Project 3: Improved Matrix Multiplication in C

12113053 THA Sreyny

1. Introduction

Matrix multiplication, a very basic operation in linear algebra, plays an important role in deep learning. Please implement matrix multiplication in C and try your best to improve its speed.

2. Methods

In this project, I have implemented follwing methods:

- matmul_plain(): a brute force matrix multiplication
- matmul_swapJK(): this method just change the position of j and k
- matmul_block(): this method divides matrix into blocks for friendly cache use
- matmul_improved(): using SIMD+OpenMp
- matmul_blas(): using blas library

Below the brief ideas of each methods:

matmul_plain():

```
Matrix *matmul_plain(const Matrix *mat1, const Matrix *mat2) {
    if (mat1->cols != mat2->rows) {
        fprintf(stderr, "Invalid matrix dimensions for multiplication\n");
        return NULL;
    }

    Matrix *result = create_matrix(mat1->rows, mat2->cols);

    for (size_t i = 0; i < mat1->rows; i++) {
        for (size_t j = 0; j < mat2->cols; j++) {
            float sum = 0.0f;
            for (size_t k = 0; k < mat1->cols; k++) {
                sum += mat1->data[i * mat1->cols + k] * mat2->data[k * mat2->cols
+ j];
            }
            result->data[i * result->cols + j] = sum;
        }
    }

    return result;
}
```

matmul_swapJK():

```c
Matrix *matmul_swapJK(const Matrix *mat1, const Matrix *mat2) {
    if (mat1->cols != mat2->rows) {
        fprintf(stderr, "Invalid matrix dimensions for multiplication\n");
        return NULL;
    }

    Matrix *result = create_matrix(mat1->rows, mat2->cols);
    size_t j;
    for (size_t i = 0; i < mat1->rows; i++) {
        for (size_t k = 0; k < mat2->cols; k++) {
            float sum = 0.0f;
            for (j = 0; j < mat1->cols; j++) {
                sum += mat1->data[i * mat1->cols + k] * mat2->data[k * mat2->cols + j];
            }
            result->data[i * result->cols + j] = sum;
        }
    }

    return result;
}
```

matmul_block():

```c
Matrix *matmul_block(const Matrix *mat1, const Matrix *mat2) {
    if (mat1->cols != mat2->rows) {
        fprintf(stderr, "Invalid matrix dimensions for multiplication\n");
        return NULL;
    }
    size_t block_size=32;
    size_t rows = mat1->rows;
    size_t cols = mat2->cols;
    size_t common_dim = mat1->cols;

    Matrix *result = create_matrix(rows, cols);

    for (size_t ii = 0; ii < rows; ii += block_size) {
        for (size_t jj = 0; jj < cols; jj += block_size) {
            for (size_t kk = 0; kk < common_dim; kk += block_size) {
                for (size_t i = ii; i < ii + block_size && i < rows; i++) {
                    for (size_t j = jj; j < jj + block_size && j < cols; j++) {
                        for (size_t k = kk; k < kk + block_size && k < common_dim; k++) {
                            result->data[i * cols + j] += mat1->data[i * common_dim + k] * mat2->data[k * cols + j];
                        }
                    }
                }
            }
        }
    }
    return result;
}
```

matmul_improved():

```
Matrix *matmul_improved(const Matrix *mat1, const Matrix *mat2) {
    if (mat1->cols != mat2->rows) {
        fprintf(stderr, "Invalid matrix dimensions for multiplication\n");
        return NULL;
    }

    Matrix *result = create_matrix(mat1->rows, mat2->cols);

#pragma omp parallel for
    for (size_t i = 0; i < mat1->rows; i++) {
        for (size_t j = 0; j < mat2->cols; j++) {
            __m256 sum = _mm256_setzero_ps();
            for (size_t k = 0; k < mat1->cols; k += 8) {
                __m256 a = _mm256_loadu_ps(&mat1->data[i * mat1->cols + k]);
                __m256 b = _mm256_loadu_ps(&mat2->data[k * mat2->cols + j]);
                sum = _mm256_add_ps(sum, _mm256_mul_ps(a, b));
            }
            float tmp[8];
            _mm256_storeu_ps(tmp, sum);
            float total = 0.0f;
            for (size_t k = 0; k < 8; k++) {
                total += tmp[k];
            }
            for (size_t k = mat1->cols - mat1->cols % 8; k < mat1->cols; k++) {
                total += mat1->data[i * mat1->cols + k] * mat2->data[k * mat2->cols + j];
            }
            result->data[i * result->cols + j] = total;
        }
    }
    return result;
}
```

matmul_blas():

