C/C++ Programming Project 1 Report

Name: Longqing Chen SID: 22010057

Index

1. Task 1 and 2: Using `struct` for matrices and compute matrix multiplication for two matrices

a) Codes and explanations

```
#include <iostream>
using namespace std;

struct matrix{
    int r;
    int c;
    float mat[10][10];
}A,B,C;

int main(){
    cout << "Enter the number of rows and columns for A: \n";
    cin >> A.r >> A.c;
    cout << "Enter the number of rows and columns for B: \n";
    cin >> B.r >> B.c;

    if (A.c != B.r)
    {
        cout << "The number of rows of B must equal the number of co
lumns of A." << endl;
        exit(1);
    }
```

Creates a `struct` containing the number of rows and columns of matrices, and a 2-dimensional array to store data. Creates structs A, B, and C at the end.

Checks and exits the program if the column number of A does not equal the row number of B.

```
    for (int i=0;i<A.r;i++)
    {
        cout << "Enter the #" << i+1 << " row of matrix A: " << endl
;
        for (int j=0;j<A.c;j++)
        {
            cin >> A.mat[i][j];
            if (!cin)
            {
                cout << "Wrong input!" << endl;
                exit(1);
            }
        }
    }
    cout << "Matrix A is: " << endl;

    for(int i=0;i<A.r;i++)
    {
        for(int j=0;j<A.c;j++)
        {
            cout << A.mat[i][j] << "  ";
        }
        cout << endl;
    }

    for (int i=0;i<B.r;i++)
    {
        cout << "Enter the #" << i+1 << " row of matrix B: " << endl
;
        for (int j=0;j<B.c;j++)
        {
            cin >> B.mat[i][j];
            if (!cin)
            {
                cout << "Wrong input!" << endl;
                exit(1);
            }
        }
    }
    cout << "Matrix B is: " << endl;

    for(int i=0;i<B.r;i++)
    {
        for(int j=0;j<B.c;j++)
```

Checks and exits the program if the input does not align with the set conditions, e.g. not a float. The exception is if the inputs are too many, it only takes the first `j` inputs.

```
        {
            cout << B.mat[i][j] << "  ";
        }
        cout << endl;
    }
    for(int i=0;i<A.r;i++)
    {
        for(int j=0;j<B.c;j++)
        {
            for(int k=0;k<A.c;k++)
            {
                C.mat[i][j] += A.mat[i][k] * B.mat[k][j];
            }
        }
    }

    cout << "The result is: " << endl;

    for(int i=0;i<A.r;i++)
    {
        for(int j=0;j<B.c;j++)
        {
            cout << C.mat[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

b) Results and Analysis

i) The correct input

```
Enter the number of rows and columns for A:
2 3
Enter the number of rows and columns for B:
3 4
Enter the #1 row of matrix A:
1.2 2.3 3.4
Enter the #2 row of matrix A:
1.1 2.3 4.2 1.2
Matrix A is:
1.2  2.3  3.4
1.1  2.3  4.2
Enter the #1 row of matrix B:
3 2.3 1.2 4.2
Enter the #2 row of matrix B:
2.3 4 2.
Enter the #3 row of matrix B:
2.3 4.1 2.3
2
Matrix B is:
1.2  3  2.3  1.2
4.2  2.3  4  2
2.3  4.1  2.3  2
The result is:
18.92 22.83 19.78 12.84
20.64 25.81 21.39 14.32
```

ii) Column # of A $\neq$ Row # of B

```
Enter the number of rows and columns for A:
2 3
Enter the number of rows and columns for B:
2 3
The number of rows of B must equal the number of columns of A.
```

iii) Wrong input

```
Enter the number of rows and columns for A:
2 3
Enter the number of rows and columns for B:
3 2
Enter the #1 row of matrix A:
1.2 2.3 4.3
Enter the #2 row of matrix A:
1

d
Wrong input!
```

2. Task 3: Measuring the time it takes to compute multiplication of two matrices each with 200M elements

a) Codes and explanations

```
#include <iostream>
#include <time.h>
#include <chrono>

using namespace std;
float** create_matrix(int rows, int cols);
void destroy_matrix(float** &mat, int r);

int main(){


    float ** matA = create_matrix(20,10000000);
    float ** matB = create_matrix(10000000,20);
    float ** matC = create_matrix(20,20);

    for(int i=0;i<20;i++)
    {
        for (int j=0;j<10000000;j++)
        {
            srand((unsigned)time(NULL));
            matA[i][j]=rand()*1.0/RAND_MAX*(10-1)+1;
        }
    }
```

Uses ** (pointer to pointer) to create three matrices dynamically. The current dimension of the result matrix C is 20*20. Another dimension 2*2 would be analyzed later.

```
    for(int i=0;i<10000000;i++)
    {
        for (int j=0;j<20;j++)
        {
            srand((unsigned)time(NULL));
            matB[i][j]=rand()*1.0/RAND_MAX*(10-1)+1;
        }
    }
```

Randomizes the entries.

