CS205 C/C++Program Design - Project 2

姓名: 张琦

SID：12010123

Part 1 - 分析问题

1.1 基本思路:

1. 设计一个能够实现矩阵相乘的程序，该程序计算的矩阵需要程序从`txt`文档中获取，矩阵相乘的结果需要存储在新生成的 `txt` 文档中。
2. 程序要分别计算`float`类型和`double`类型的矩阵并比较两种类型计算所需的时间和计算结果的精度。
3. 提出方法提高程序的运行速度

- 解决步骤：
- 程序设计部分：

1.从命令行传入读写文件的位置需要把主函数表示为：

```
int main(int argc, char **argv)
```

2.数据的读入

2.1 从`txt`文件中读取用于表示矩阵的二维数组，我们导入文件流头文件<*fstream*>中的`ifstream`类。

该类的对象会在当前目录中搜索符合传入的文件名的文档，先用`open()`打开目标文档，再调用`getline()`获取文件一行的内容。

2.2 文档中的数据已经利用空格分隔，所以利用`istringstream`读取的每一个字符串都是一个浮点数。

2.3 利用可变长的<*vector*>容器储存从`istringstream`读取的浮点数。

参考博客：*https://blog.csdn.net/qq_22080999/article/details/82532157*

(使用二维向量`vector<vector<double>>`存储文本中的数据不会使double计算结果失精，但`vector<vector<float>>`会使double失精)

获得的行数和列数进行动态内存分配，分别创建float * * 和double * * 并给它们赋值。

```
ifstream inflie;

inflie.open(argv[1]);
while (getline(inflie, str1)) //读到空白处停止
{
    istringstream input(str1);
    vector<double> tmp;
    double a;
    while (input >> a)
    {
        tmp.push_back(a);
    }
    doubletempA.push_back(tmp);
}
inflie.close();
```

3.创建矩阵并实现矩阵乘法

3.1 从<*vector*>的`size()`函数获取矩阵的行数和列数，利用获得的行数和列数进行动态内存分配，分别创建`float**`和`double **`并给它们赋值。

首先这使得命令行分别传入三个不同大小的矩阵时，程序可以自己识别并创建相应大小的二维数组。其次利用`new`关键字申请的内存存储在堆中，堆的内存空间足够大且存储在堆中的变量不会像栈中变量一样自动分配内存并自动回收，可以满足存储超大矩阵（如2048 * 2048）的需求。

动态内存申请：

```
int size = row;
double **answer;
answer = new double *[size];
for (int i = 0; i < size; i++) //动态内存申请
{
    answer[i] = new double[size];
}
```

4.我们使用普通的三重循环实现矩阵乘法：

三重循环分别对应矩阵A的行遍历，矩阵B的列遍历和矩阵AB相应位置浮点数的乘积遍历。

结果存储在二维数组中。

优化前：

```
for (int i = 0; i < size; i++)
{
    for (int j = 0; j < size; j++)
    {
        for (int k = 0; k < size; k++)
        {
            answer[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

优化后：（以第一类内存寻址优化为例：改变了循环的顺序，属于内存寻址优化，时间分析在下文）

```
for (int i = 0; i < size; i++)
{
    for (int k = 0; k < size; k++)
    {
        float middle2 = A[i][k];
        for (int j = 0; j < size; j++)
        {
            answer[i][j] += middle2 * B[k][j];
        }
    }
}
```

5.时间记录

使用<*ctime*> 中的`clock()`获取当前时间，（clock末-clock初）即为程序运行时间。

6.输出计算结果

6.1 利用`ofstream`在当前目录寻找从命令行传入的文件名对应的文件，若文件不存在则会新建一个。

6.2 以类似`iostream`的输出模式将计算结果输出到目标文件中。

```
double **double_Answer = FastMatMul(doubletempA, doubletempB, size);