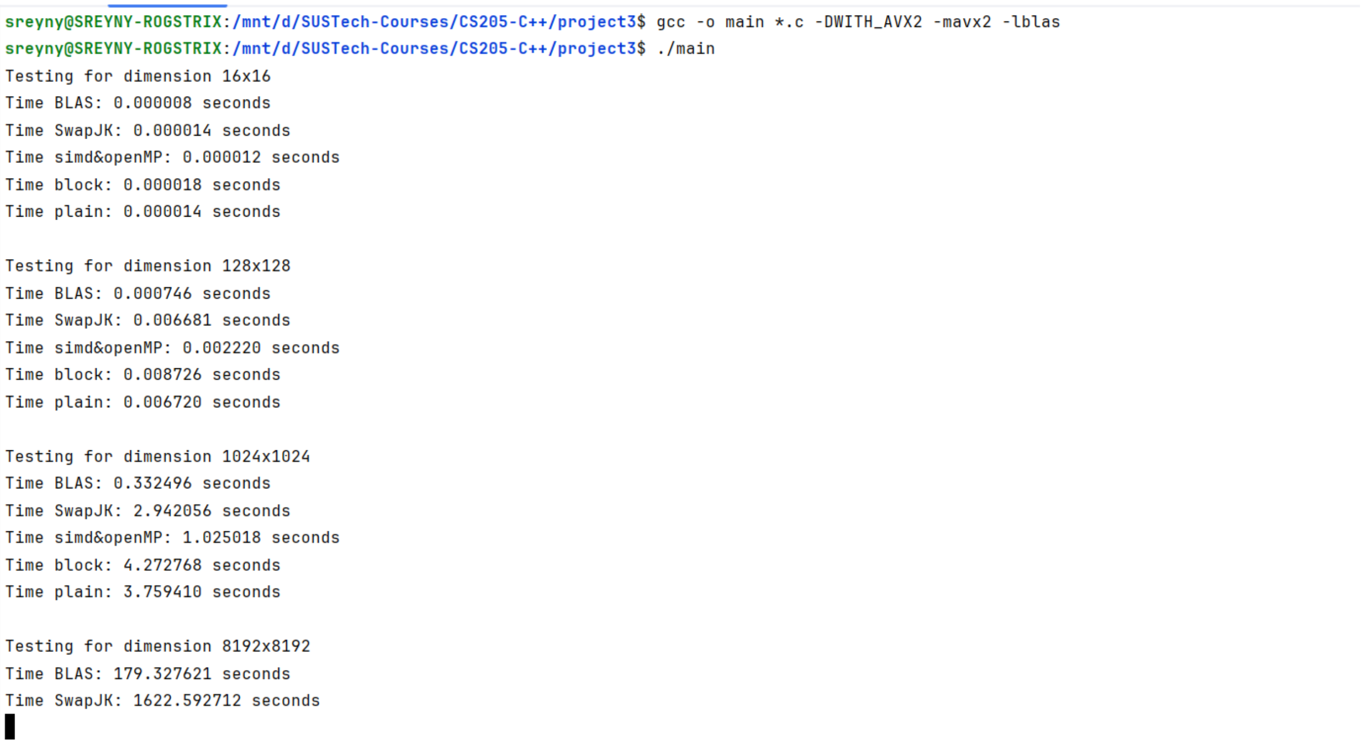
```
Matrix *matmul_blas(const Matrix *mat1, const Matrix *mat2) {
    if (mat1->cols != mat2->rows) {
        fprintf(stderr, "Invalid matrix dimensions for multiplication\n");
        return NULL;
    }
    Matrix *result = create_matrix(mat1->rows, mat2->cols);

    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, mat1->rows, mat2->cols,
mat1->cols,
                    1.0f, mat1->data, mat1->cols, mat2->data, mat2->cols, 0.0f,
result->data, result->cols);

     return result;
}
```

