CS205 C/C++ Programming - Project Report 2

Name:张闻城

SID:12010324

Part 1 - Analysis

1. **Time consumption: float vs. double**

   After some tests, it seems that multiplication of **float** is more faster than multiplication of **double**. I think there are two main reasons that cause this result:

   - **Memory consumed:**

     duoble takes more memory to store data. Take multiplcation between two 2048x2048 matrices as example. A 2048x2048 matrix has 4194304 elements. If stored in **float**, it takes 16Mb of bits, while it takes 32Mb when stored in **double**. The huge difference between memory consumed to store data will cause huge difference between the times of data manipulation and data access. Thus **duoble** takes more time to get the result compared with **float**, especially when the size of matrix is large.

   - **Hardware:**

     The performence of multiplication maybe differ in **float** and **double**. Refer to a statement in ***Optimizing software in C++***,

     > Single precision division, square root and mathematical functions are calculated faster than double precision when the XMM registers are used, while the speed of addition, subtraction, multiplication, etc. is still the same regardless of precision on most processors (when vector operations are not used).

     So maybe undelying structure of hardware will also cause the diference of time consumption.

   - **Something interesting:**

     - The explanation above seems to be right. But after I compare the time consumed during calculating some data (the number of data is not large, just 100) of **float** with that of **double**, I find that **double** is faster than **float**, which is weird.

       Test program

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

       int main()
       {
           float fArr[] = {88.3f};
           double dArr[] = {88.3};
           struct timespec start = {0, 0};
       ```

```
    struct timespec end = {0, 0};

    float fRes = 0.0f;
    clock_gettime(CLOCK_REALTIME, &start);
    for (int i = 0; i < 100; i++)
    {
        fRes += fArr[0] * fArr[0];
    }
    clock_gettime(CLOCK_REALTIME, &end);
    printf("float计算运行时间:%lds %ldns\n", end.tv_sec - start.tv_sec, end.tv_nsec - start.tv_nsec);

    double dRes = 0.0;
    clock_gettime(CLOCK_REALTIME, &start);
    for (int i = 0; i < 100; i++)
    {
        dRes += dArr[0] * dArr[0];
    }
    clock_gettime(CLOCK_REALTIME, &end);
    printf("double计算运行时间:%lds %ldns\n", end.tv_sec - start.tv_sec, end.tv_nsec - start.tv_nsec);

    return 0;
}
```

Output:

After searching on the Internet, I find out the reason. The calculations between floating-point number in computer are all performed as double-precision, no matter it is **double** or not. Thus, the first step of calculation between **float** is to convert **float** to **double**. So, without the time cnosumption of converting progress, **double** will be faster (if the amount of data is not large).

Such being the case, why **float** is still faster than **duoble** in the progress of matirx multiplication? I believe the it is still due to the memory consumption. Since the difference of memory consumption is extremely large (that of the [example]() at the top of article is **16Mb!**), the effect of manipulating such huge data dominates the reason of time consumption difference.

- The explanation seems to be right again. ***However***, after I search on the Internet over again and again, I find something new, again. It is said that there is no such a rule that says **float** will be converted into **double** in **C++**. It also has something to do with the instruction set (**FPU** and **SSE/AVX**). Different instruction set has diferent performence with floating-point number calculation (**FPU** does all floating-point number's calcution as 80-bit number, while **SEE/AVX** do **float** and **double** calculation seperately).

To conclude, the difference of data length is the main reason that cause the difference of matrix multiplication's time consumption between float and double.

2. **Accuracy: float vs. double**

On a 64-bit operating system with a x64 CPU, the size of **float** in C++ is 4 bytes while that of **double** is 8 bytes. Thus **double** can store more bits of mantissa and exponent (11-bit of exponent and 52-bit of mantissa). Obviously, **double** has more accuracy than **float**.