//写入double
clock_t doubleStart = clock();
for (int i = 0; i < size; i++)
{
    for (int j = 0; j < size; j++)
    {
        outfile << double_Answer[i][j] << " ";
    }
    outfile << endl;
}
clock_t doubleEnd = clock();
```

7.精度分析

先定位到float矩阵和double矩阵同一个位置的计算结果(如32x32矩阵的第一行第一列的元素),再利用指针分别找到两个浮点数的地址，将地址存储的二进制数打印出来从而比较精度，具体实施在下文。

```
void print_float_Storage_Format(float f) ///float类型的长度是4bytes,32bits
{
    char *p = (char *)&f; /////将存储结果转换成16进制数
    cout << transfer_to_binaryNumber((int)(*p)) << endl;
    cout << transfer_to_binaryNumber((int)(*(p + 1))) << endl;
    cout << transfer_to_binaryNumber((int)(*(p + 2))) << endl;
    cout << transfer_to_binaryNumber((int)(*(p + 3))) << endl;
}
```

7. 加速过程

7.1：内存寻址优化1：改变循环的顺序

7.2: 优化硬件设施（CPU超频）

7.3：多线程加速

7.4：内存寻址优化2：矩阵分块算法

1.2 使用的头文件及作用：

1.2.1 <*iostream*>： 将文本输入到控制台上

1.2.2 <*fstream*>： 把程序跟文件关联起来，从文件读入数据，将数据输出到文件

1.2.3 <*vector*>: 利用vector容器存储文本中的浮点数

1.2.4 <*string*>： 字符串

1.2.5 <*sstream*>： 提供程序和string对象之间的I/O

1.2.6 <*ctime*>: 计时器

(<*bits/stdc++.h*> ：万能头文件)

1.3 操作环境：

- 系统：`Ubuntu20.04（64bit）`
- 编译器：`g++ 9.2.0`
- CPU: `I7-10750H 2.60GHZ`

Part 2 - 源代码

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <ctime>
#include <iomanip>
#include <thread>
#include <cmath>
#include <bits/stdc++.h> ///万能头文件

using namespace std;

vector<vector<double>> file_Reading(string filename);
void float_file_Writing(float **float_matrix, int size, string filename);
void double_file_Writing(double **double_matrix, int size, string filename);
double **FastMatMul(vector<vector<double>> A, vector<vector<double>> B, int
row);
float **FastMatMul(int row, vector<vector<double>> A, vector<vector<double>> B);
double **SlowMatMul(vector<vector<double>> A, vector<vector<double>> B, int
row);
float **SlowMatMul(int row, vector<vector<double>> A, vector<vector<double>> B);
void print_float_Storage_Format(float f);
void print_double_Storage_Format(double f);
int transfer_to_binaryNumber(int n);
void DoubleToString(double *p_dl);
void thread1(int size, vector<vector<double>> doubletempA,
vector<vector<double>> doubletempB, string filename);
void thread2(int size, vector<vector<double>> doubletempA,
vector<vector<double>> doubletempB, string filename);
double **double_block_MatMul(vector<vector<double>> A, vector<vector<double>> B,
int row);
float **float_block_MatMul(vector<vector<double>> A, vector<vector<double>> B,
int row);

int main(int argc, char **argv)
{
    clock_t AllStart = clock();

    vector<vector<double>> doubletempA, doubletempB; // 设置可变长度宽度的二维Vector
容器
    string filename = argv[3];
    int size;
    doubletempA = file_Reading(argv[1]);
    doubletempB = file_Reading(argv[2]);
```

```
    size = doubletempA.size(); //用来创建矩阵的关键量

    thread task01 = thread(thread1, size, doubletempA, doubletempB, filename);
    thread task02 = thread(thread2, size, doubletempA, doubletempB, filename);
    task01.join();
    task02.join();

    clock_t AllEnd = clock();

    cout << "总时长：  " << (double)(AllEnd - AllStart) / CLOCKS_PER_SEC << " s" <<
endl;

    return 0;

    //验证浮点数在内存中的存储形式
    // cout << "float计算结果：  " << float_Answer[0][0] << endl;