3. Time Execution Without option O3

Without optimizaion flag O3, the result provided as below:



```
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/CS205-C++/project3$ gcc -o main *.c -DWITH_AVX2 -mavx2 -lblas
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/CS205-C++/project3$ ./main
Testing for dimension 16x16
Time BLAS: 0.000008 seconds
Time SwapJK: 0.000014 seconds
Time simd&openMP: 0.000012 seconds
Time block: 0.000018 seconds
Time plain: 0.000014 seconds

Testing for dimension 128x128
Time BLAS: 0.000746 seconds
Time SwapJK: 0.006681 seconds
Time simd&openMP: 0.002220 seconds
Time block: 0.008726 seconds
Time plain: 0.006720 seconds

Testing for dimension 1024x1024
Time BLAS: 0.332496 seconds
Time SwapJK: 2.942056 seconds
Time simd&openMP: 1.025018 seconds
Time block: 4.272768 seconds
Time plain: 3.759410 seconds

Testing for dimension 8192x8192
Time BLAS: 179.327621 seconds
Time SwapJK: 1622.592712 seconds
```

for small size of matrices 16x16, 128x128

- plain methods and swapJK methods take almost same time to compile
- block methods a bit longer time than plain and swapJK methods
- for SIMD&OpenMP, can improve the speed significantly
- Blas has the fastest speed

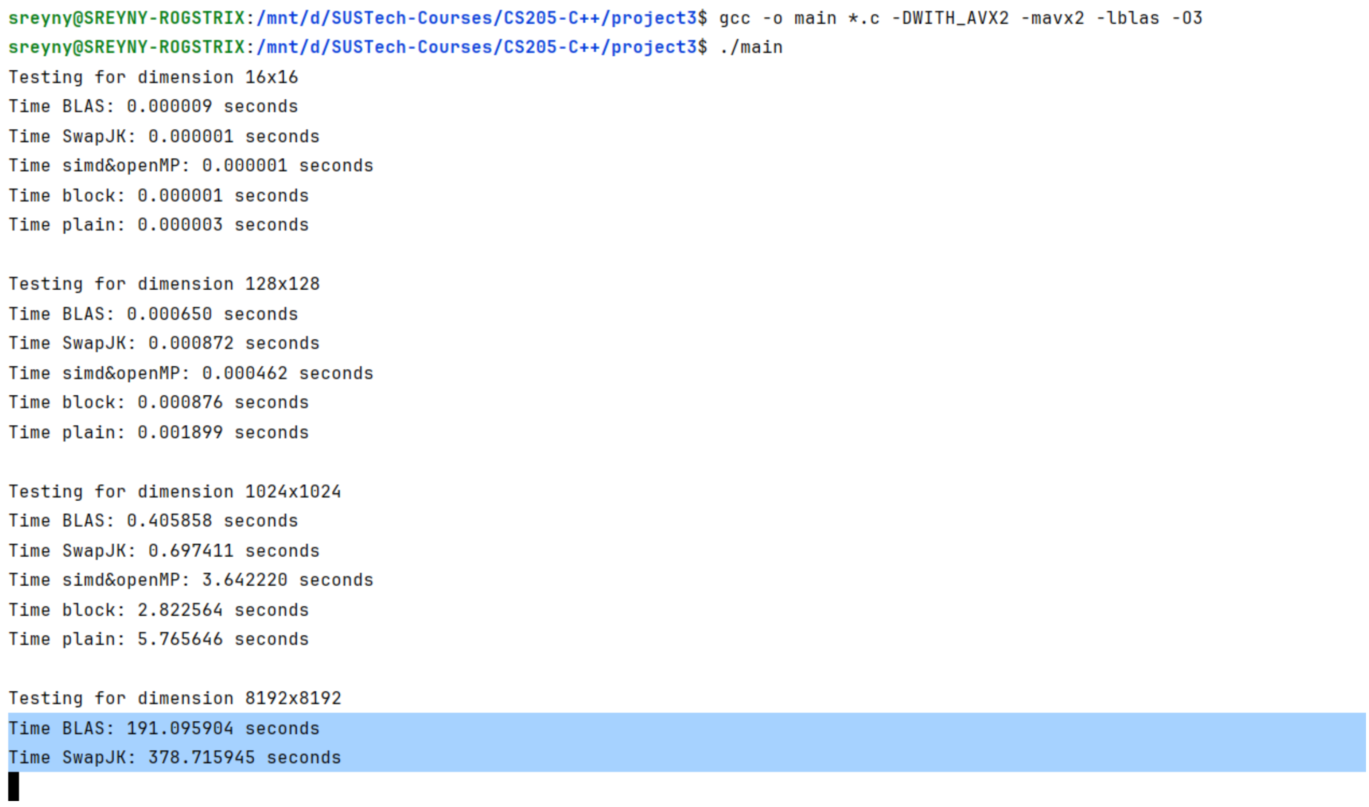
for large size of mattrices 1024*1024,...

