CMOR 421/521 Assignment: Matrix Transpose and Multiplication

Yuhao Liu

February 14, 2025

Contents

1 Directory Structure

Below is my file organization for this assignment. My final zip file follows this structure (driver files in the top-level directory, `docs/` for LaTeX, `src/` for source files, and `include/` for header files):

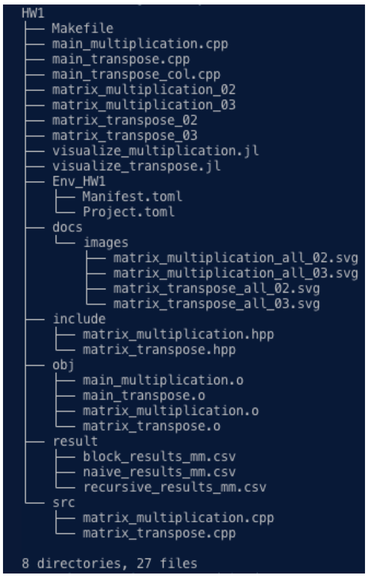

Figure 1: structure

- The `main_mutiplication.cpp` and `main_transpose.cpp` are include `main` function for matrix transpose and matrix multiplication in naive method, cache-block method, and recursive method.
- The `main_mutiplication_col.cpp` is used to test when A^T is stored in column major format, the performance for different methods.
- The `visualize_mutiplixation.jl` and `visualize_transpose.jl` are Julia code for plot the figures.
- The folder `Env_HW1` are the Julia project local environment. You need to use the following command in terminal for execute these files. First,

  ```
  cd ./HW1
  ```

  and using `]` to enter the environment space, and then using command `activate Env_HW1` to load the local environment. You can also use `st` to check what library I used.
- The folder `docs/` is to store the Latex file and images.
- The folder `include/` is the place for `.hpp` files. The `matrix_transpose.hpp` and `matrix_multiplica` are in there.
- The folder `src/` is the place for `matrix_transpose.cpp` and `matrix_multiplication.cpp` which are used to implemented all different methods for matrix transpose, matrix multiplication, and the timing analysis functions.

2 How to Build and Run the Code

- **Build Instructions:**
  - For matrix transpose, you can use

    ```
    make matrix_transpose_02
    ```

    and

    ```
    make matrix_transpose_03
    ```

    to compile the program with `-O2` or `-O3` optimization flags.
  - For matrix multiplication, you can use

    ```
    make matrix_multiplication_02
    ```

    and

    ```
    make matrix_multiplication_03
    ```

    to compile the program with `-O2` or `-O3` optimization flags.
  - For the matrix transpose test when A^T is stored column major, you can use

    ```
    g++ -std=c++17 -O3 main_transpose_col.cpp -o main_transp
    ```

    to compile.
- **Running Instructions:**

  Once you build the execution file, it will appear as driver files. You can use the following command to run the code:

  ```
  ./matrix_transpose_02
  ./matrix_transpose_03
  ./matrix_multiplication_02
  ./matrix_multiplication_03
  ./main_transpose_col
  ```

- **Execution:** When you execute the program, in the terminal you can see the output like the following figure:

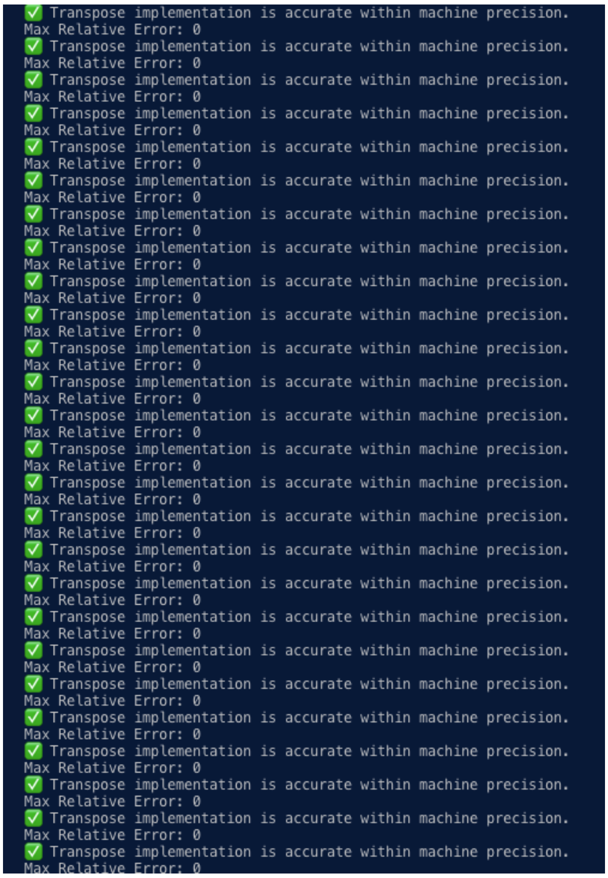

Figure 2: Method accuracy

This will check that the relative error for each matrix transpose implementation is zero up to machine precision.