    // cout << "float存储形式结果：  " << endl;
    // print_float_Storage_Format(float_Answer[0][0]);
    // cout << "double计算结果：  " << double_Answer[0][0] << endl;
    // cout << "double存储形式结果：  " << endl;
    // DoubleToString(&double_Answer[0][0]);
}

void thread1(int size, vector<vector<double>> doubletempA,
vector<vector<double>> doubletempB, string filename)
{
    clock_t floatStart = clock();
    float **float_Answer = FastMatMul(size, doubletempA, doubletempB);
    float_file_Writing(float_Answer, size, filename);
    clock_t floatEnd = clock();
    delete float_Answer;
    cout << "float类型矩阵的输出时长：  " << (double)(floatEnd - floatStart) /
CLOCKS_PER_SEC << " s" << endl;
}

void thread2(int size, vector<vector<double>> doubletempA,
vector<vector<double>> doubletempB, string filename)
{
    clock_t doubleStart = clock();
    double **double_Answer = FastMatMul(doubletempA, doubletempB, size);
    double_file_Writing(double_Answer, size, filename);
    clock_t doubleEnd = clock();
    delete double_Answer;
    cout << "double类型矩阵的输出时长：  " << (double)(doubleEnd - doubleStart) /
CLOCKS_PER_SEC << " s" << endl;
}

void float_file_Writing(float **float_matrix, int size, string filename)
{
    ofstream outfile("float_" + filename);
    outfile << "Float version :" << endl;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            outfile << float_matrix[i][j] << " ";
        }
        outfile << endl;
```

```
    }
    outfile << endl
            << endl
            << endl
            << endl
            << endl
            << endl
            << endl;
    outfile << flush;
    outfile.close();
}

void double_file_Writing(double **double_matrix, int size, string filename)
{
    ofstream outfile("double_" + filename);
    outfile << "Double version :" << endl;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            outfile << double_matrix[i][j] << " "; ///设置为最大精度
        }
        outfile << endl;
    }
    outfile << endl
            << endl
            << endl
            << endl
            << endl
            << endl
            << endl;
    outfile << flush;
    outfile.close();
}

vector<vector<double>> file_Reading(string filename)
{
    vector<vector<double>> doubletempA;
    ifstream inflie;
    string str1;
    inflie.open(filename);
    while (getline(inflie, str1)) //读到空白处停止
    {
        istringstream input(str1);
        vector<double> tmp;
        double a;
        while (input >> a)
        {
            tmp.push_back(a);
        }
        doubletempA.push_back(tmp);
    }
    inflie.close();
    return doubletempA;
}
```

```
double **double_block_MatMul(vector<vector<double>> A, vector<vector<double>> B,
int row)
{
    int size = row;
    double **answer;
    answer = new double *[size];
    for (int i = 0; i < size; i++)
    {
        answer[i] = new double[size];
    }

    int block_size = 7;
    for (int bi = 0; bi < row; bi += block_size)
    {
        for (int bj = 0; bj < row; bj += block_size)
        {
            for (int bk = 0; bk < row; bk += block_size)
            {
                for (int i = bi; i < min(bi + block_size, row); ++i)
                {