- swapJK methods tends to be faster than plain and block methods
- Blas is still the best

4. Time Execution With option O3

As the size of matrices getting larger

- The fastest one is still Blas
- but swapJK is faster than block , block is faster than SIMD&OpenMP
- For 64Kx64K, I got memory allocation fail

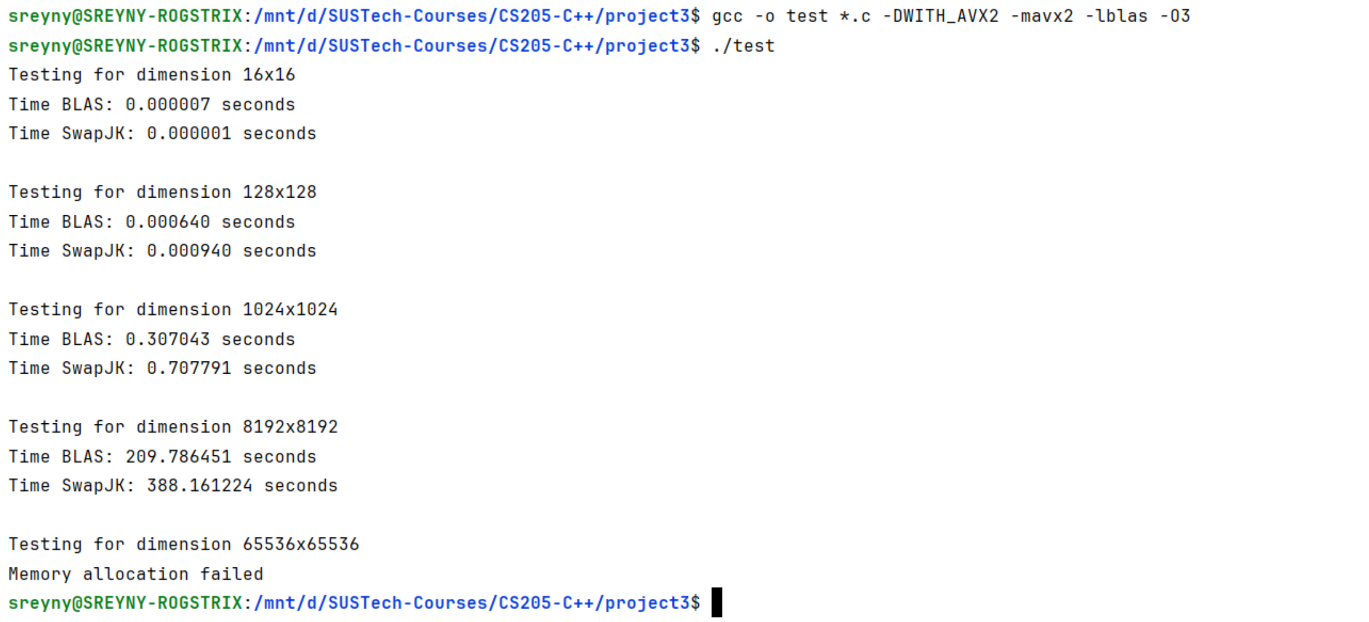


```
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/CS205-C++/project3$ gcc -o main *.c -DWITH_AVX2 -mavx2 -lblas -O3
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/CS205-C++/project3$ ./main
Testing for dimension 16x16
Time BLAS: 0.000009 seconds
Time SwapJK: 0.000001 seconds
Time simd&openMP: 0.000001 seconds
Time block: 0.000001 seconds
Time plain: 0.000003 seconds

Testing for dimension 128x128
Time BLAS: 0.000650 seconds
Time SwapJK: 0.000872 seconds
Time simd&openMP: 0.000462 seconds
Time block: 0.000876 seconds
Time plain: 0.001899 seconds

Testing for dimension 1024x1024
Time BLAS: 0.405858 seconds
Time SwapJK: 0.697411 seconds
Time simd&openMP: 3.642220 seconds
Time block: 2.822564 seconds
Time plain: 5.765646 seconds

Testing for dimension 8192x8192
Time BLAS: 191.095904 seconds
Time SwapJK: 378.715945 seconds
```