After you run one of above command, you will see there are two new folders `obj/` and `result/`. The first on is the place for `.o` file and the second one is the results in three different `.csv` files, `naive_results.csv`, `block_results.csv`, and `recursive_results.csv`.

If you want to clean all of them, using

```
make clean
```

and this command will clean folder `obj/` and `result/`.

3 Analysis

In this section, I will show you the analysis for matrix transpose and matrix multiplication.

3.1 Matrix transpose

For analysis the efficiency among naive matrix transpose, cache-block matrix transpose, and recursive matrix transpose, you mainly use the file `main_transpose.cpp`, `matrix_transpose.cpp` in folder `src/`, and `matrix_transpose.hpp` in folder `include/`.

All my tests used the following setting:

```
int BLOCK_SIZE = 16;
int threshold = 16;
vector<int> sizes = {32, 64, 128, 256, 512, 1024, 2048};
vector<int> block_sizes = {8, 16, 32, 64, 128};
vector<int> thresholds = {8, 16, 32, 64, 128};
```

- You can change `size` for different matrix.
- You can change `BLOCK_SIZE` for different block size used in cache-block matrix transpose method.
- You can change `threshold`, for terminating that the matrix is smaller than these threshold sizes.

The following pictures are with `-O2` and `-O3` optimization flags.

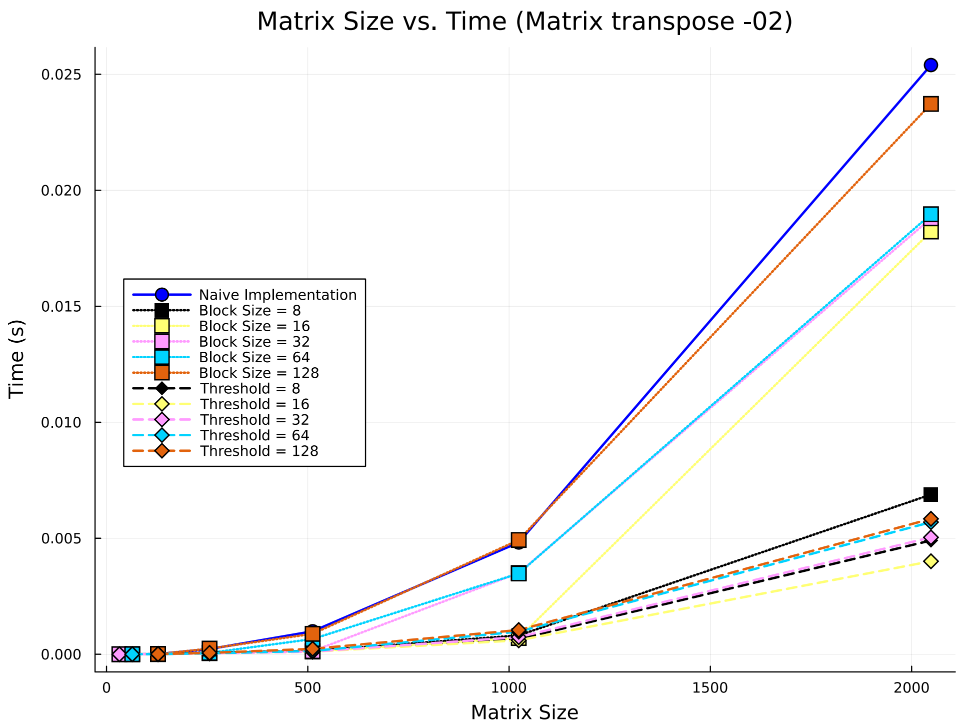

Figure 3: Matrix Transpose (-O2)

