CMOR 420/520, Homework #2: LaTeX Submission

bb118

October 19, 2025

1 Directory organization

The below represents the organization of the main code files for the assignment. The submitted directory has more files than just those listed below, but the other files are not relevant for compilation of any of the assigned `.c` files.

```
homework-2/
├── include/
│   └── MyMatrix.h
├── src/
│   └── MyMatrix.c
├── test_matmul.c
├── time_matmul.c
└── test_reshape.c
```

2 Compilation and Running

There are 3 scripts that compile to executable files within the project: `test_matmul.c`, `time_matmul.c`, and `test_reshape.c`. The positions of these files within the main directory is captured in the organization tree in §1. The compilation commands for each of these files respectively are as follows:

1. (Compilation of `test_matmul.c`) To get an executable named `test_matmul` paste and execute the following into the terminal:

   ```
   gcc -I./include src/MyMatrix.c test_matmul.c -o test_matmul
   ```

   To run the resulting executable, paste and execute the following into the terminal:

   ```
   ./test_matmul
   ```

2. (Compilation of `time_matmul.c`) To get an executable named `time_matmul` paste and execute the following into the terminal:

   ```
   gcc -I./include src/MyMatrix.c time_matmul.c -o time_matmul
   ```

   If desiring to compile with `O0` optimization, paste and execute instead the following to obtain an executable named `time_matmul_O0`:

```
gcc -O0 -I./include src/MyMatrix.c time_matmul.c -o time_matmul_O0
```

If desiring to compile with O3 optimization, paste and execute instead the following to obtain an executable named time_matmul_O3:

```
gcc -O3 -I./include src/MyMatrix.c time_matmul.c -o time_matmul_O3
```

To run the executable resulting from each of these compilations, paste and execute the following into the terminal, respectively:

```
./time_matmul
./time_matmul_O0
./time_matmul_O3
```

3. (Compilation of test_reshape.c) To get an executable named test_reshape, paste and execute the following into the terminal:

```
gcc -I./include src/MyMatrix.c test_reshape.c -o test_reshape
```

To run the resulting executable, paste and execute the following into the terminal:

```
./test_reshape
```

3 Verification / Output

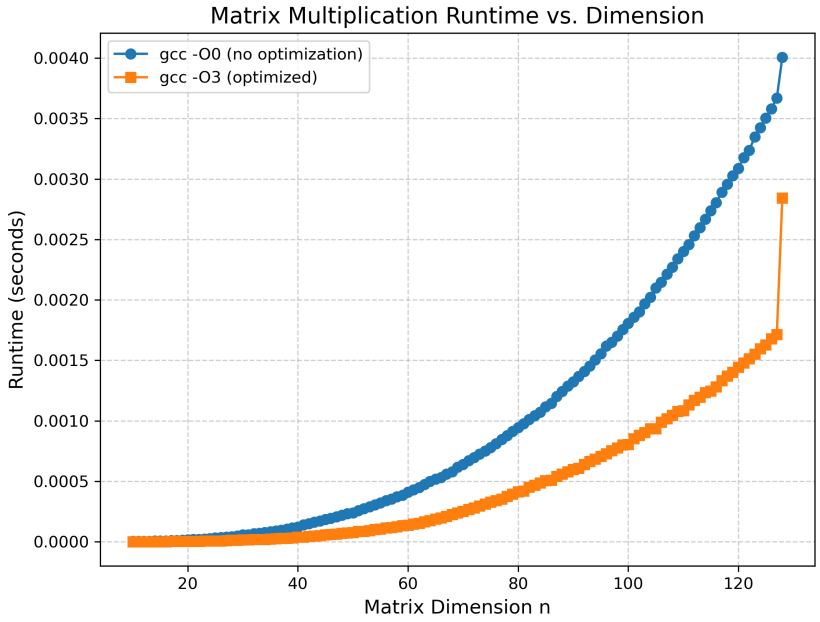
The output of the executable obtained by compiling the test_matmul.c script is:

```
The maximum absolute difference between the true matrix product
and the computed matrix product is 0.000000000
```

Within the code there is also a commented out part that also prints the matrix resulting from the product. If that code is un-commented, the output becomes:

```
    C = A * B =
70.000000000 80.000000000 90.000000000
158.000000000 184.000000000 210.000000000
246.000000000 288.000000000 330.000000000
The maximum absolute difference between the true matrix product
and the computed matrix product is 0.000000000
```

The output of the executable obtained by compiling the time_matmul.c script is matrix_runtimes.png, which is included in the docs folder within the zipped directory. This image is printed here:



The data that was used to make this image with `plot_runtimes.py` is also included in the main directory. These are the files `runtimes_O0.txt` and `runtimes_O3.txt`.

The output of the executable obtained by compiling the `test_reshape.c` is

```
The matrix before reshaping is:
   1.000    2.000    3.000    4.000
   5.000    6.000    7.000    8.000
   9.000   10.000   11.000   12.000
The matrix after reshaping is:
   1.000    2.000    3.000
   4.000    5.000    6.000
   7.000    8.000    9.000
  10.000   11.000   12.000
```

4 Questions

1. Let A and B denote the matrices in question, and suppose we are talking about appending B to A (either vertically or horizontally). Appending B to A vertically would be easier (than horizontally) because this corresponds to the rows of B being appended to the rows of A. Thus, the contiguous list of rows (for the `MyMatrix` object this would be `A.data`) of B can be appended to the contiguous list of rows of A, which is simple. Meanwhile, to append horizontally we need to edit each row of A, which in particular entails moving the values of the contiguous data around and inserting the values of B in the middle of this contiguous data.

2. Yes we could define a contiguous column major format for an $m \times n$ matrix. The important thing to keep in mind is that now the first index of `MyMatrix.data` is now a column index instead of a row index. But if that is ok with the user/program context, then basically the same logic holds: we allocate a contiguous block of memory of length $n *$ `sizeof(double*)` for a sequence of pointers (each one now representing the beginning of a *column*), we assign a contiguous block of memory of length

$n*m*$`sizeof(double)` for the elements of the matrix, we set `MyMatrix.data[0]` to point to the start of the contiguous block of elements of the matrix, and then we iteratively assign each pointer in the contiguous array of pointers to point to `MyMatrix.data[0]+i*`n (notice how we now increment by n instead of m).

5 Additional CMOR520 Tasks

As mentioned in §2, to get an executable named `test_reshape`, paste and execute the following into the terminal:

```
gcc -I./include src/MyMatrix.c test_reshape.c -o test_reshape
```

To run the test of the reshape function, first compile using the command above, and then paste and execute the following in your terminal:

```
./test_reshape
```

As mentioned in §3, the output of `test_reshape.c` is

```
The matrix before reshaping is:
   1.000    2.000    3.000    4.000
   5.000    6.000    7.000    8.000
   9.000   10.000   11.000   12.000
The matrix after reshaping is:
   1.000    2.000    3.000
   4.000    5.000    6.000
   7.000    8.000    9.000
  10.000   11.000   12.000
```