                    for (int j = bj; j < min(bj + block_size, row); ++j)
                    {
                        for (int k = bk; k < min(bk + block_size, row); ++k)
                        {
                            answer[i][j] += A[i][k] * B[k][j];
                        }
                    }
                }
            }
        }
    }
    return answer;
}
float **float_block_MatMul(vector<vector<double>> A, vector<vector<double>> B,
int row)
{
    int size = row;
    float **answer;
    answer = new float *[size];
    for (int i = 0; i < size; i++)
    {
        answer[i] = new float[size];
    }

    int block_size = 7;
    for (int bi = 0; bi < row; bi += block_size)
    {
        for (int bj = 0; bj < row; bj += block_size)
        {
            for (int bk = 0; bk < row; bk += block_size)
            {
                for (int i = bi; i < min(bi + block_size, row); ++i)
                {
                    for (int j = bj; j < min(bj + block_size, row); ++j)
                    {
                        for (int k = bk; k < min(bk + block_size, row); ++k)
                        {
                            answer[i][j] += A[i][k] * B[k][j];
```

```
                        }
                    }
                }
            }
        }
    }
    return answer;
}

double **FastMatMul(vector<vector<double>> A, vector<vector<double>> B, int row)
//乘法函数重载
{

    int size = row;
    double **answer;
    answer = new double *[size];
    for (int i = 0; i < size; i++) //动态内存申请
    {
        answer[i] = new double[size];
    }


    for (int i = 0; i < size; i++)
    {
        for (int k = 0; k < size; k++)
        {
            double middle1 = A[i][k];
            for (int j = 0; j < size; j++)
            {
                answer[i][j] += middle1 * B[k][j];
            }
        }
    }
    return answer;
}

float **FastMatMul(int row, vector<vector<double>> A, vector<vector<double>> B)
//乘法函数重载
{
    int size = row;
    float **answer;
    answer = new float *[size];
    for (int i = 0; i < size; i++)
    {
        answer[i] = new float[size];
    }

    for (int i = 0; i < size; i++)
    {
        for (int k = 0; k < size; k++)
        {
            float middle2 = A[i][k];
            for (int j = 0; j < size; j++)
            {
                answer[i][j] += middle2 * B[k][j];
            }
        }
    }
```

```
    return answer;
}

double **SlowMatMul(vector<vector<double>> A, vector<vector<double>> B, int row)
{
    int size = row;
    double **answer;
    answer = new double *[size];
    for (int i = 0; i < size; i++)
    {
        answer[i] = new double[size];
    }

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            for (int k = 0; k < size; k++)
            {
                answer[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    return answer;
}

float **SlowMatMul(int row, vector<vector<double>> A, vector<vector<double>> B)
{
    int size = row;
    float **answer;
    answer = new float *[size];
    for (int i = 0; i < size; i++)
    {
        answer[i] = new float[size];
    }
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            for (int k = 0; k < size; k++)