```
auto start = chrono::steady_clock::now();
    for(int i=0;i<20;i++)
    {
        for(int j=0;j<20;j++)
        {
            for(int k=0;k<10000000;k++)
            {
                matC[i][j] += matA[i][k] * matB[k][j];
            }
        }
    }
auto end = chrono::steady_clock::now();


cout << "Calculation took: " << chrono::duration_cast< chrono::milli
seconds>(end - start).count() << "ms.";

    destroy_matrix(matA, 20);
    destroy_matrix(matB, 10000000);
    destroy_matrix(matC, 20);

    return 0;
}

float** create_matrix(int r, int c)
{
    float** mat = new float* [r];
    for (int i=0;i<r;i++)
    {
        mat[i] = new float[c]();
    }
    return mat;
}
```

Measures time and performs matrix multiplication using brute-force algorithm.

```
void destroy_matrix(float** &mat, int r)
{
    if (mat)
    {
        for (int i=0;i<r;i++)
        {
            delete[] mat[i];
            delete[] mat;
            mat = nullptr;
        }
    }
}
```

b) Results and Analysis

i) C_{2x2} Brute-force

As the randomization of entries used CPU time as seeds, the resulting entries would be the same.

```
The result is:
1.07374e+09 1.07374e+09
1.07374e+09 1.07374e+09
Calculation took: 1689ms.
```

ii) C_{20x20} Brute-force (Aligns with the codes above)

```
6.21161e+08 6.21161e+08 6.2
161e+08 6.21161e+08 6.21161
6.21161e+08 6.21161e+08 6.2
161e+08 6.21161e+08 6.21161
Calculation took: 72112ms.
```

iii) Other dimensions

Counting additions and multiplications separately, as the number of loops going up with the row numbers going up, the calculation time increases significantly. Thus, the analysis in Task 5 would focus primarily on efficiency improvement of the two mentioned dimensions of C.

3. Task 4 and 5: Efficiency Improvement and Results Comparison with OpenBLAS

a) Changes from Task 3

i) Swap `j` and `k` in multiplication

```
for(int i=0;i<20;i++)
{
    for(int k=0;k<10000000;k++)
    {
        for(int j=0;k<20;j++)
        {
            matC[i][j] += matA[i][k] * matB[k][j];
        }
    }
}
```

The reasoning behind this is that a `k` index on the inner-most loop will cause a cache miss in `matB` on every iteration. With `j` as the inner-most index, both `matC` and `matB` are accessed contiguously, while `matA` stays put.

Results of swapping:

1. C_{2x2}

```
The result is:
1.07374e+09 1.07374e+09
1.07374e+09 1.07374e+09
Calculation took: 1473ms.
```

Efficiency improved by 200ms.

2. C_{20x20}

```
5.52284e+08 5.52284e+08 5.5
284e+08 5.52284e+08 5.52284
+08 5.52284e+08 5.52284e+08
5.52284e+08
Calculation took: 18990ms.
```

Efficiency improved by 54000ms.

ii) Blocking algorithm

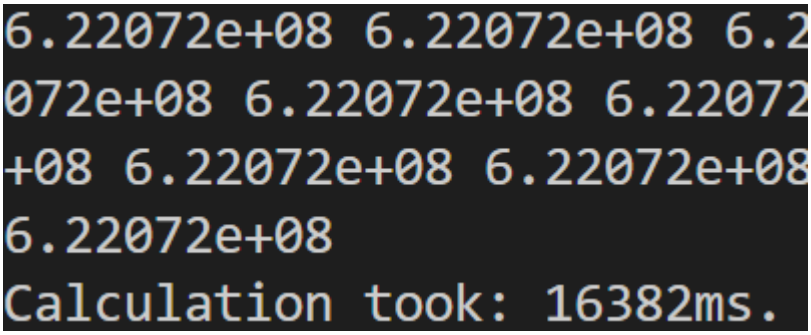
```
for(int i=0;i<20;i+=20)
    for(int k=0;k<10000000;k+=20)
        for(int j=0;j<20;j+=20)
            for(int i1 = 0; i1 < 20;i1++)
                for(int k1 = 0; k1 < 20;k1++)
```