Part 2 - Improve the program

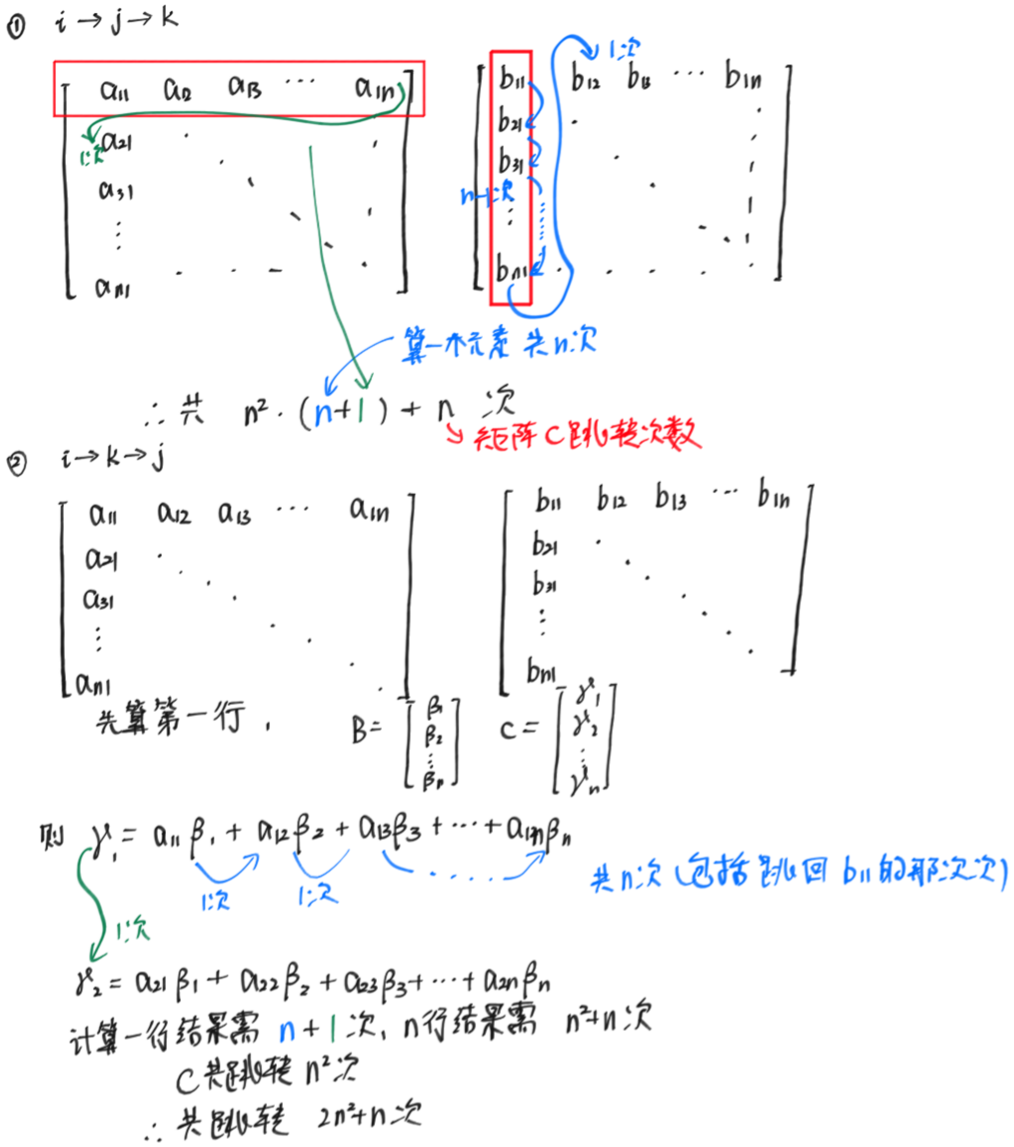
1. **Improve the speed:**

   When the size of is huge, it will take much time to finish the multiplication (eg. $A_{2048\times2048}\times B_{2048\times2048}$ takes about 50s). After searching on the Internet, I found a way to imporve the speed.

   - **Reference:** [C++加速矩阵乘法的最简单方法](#)

     One way to improve the speed is to change the order of loop. It is said that that the data stored in a row of an array is continuous while in column is not. Thus it takes few time to access the data that stored in a row (because you don't have to jump to other address to access discontinuous data). The simple order of loop is $i\to j\to k$. Assume the size of two matrices are all $n\times n$. The number leaps to access discontinuous data is n^3+n^2+n. If change the order to $i\to k\to j$, it needs $2n^2+n$ leaps. Thus the speed will be improve.

     Details of how to get the number of leaps in a matirx multiplication



   - **Comparison of time consumption:**

```
zephyrus@LAPTOP-IIC396SP:/mnt/d/OneDrive - Office/File/Project/VS Code/C++/CS205-Project 2$ ./main mat-A-2048.txt mat-B-2048.txt out.txt
2048x2048的矩阵与2048x2048的矩阵以float,按ijk的顺序相乘得出结果共耗时: 43.827401s
2048x2048的矩阵与2048x2048的矩阵以float,按ikj的顺序相乘得出结果共耗时: 28.053239s
2048x2048的矩阵与2048x2048的矩阵以double,按ijk的顺序相乘得出结果共耗时: 49.933257s
2048x2048的矩阵与2048x2048的矩阵以double,按ikj的顺序相乘得出结果共耗时: 28.836138s
```

> ***Note:****The time consumed only contains the progress of matrix multiplication, not including file reading.*

2. **Get the size of matrix:**

   I get the size of matrix by count how many lines (to get number of rows) and how many spaces within a single line (to get number of columns) in **mat.txt**, so user don't need to add another argument in the file name to indicate the size of matrix.