            {
                answer[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return answer;
}

void print_float_Storage_Format(float f) ///float类型的长度是4bytes,32bits
{
    char *p = (char *)&f;
    cout << "[";
    cout << transfer_to_binaryNumber((int)(*p));
    cout << transfer_to_binaryNumber((int)(*(p + 1)));
    cout << transfer_to_binaryNumber((int)(*(p + 2)));
    cout << transfer_to_binaryNumber((int)(*(p + 3))) << "]" << endl;
```

```
}

void print_double_Storage_Format(double f) ///double类型的长度是8bytes,64bits
{
    char *p = (char *)&f;
    cout << hex << ((int)(*p)) << " ";
    cout << hex << ((int)(*(p + 1))) << " ";
    cout << hex << ((int)(*(p + 2))) << " ";
    cout << hex << ((int)(*(p + 3))) << " ";
    cout << hex << ((int)(*(p + 4))) << " ";
    cout << hex << ((int)(*(p + 5))) << " ";
    cout << hex << ((int)(*(p + 6))) << " ";
    cout << hex << ((int)(*(p + 7))) << endl;
}

void DoubleToString(double *p_dl)
{
    char c_dl_16[16 + 1];
    char c_dl_64[64 + 1];
    memset(c_dl_64, '\0', sizeof(c_dl_64));
    unsigned char *t = (unsigned char *)p_dl;
    int charCount = sizeof(double);
    memset(c_dl_16, '\0', sizeof(c_dl_16));

    if (t == NULL)
        return;
    int i = 0;
    for (i = 0; i < charCount; i++)
    {
        sprintf(c_dl_16 + i * 2, "%02x", t[i]);
    }
    /*printf("\n");  */
    printf("[%s]\n", c_dl_16);
    char tmpc[4 + 1];
    for (int j = 0; j < 16; j++)
    {
        memset(tmpc, '\0', sizeof(tmpc));
        switch (c_dl_16[j])
        {
        case '0':
            memcpy(tmpc, "0000", 4);
            break;
        case '1':
            memcpy(tmpc, "0001", 4);
            break;
        case '2':
            memcpy(tmpc, "0010", 4);
            break;
        case '3':
            memcpy(tmpc, "0011", 4);
            break;
        case '4':
            memcpy(tmpc, "0100", 4);
            break;
        case '5':
            memcpy(tmpc, "0101", 4);
            break;
        case '6':
```

```
            memcpy(tmpc, "0110", 4);
            break;
        case '7':
            memcpy(tmpc, "0111", 4);
            break;
        case '8':
            memcpy(tmpc, "1000", 4);
            break;
        case '9':
            memcpy(tmpc, "1001", 4);
            break;
        case 'a':
            memcpy(tmpc, "1010", 4);
            break;
        case 'b':
            memcpy(tmpc, "1011", 4);
            break;
        case 'c':
            memcpy(tmpc, "1100", 4);
            break;
        case 'd':
            memcpy(tmpc, "1101", 4);
            break;
        case 'e':
            memcpy(tmpc, "1110", 4);
            break;
        case 'f':
            memcpy(tmpc, "1111", 4);
            break;
        }
        sprintf(c_dl_64 + j * 4, "%s", tmpc);
    }
    printf("[%s]\n", c_dl_64);
}

//利用数字逻辑十进制转二进制数的方法设计的转换函数
int transfer_to_binaryNumber(int n)
{
    int flag = 1, y = 0, remainder;
    while (1)
    {
        remainder = n % 2;
        n /= 2;
        y += remainder * flag;
        flag *= 10;
        if (n == 0)
        {
            break;
        }
    }
    return y;
}
```