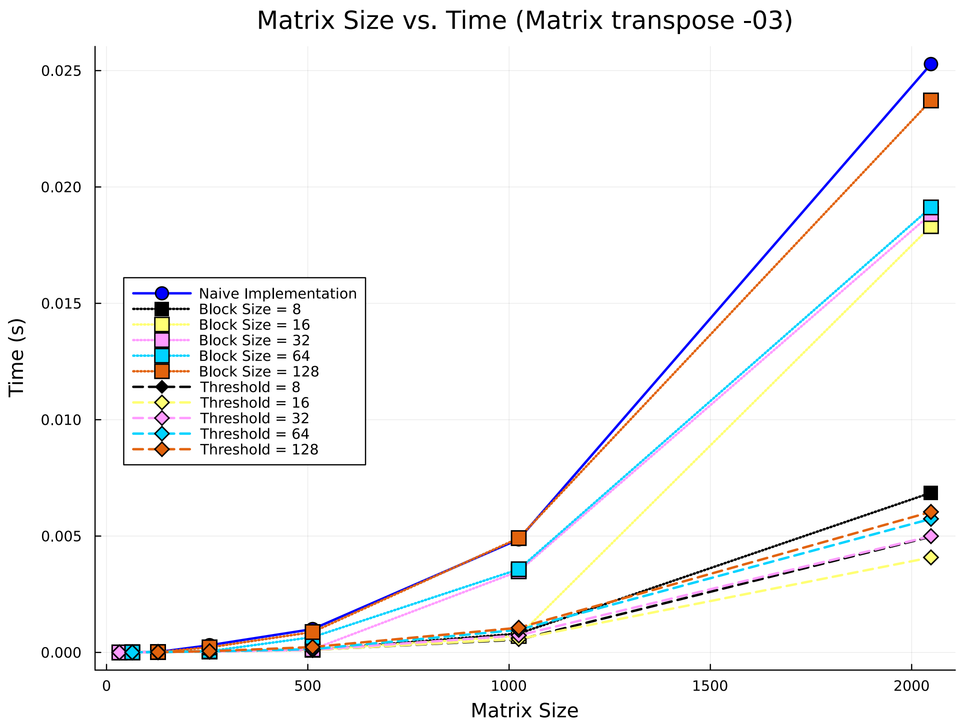

Figure 4: Matrix Transpose (-O3)

According to these figures, we could find that when matrix size become larger and larger, the recursive method gets the best performance. For the cache-blocked method with 2048×2048 matrix, the recursive threshold size with 16 get the best performance.

For the cache-block method, we can see it is faster than naive version but slower than recursive when matrix is large enough. When the block size is 8, this method have the best performace.

3.1.1 Analysis for slow memory used for naive matrix transposition

For an $N \times N$ matrix, the naive transpose algorithm reads each of the N^2 elements. The naive method is:

```
// -----------------------------------------------
// Naive Transposition
// -----------------------------------------------
void transpose_naive(const int n, double * AT, double * A) {
    for (int i = 0; i < n; ++i){
        for (int j = 0; j < n; ++j){
            AT[j * n + i] = A[i * n + j];
        }
    }
}
```

- Read `A[i][j]` from slow memory, which need N^2 reads
- Write `B[j][i]` to slow memory, which need N^2 writes

Then, the total means $2N^2$ slow memory accesses in the naive algorithm.

3.1.2 A^T store in column major

Since, we know that for 2048×2048 matrix, cache-block method reach the best performance at 8 block size and recursive method reach the best performance at 16 threshold size. So If A^T stored in column major, my test is under these situation, and the result is as following:

```
Naive transpose time: 0.191186 s
Max Relative Error: 0
Transpose implementation is accurate within machine precision.
Blocked transpose time: 0.018329 s
Max Relative Error: 0
Transpose implementation is accurate within machine precision.
Recursive transpose time: 0.018035 s
```

The recursive transpose still hold the best performance even the cache-block method has very closed time.

3.2 (For CMOR 521) Matrix-Matrix Multiplication

For analysis the efficiency among naive matrix transpose, cache-block matrix transpose, and recursive matrix transpose, you mainly use the file `main_multiplication.cpp`, `matrix_multiplication.cpp` in folder `src/`, and `matrix_multiplication.hpp` in folder `include/`.

All my tests used the following setting:

```
int BLOCK_SIZE = 16;
int threshold = 16;
vector<int> sizes = {32, 64, 128, 256, 512, 1024, 2048};
vector<int> block_sizes = {8, 16, 32, 64, 128};
vector<int> thresholds = {8, 16, 32, 64, 128};
```

- You can change `size` for different matrix.
- You can change `BLOCK_SIZE` for different block size used in cache-block matrix-matrix multiplication method.
- You can change `threshold`, for terminating that the matrix is smaller than these threshold sizes.

The following pictures are with `-O2` and `-O3` optimization flags.