```
                        for(int j1 = 0; j1 < 20;j1++)
                            matC[i+i1][j+j1] += matA[i+i1][k+k1]
* matB[k+k1][j+j1];
```

The reasoning behind this is that by having individual elements as subarrays of data, the operation reuses data that is already in the local memory. Calculation of big amount of numbers is broken into small chunks of computation, each of which uses a small enough piece of the data. The iteration in the code in task there of each loop have n^2 operations (addition and multiplication) and reference to data without reuse. Blocking helps reuse the data referred. The efficiency also slightly improved when the blocks are larger.

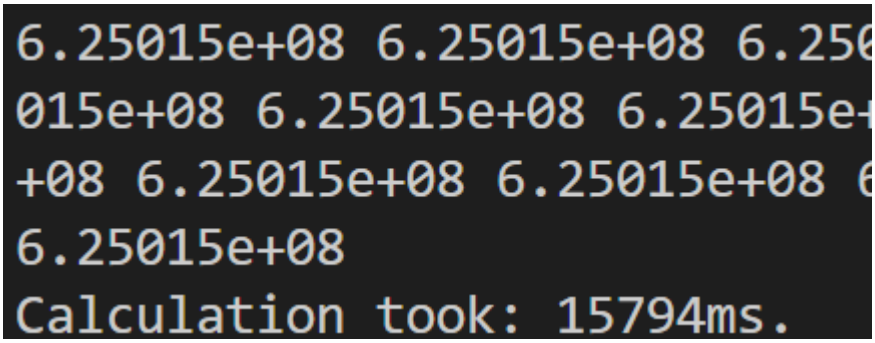
Results of blocking:

1. C_{20x20} with 10-block



Efficiency improved by 2000 ms.

2. C_{20x20} with 20-block



Efficiency improved by 2500 ms.

iii) Compiler Specification

Compile the program using flags: `g++ matrixmult200M_2.cpp -Ofast -march=native -funroll-loops`

Using flags also significantly improved efficiency. They are especially helpful with speeding up program performances. For instance, `-Ofast` disregards strict standards compliance. `-Ofast` enables all `-O3` optimizations. It also enables optimizations that are not valid for all standard-compliant programs. It turns on `-ffast-math` and the Fortran-specific `-fno-protect-parens` and `-fstack-arrays`.

Results of compiler specification

1. C_{2x2}

```
The result is:
1.07374e+09 1.07374e+09
1.07374e+09 1.07374e+09
Calculation took: 388ms.
```

Efficiency improved by 1100ms.

2. C_{20x20} with 20-block

```
6.40255e+08
6.40255e+08 6.40255e+08 6.
255e+08 6.40255e+08 6.4025
+08 6.40255e+08 6.40255e+0
6.40255e+08
Calculation took: 626ms.
```

Efficiency improved by 15000ms.

b) OpenBLAS codes and results

```
#include <iostream>
#include <time.h>
#include <chrono>

#include "cblas.h"

using namespace std;

void cblas_sgemm(const enum CBLAS_ORDER __Order, const enum CBLAS_TR
ANSPOSE __TransA, const enum CBLAS_TRANSPOSE __TransB, const int __M
, const int __N, const int __K, const float __alpha, const float *__
A, const int __lda, const float *__B, const int __ldb, const float _
_beta, float *__C, const int __ldc);


int main(){

    float *matA = (float *)malloc(20 * 100000000 * sizeof(float));
    float *matB = (float *)malloc(10000000 * 20 * sizeof(float));
    float *matC = (float *)malloc(20 * 20 * sizeof(float));

    for(int i=0;i<200000000;i++)
    {
```

```
        srand((unsigned)time(NULL));
        matA[i]= rand()*1.0/RAND_MAX*(10-1)+1;
        matB[i]= rand()*1.0/RAND_MAX*(10-1)+1;
    }

    auto start = chrono::steady_clock::now();
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, 20, 20,10
000000,1,matA,10000000,matB,20,0,matC,20);
    auto end = chrono::steady_clock::now();

    cout << "The calculation took: " << chrono::duration_cast< chron
o::milliseconds>(end - start).count() << "ms.";

    cout << "The result is: " << endl;

    for(int i=0;i<400;i++)
    {
        cout << matC[i] << " ";
        if ((i+1)%20 == 0)
            cout << endl;
    }


    free(matA);
    free(matB);
    free(matC);

    return 0;
}
```

Compile using: `g++ matrixmult200M_ob.cpp -I C:/OpenBlas/OpenBLAS-0.3.10/Temp/include/ -L C:/OpenBlas/OpenBLAS-0.3.10/Temp/lib -Lpath_to_openblas_lib_directory -lopenblas -lpthread -lgfortran`

Results of OpenBLAS on C_{20x20}

```
The calculation took: 594ms.The result is:
1.63886e+08 1.63886e+08 1.63886e+08 1.63886e+08 1.63886e+08 1.6
63886e+08 1.63887e+08 1.63887e+08 1.63887e+08 1.63887e+08 1.638
1.63916e+08 1.63916e+08 1.63916e+08 1.63916e+08 1.63916e+08 1.6
63916e+08 1.63916e+08 1.63916e+08 1.63916e+08 1.63916e+08 1.639
1.63918e+08 1.63918e+08 1.63918e+08 1.63918e+08 1.63918e+08 1.6
63918e+08 1.63918e+08 1.63918e+08 1.63918e+08 1.63918e+08 1.639
```

Task 6: GitHub and Helpful Links

GitHub link:
https://github.com/lounachen/cpp/tree/master/assignment/project1

Blocking algorithm:
http://www.netlib.org/utk/papers/autoblock/node2.html
Flag specification:
https://gcc.gnu.org/onlinedocs/gcc-4.1.0/gcc/Optimize-Options.html
OpenBLAS parameters:
https://developer.apple.com/documentation/accelerate/1513264-cblas_sgemm?language=objc