Part 3 - 结果检验、比较和优化

1.计算结果检验：

因为计算不同大小的矩阵的算法完全相同，为了方便展示，仅展示了32x32矩阵乘积的结果：

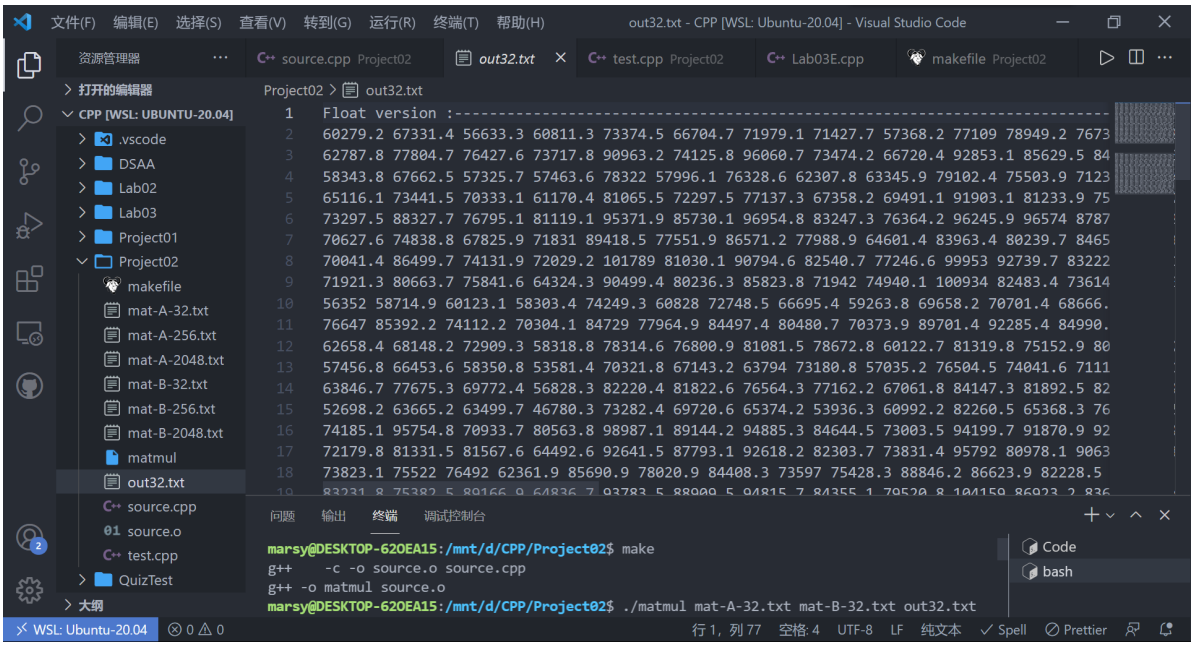
使用MATLAB对程序的计算结果进行检验：

Matlab计算结果：

untitled.mlx | ans = 32×32

	1	2	3	4	5	6	7
1	60279.15	67331.41	56633.33	60811.26	73374.49	66704.71	719
2	62787.81	77804.66	76427.64	73717.81	90963.16	74125.79	960
3	58343.78	67662.49	57325.66	57463.63	78322.03	57996.06	763
4	65116.06	73441.54	70333.09	61170.42	81065.48	72297.54	771
5	73297.5	88327.74	76795.11	81119.1	95371.84	85730.06	969
6	70627.64	74838.81	67825.94	71831.01	89418.45	77551.92	865
7	70041.44	86499.73	74131.91	72029.21	101789.25	81030.09	907
8	71921.27	80663.66	75841.62	64324.26	90499.37	80236.33	858
9	56351.95	58714.94	60123.12	58303.36	74249.32	60828.02	727
10	76646.99	85392.24	74112.17	70304.13	84728.99	77964.85	844
11	62658.36	68148.18	72909.31	58318.81	78314.58	76800.91	810
12	57456.84	66453.59	58350.81	53581.4	70321.75	67143.23	637
13	63846.71	77675.32	69772.45	56828.27	82220.42	81822.58	765
14	52698.16	63665.2	63499.7	46780.28	73282.43	69720.63	653
15	74185.09	95754.78	70933.68	80563.83	98987.13	89144.17	948
16	72179.75	81331.53	81567.6	64492.58	92641.48	87793.1	926

C++计算结果：



虽然只检验了16行矩阵，但可以看出算法是正确的

2.关于float和double类型数时间消耗的比较：

我为了节省内存和时间，只进行了一次读入。

读入时使用vector容器存储了float类型浮点数，再利用存储的数据进行计算，因此两种类型的浮点数的读入时间可以视为是相同。而且读入的时间非常短，假设不考虑读入时间，只考虑float和double的矩阵计算以及写入文本所需的时间：

（没有设置两种浮点类型数的输出有效数字）

float和double类型矩阵计算时间消耗的比较(普通加减乘除对比)：

32 x 32矩阵：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-32.txt mat-B-32.txt out32.txt
float类型矩阵计算的所需的时间：  0.000239 s
double类型矩阵计算的所需的时间：  0.000205 s
```

256 x 256矩阵：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-256.txt mat-B-256.txt out256.t
xt
float类型矩阵计算的所需的时间：  0.095273 s
double类型矩阵计算的所需的时间：  0.093793 s
```

2048 x 2048矩阵:

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-2048.txt mat-B-2048.txt out204
8.txt
float类型矩阵计算的所需的时间：  50.9369 s
double类型矩阵计算的所需的时间：  57.5442 s
```

2.2 将float和double类型矩阵写入文档时间消耗的比较(硬盘读写对比)：

32 x 32矩阵：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-32.txt mat-B-32.txt out32.txt
float类型矩阵写入文本的所需的时间：  0.001248 s
double类型矩阵写入文本的所需的时间：  0.001205 s
```