3. **Multiplication between random size of matrices:**

   Since we can get size of matrix, it is easy to implement. Besides, my program will check if it is possible to do multiplication of two matrices (eg. $A_{m\times n} \times B_{s\times t}$ requires **n = s**).

Part 3 - Code

Things to be known before testing the program:

The format of command line parameter is not strictly restricted. The size of matrix is not required to be mentioned in the file's name. And the name of the file that stored the result is also not neccessary.

```
./matmul mat-A.txt mat-B.txt
```

```
#include <iostream>
#include <string>
#include <cstring>
#include <fstream>
#include <time.h>
using namespace std;

class Matrix
{
private:
    //attributes
    float **fmat;
    double **dmat;
    unsigned int row;
    unsigned int col;
    string filename;

    /**
     * @brief 获取矩阵的行数与列数
     */
    void setSize()
    {
        ifstream ifs;
        ifs.open(filename, ios::in);
        if (!ifs.is_open())
        {
            cout << "读取矩阵 " << filename << " 失败，程序退出" << endl;
            exit(100);
        }
        string buffer;
        while (getline(ifs, buffer))
        {
            row++;
        }
```

```
        ifs.close();
        ifs.open(filename, ios::in);
        if (getline(ifs, buffer))
        {
            for (int i = 0; i < buffer.length(); i++)
            {
                if (buffer[i] == 32)
                {
                    col++;
                }
            }
            col++;
        }
        ifs.close();
    }

public:
    /**
     * @brief 构造矩阵
     * @param filename 要读取的txt文件的名称
     */
    Matrix(string filename)
    {
        row = 0;
        col = 0;
        this->filename = filename;
        setSize();
    }

    unsigned getRow()
    {
        return row;
    }

    unsigned getCol()
    {
        return col;
    }

    /**
     * @brief 将txt文件中的矩阵读取到二维数组中去
     */
    void prepareMat()
    {
        fmat = new float *[row];
        dmat = new double *[row];
        for (int i = 0; i < row; i++)
        {
            fmat[i] = new float[col];
            dmat[i] = new double[col];
        }

        //Initialization
        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                fmat[i][j] = 0.0f;
```

```
                dmat[i][j] = 0.0;
            }
        }

        //Read file
        ifstream ifs;
        ifs.open(filename, ios::in);
        if (!ifs.is_open())
        {
            cout << "读取矩阵 " << filename << " 失败，程序退出" << endl;
            exit(100);
        }
        char buffer[32];
        unsigned int i = 0;
        unsigned int j = 0;
        while (ifs >> buffer)
        {
            if (i == row && j == col)
            {
                break;
            }
            fmat[i][j] = stof(buffer);
            dmat[i][j] = stod(buffer);
            if (++j >= col)
            {
                i++;
                j = 0;
            }
        }
        ifs.close();
    }

    /**
     * @brief float矩阵的乘法，乘法顺序为当前对象的矩阵乘以参数的矩阵
     * @param mat 另一个矩阵对象
     */
    void fmatMul(Matrix mat)
    {
        float **res = new float *[this->row];
        for (int j = 0; j < this->row; j++)
        {
            res[j] = new float[mat.col];
        }
        //Initialization
        for (int i = 0; i < this->row; i++)
        {
            for (int j = 0; j < mat.col; j++)
            {
                res[i][j] = 0.0f;
            }
        }

        time_t start, end;
        start = clock();

        //将矩阵按照 i->j->k 的顺序进行乘法运算
        for (int i = 0; i < this->row; i++)
        {
```

```
            for (int j = 0; j < mat.col; j++)
            {
                float c_i_j = 0.0f;
                for (int k = 0; k < this->col; k++)
                {
                    c_i_j += this->fmat[i][k] * mat.fmat[k][j];
                }
                res[i][j] = c_i_j;
            }
        }

        end = clock();
        printf("%dx%d的矩阵与%dx%d的矩阵以float,按ijk的顺序相乘得出结果共耗时:%fs\n", this->row, this->col, mat.row, mat.col, (double(end - start) / CLOCKS_PER_SEC));

        //将矩阵中的每个元素都重置，为下次运算做准备
        for (int i = 0; i < this->row; i++)
        {
            for (int j = 0; j < mat.col; j++)