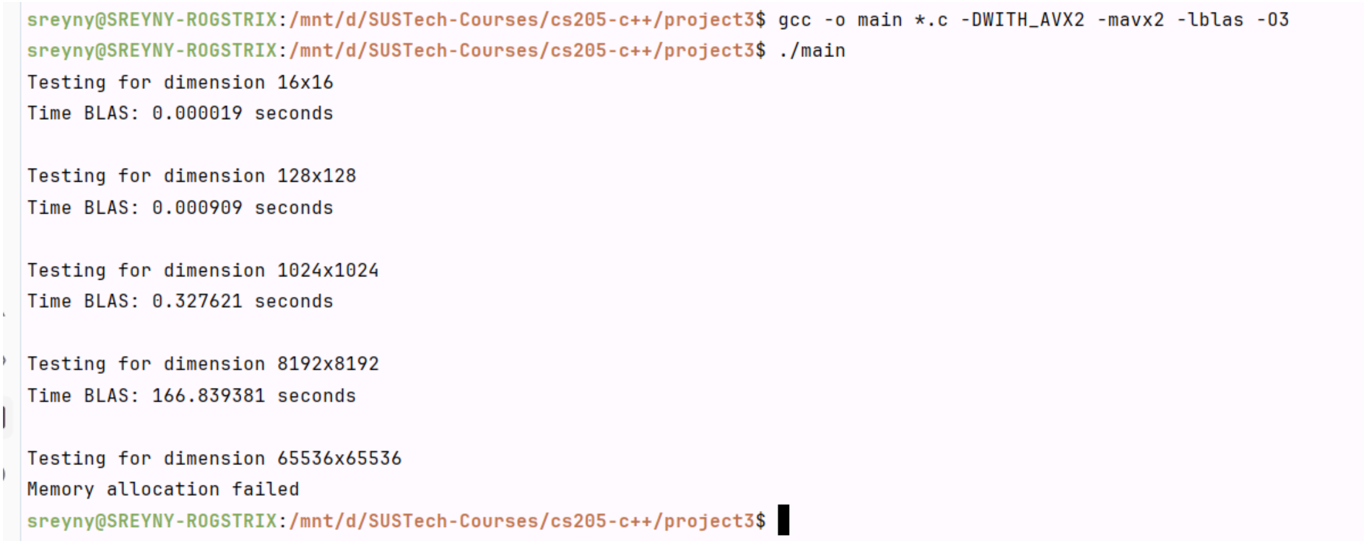
```
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/CS205-C++/project3$ gcc -o test *.c -DWITH_AVX2 -mavx2 -lblas -O3
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/CS205-C++/project3$ ./test
Testing for dimension 16x16
Time BLAS: 0.000007 seconds
Time SwapJK: 0.000001 seconds

Testing for dimension 128x128
Time BLAS: 0.000640 seconds
Time SwapJK: 0.000940 seconds

Testing for dimension 1024x1024
Time BLAS: 0.307043 seconds
Time SwapJK: 0.707791 seconds

Testing for dimension 8192x8192
Time BLAS: 209.786451 seconds
Time SwapJK: 388.161224 seconds

Testing for dimension 65536x65536
Memory allocation failed
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/CS205-C++/project3$
```



```
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/cs205-c++/project3$ gcc -o main *.c -DWITH_AVX2 -mavx2 -lblas -O3
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/cs205-c++/project3$ ./main
Testing for dimension 16x16
Time BLAS: 0.000019 seconds

Testing for dimension 128x128
Time BLAS: 0.000909 seconds

Testing for dimension 1024x1024
Time BLAS: 0.327621 seconds

Testing for dimension 8192x8192
Time BLAS: 166.839381 seconds

Testing for dimension 65536x65536
Memory allocation failed
sreyny@SREYNY-ROGSTRIX:/mnt/d/SUSTech-Courses/cs205-c++/project3$
```

FYI

For Source Code Github.

Use following command line to compile the code:

This command line include AVX2 and Blas library

```
gcc -o main *.c -DWITH_AVX2 -mavx2 -lblas
```

This command line include AVX2, Blas library and Optimization flag -O3

```
gcc -o main *.c -DWITH_AVX2 -mavx2 -lblas -O3
```

Reference

1. Improving the performance of Matrix Multiplication stackoverflow.
2. Matrix Multiplication: Optimizing the code from 6 hours to 1 sec (2021). Meduim.