256 x 256矩阵：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-256.txt mat-B-256.txt out256.txt
float类型矩阵写入文本的所需的时间：  0.035298 s
double类型矩阵写入文本的所需的时间：  0.035304 s
```

2048 x 2048矩阵:

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-2048.txt mat-B-2048.txt out2048.txt
float类型矩阵写入文本的所需的时间：  2.29607 s
double类型矩阵写入文本的所需的时间：  2.26523 s
```

从上述结果可以看出，在数据量小的时候，double类型矩阵计算和写入所需的总时间和float类型矩阵计算和写入所需总时间相差不大。

对于矩阵计算部分的时间消耗，即普通加减乘除对比的时间消耗，数据量小时float类型和double类型时间消耗差别不大。当数据量大时，double类型会比float类型慢了近1/10，所以在算数运算过程中，float会比double快，且数据量越大效果越明显。可以看出在普通算数运算方面，float更有优势。

对于将计算结果写入文本所需要的时间，即硬盘读写的时间消耗，double类型所需的时间均比float类型所需的时间短，可以看出在读写方面，double更有优势。

（由于时间缘故，没有设计数据量更大或数据量更小的矩阵样例来验证上述结论是否是普遍适用结论）

3.关于float和double类型数精度的比较：

起初我的思路是利用<*iomanip*>中的精度设置函数来验证两种浮点类型数的精度。但是于老师告诉我，精度设置函数只在流中起作用，设置精度后通过流输出并显示在控制台上的数据并不是计算机真正存储的数据。于老师强调计算机内的数据都以二进制的形式存储，于是我考虑以二进制的形式展示两种不同浮点数对同一计算结果的存储方式。

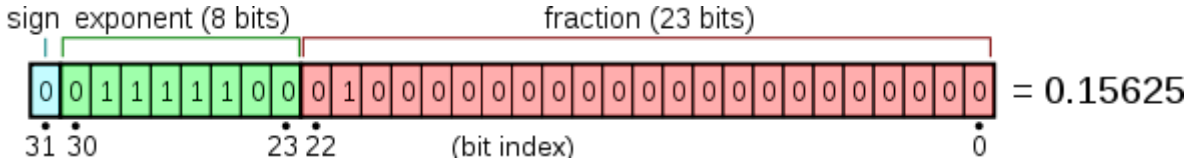
我们都知道double的字节数是8位，float的字节数是4位。

所以最后，我决定用指针分别找到利用两个浮点数得到的"同一计算结果"(如32x32矩阵的第一行第一列的元素)的地址，并将地址存储的二进制数打印出来，比较两种浮点数的二进制的有效位数（非0位数）来比较精度。

float类型数在内存中的存储模式，double类型数同理:

分别对应： 符号位 阶码 尾数

(图片来源：CS203 Lecture2 PPT)



使用函数：(以float类型的为例)

```
void print_float_Storage_Format(float f) ///float类型的长度是4bytes,32bits
{
    char *p = (char *)&f; /////将存储结果转换成16进制数
    cout << "[";
    cout << transfer_to_binaryNumber((int)(*p));
    cout << transfer_to_binaryNumber((int)(*(p + 1)));
    cout << transfer_to_binaryNumber((int)(*(p + 2)));
    cout << transfer_to_binaryNumber((int)(*(p + 3))) << "]" << endl;
}
```

```
for (int i = 0; i < size; i++)
{
    for (int j = 0; j < size; j++)
    {
        for (int k = 0; k < size; k++)
        {
            answer[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

