(C_crs.nnz * sizeof(double));

  // computation 2
  sampled_matrix_multiplication_crs (&C_crs, A, B, &S_crs);

  // deallocate data structures
  // .....

  return 0;
}
```

4.3 OpenMP parallelization

The two `sampled_matrix_multiplication` functions should be parallelized by adding suitable OpenMP directives. (The other two functions don't need to be parallelized.) Note: It is possible to have both

the serial and parallel versions in the same code, with help of `#ifdef _OPENMP ... #endif` when needed.

4.4 Submission

Each student should submit, via Devilry, a tarball (`.tar`) or a zip file (`.zip`). Upon unpacking/unzipping it should produce a folder named `IN3200_Oblig1_xxx` or `IN4200_Oblig1_xxx`, where `xxx` should be the username of the student.

Inside the folder, there should be *at least* the following files:

- `read_sparse_matrix_from_file.c`
- `sampled_matrix_multiplication_coo.c`
- `translate_coo_to_crs.c`
- `sampled_matrix_multiplication_crs.c`
- `function_declarations.h` (This header file should contain the declarations of all the four functions, as well as the definition of `struct sparse_mat_coo` and `struct sparse_mat_crs`.)
- `main.c`
- `README.txt` (information about how to compile the code, and how to run the executable) or `Makefile`.
- `oblig1_report_xxx.pdf` (A 1-page short report.)

The short report should contain some time measurements of the serial and parallel executions, using a few different sparse matrices S. IN4200 students should also include in the report a short discussion about the possible reasons for different time usages due to the COO and CRS data formats.

Hint

If you do not have access to an OpenMP-capable C compiler, please consider using the "Fox" system that is provided on EduCloud at UiO. More info can be found at
https://www.uio.no/studier/emner/matnat/ifi/IN3200/v26/teaching-material/advice_to_students.html.