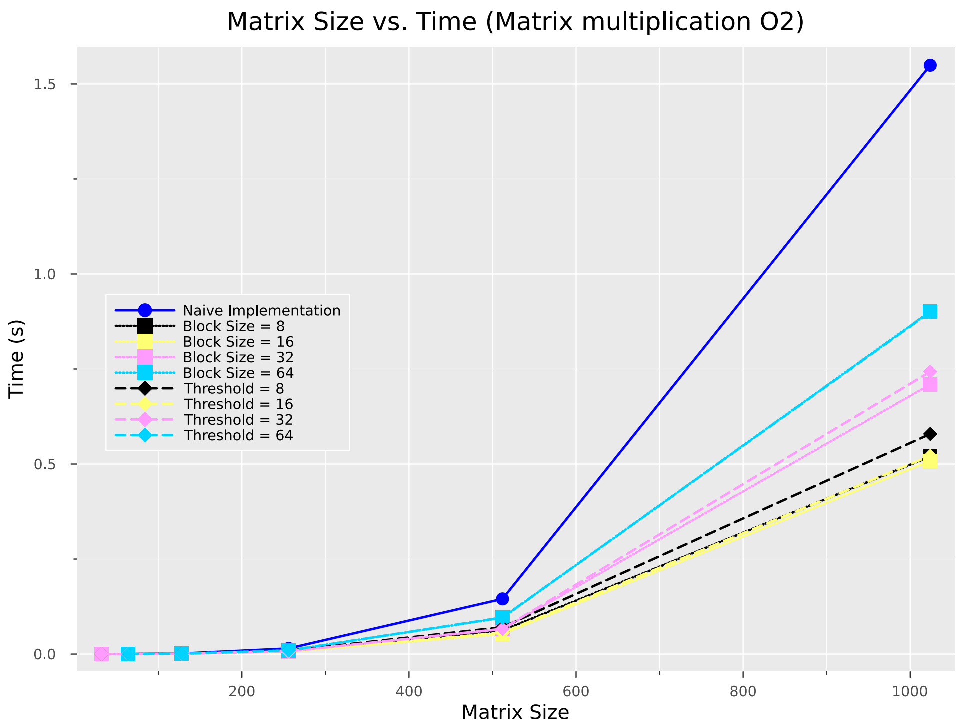

Figure 5: Matrix multiplication (-O2)

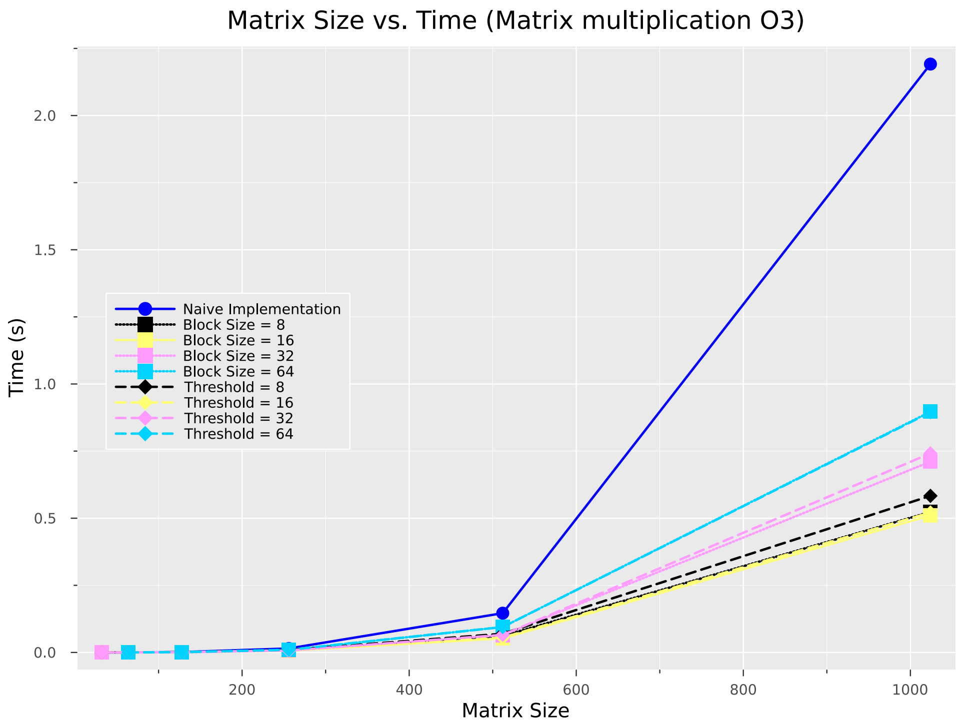

Figure 6: Matrix multiplication (-O3)

According to these figures, we could find that when matrix size become larger and larger, the recursive method gets the best performance. For the cache-blocked method with 2048×2048 matrix, both of the recursive method with 16threshold size and cache-block method with 16 block size get the best performance. The time for the are really closed. Both cache-block method and recursive method are better than naive method.