            {
                res[i][j] = 0.0f;
            }
        }

        start = clock();

        ////将矩阵按照 i->k->j 的顺序进行乘法运算
        for (int i = 0; i < this->row; i++)
        {
            for (int k = 0; k < this->col; k++)
            {
                float temp = this->fmat[i][k];
                for (int j = 0; j < mat.col; j++)
                {
                    res[i][j] += temp * mat.fmat[k][j];
                }
            }
        }

        end = clock();
        printf("%dx%d的矩阵与%dx%d的矩阵以float,按ikj的顺序相乘得出结果共耗时:%fs\n", this->row, this->col, mat.row, mat.col, (double(end - start) / CLOCKS_PER_SEC));

        //将结果写入txt文件中
        string ofile_name = "out-float-" + to_string(this->row) + "x" + to_string(mat.col) + ".txt";
        ofstream ofs;
        ofs.open(ofile_name, ios::out);
        for (int i = 0; i < this->row; i++)
        {
            for (int j = 0; j < mat.col; j++)
            {
                ofs << res[i][j] << " ";
            }
            ofs << endl;
        }
```

```
        //释放内存
        for (int i = 0; i < this->row; i++)
        {
            delete[] res[i];
        }
        delete[] res;
    }

    /**
     * @brief double矩阵的乘法，乘法顺序为当前对象的矩阵乘以参数的矩阵
     * @param mat 另一个矩阵对象
     */
    void dmatMul(Matrix mat)
    {
        double **res = new double *[this->row];
        for (int j = 0; j < this->row; j++)
        {
            res[j] = new double[mat.col];
        }
        //Initialization
        for (int i = 0; i < this->row; i++)
        {
            for (int j = 0; j < mat.col; j++)
            {
                res[i][j] = 0.0;
            }
        }

        time_t start, end;
        start = clock();
        for (int i = 0; i < this->row; i++)
        {
            for (int j = 0; j < mat.col; j++)
            {
                double c_i_j = 0.0;
                for (int k = 0; k < this->col; k++)
                {
                    c_i_j += this->dmat[i][k] * mat.dmat[k][j];
                }
                res[i][j] = c_i_j;
            }
        }
        end = clock();
        printf("%dx%d的矩阵与%dx%d的矩阵以double,按ijk的顺序相乘得出结果共耗时:%fs\n", this->row, this->col, mat.row, mat.col, (double(end - start) / CLOCKS_PER_SEC));

        for (int i = 0; i < this->row; i++)
        {
            for (int j = 0; j < mat.col; j++)
            {
                res[i][j] = 0.0;
            }
        }

        start = clock();
        for (int i = 0; i < this->row; i++)
```

```
        {
            for (int k = 0; k < this->col; k++)
            {
                double temp = this->dmat[i][k];
                for (int j = 0; j < mat.col; j++)
                {
                    res[i][j] += temp * mat.dmat[k][j];
                }
            }
        }
        end = clock();
        printf("%dx%d的矩阵与%dx%d的矩阵以double,按ikj的顺序相乘得出结果共耗时: %fs\n", this->row, this->col, mat.row, mat.col, (double(end - start) / CLOCKS_PER_SEC));

        string ofile_name = "out-double-" + to_string(this->row) + "x" + to_string(mat.col) + ".txt";
        ofstream ofs;
        ofs.open(ofile_name, ios::out);
        for (int i = 0; i < this->row; i++)
        {
            for (int j = 0; j < mat.col; j++)
            {
                ofs << res[i][j] << " ";
            }
            ofs << endl;
        }

        for (int i = 0; i < this->row; i++)
        {
            delete[] res[i];
        }
        delete[] res;
    }

    /**
     * @brief 将矩阵打印到命令行上
     */
    void printMat()
    {
        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                printf("%5.1f ", dmat[i][j]);
            }
            cout << endl;
        }
    }
};

int main(int argc, char **argv)
{
    if (argc < 3)
    {
        cout << "文件的数量不够。退出! " << endl;
        exit(100);
    }
```

```

    string ifile1_name = argv[1];
    string ifile2_name = argv[2];

    Matrix mat_A(ifile1_name);
    Matrix mat_B(ifile2_name);
    if (mat_A.getCol() != mat_B.getRow())
    {
        cout << "矩阵A的列数不等于矩阵B的行数，无法做矩阵乘法。退出！" << endl;
        exit(100);
    }

    mat_A.prepareMat();
    mat_B.prepareMat();
    mat_A.fmatMul(mat_B);
    mat_A.dmatMul(mat_B);
    return 0;
}
```