```
float **float_Answer = FastMatMul(size, floattempA, floattempB);
cout << "float计算结果:  " << float_Answer[0][0] << endl;
cout << "float存储形式结果:  " << endl;
print_float_Storage_Format(float_Answer[0][0]);
double **double_Answer = FastMatMul(floattempA, floattempB, size);
cout << "double计算结果:  " << double_Answer[0][0] << endl;
cout << "double存储形式结果:  " << endl;
print_double_Storage_Format(double_Answer[0][0]);
```

以32x32矩阵的第一行第一列的元素为例

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-32.txt mat-B-32.txt out32.txt
float计算结果:  60279.2
float存储形式结果:
[10100011101111101011000111]
double计算结果:  60279.1
double存储形式结果:
[ccccccccce46eed40]
[1100110011001100110011001100110011100100011011101110110101000000]
```

可以看出虽然在控制台中打印出来的值差别不大，但是在内存存储中，double类型的尾数位要比float类型的尾数位多得多，所以double比float精确这个说法是正确的。

4.加速方法：

4.1 内存寻址优化1：改变循环的顺序：

（参考：https://blog.csdn.net/codeur/article/details/72958907）

CPU读数据时，并不是直接访问内存，而是先查看缓存中是否有数据，有的话直接从缓存读取。而从缓存读取数据比从内存读数据快很多。因此，改变矩阵乘法的顺序，对循环进行内存寻址优化，即使得CPU多次从缓存中读取有效数据，可以实现矩阵乘法的提速。

时间比较：

以2048 x 2048的矩阵为例

提速前代码：

```cpp
float **SlowMatMul(int row, vector<vector<double>> A, vector<vector<double>> B) //乘法函数重载
{
    int size = row;
    float **answer;
    answer = new float *[size];    //更换循环顺序：https://zhuanlan.zhihu.com/p/146250334
    for (int i = 0; i < size; i++) //动态内存申请
    {
        answer[i] = new float[size];
    }
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            for (int k = 0; k < size; k++)
            {
                answer[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return answer;
}
```

提速后代码：

```cpp
float **FastMatMul(int row, vector<vector<double>> A, vector<vector<double>> B) //乘法函数重载
{
    int size = row;
    float **answer;
    answer = new float *[size];    //更换循环顺序：https://zhuanlan.zhihu.com/p/146250334
    for (int i = 0; i < size; i++) //动态内存申请
    {
        answer[i] = new float[size];
    }

    for (int i = 0; i < size; i++)
    {
        for (int k = 0; k < size; k++)
        {
            float middle2 = A[i][k];
            for (int j = 0; j < size; j++)
            {
                answer[i][j] += middle2 * B[k][j];
            }
        }
    }

    return answer;
}
```

提速前时间：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-2048.txt mat-B-2048.txt out2048.txt

程序完成运行：
总时长：  313.024 s
float类型矩阵的输出时长：  154.555 s
double类型矩阵的输出时长：  156.82 s
```

提速后时间：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-2048.txt mat-B-2048.txt out204
8.txt
float类型矩阵的输出时长：  58.3515 s
double类型矩阵的输出时长：  52.6459 s
```

可以看出这个方法是很有效的，两种浮点数类型的矩阵的输出时间都快了接近3倍。

4.2 优化硬件设施：

因为优化硬件设施的方案比较难实现，所以在这里我没有做时间测试。

但CPU性能的提升可以加快程序运行的速度是公认的,所以优化硬件设施也是一个有效的方法。

4.3 多线程加速：

我们引入<*thread*>头文件使用多线程并行的方法来加速总时间。

引入线程之后，进程就变成了分配资源的基本单位，线程是独立调度的基本单位。

在实行多线程前，程序是顺序执行的。例如我们分别进行了两个矩阵的计算，第二个矩阵一定会在第一个矩阵完成计算并写入文档后才开始计算。但是实行了双线程后，两种类型的矩阵计算并写入文档的操作是并行的，因此程序运行的总时间一定会缩短，且估计缩短为原来的一半左右。

因为我对C++<*thread*>中线程的使用非常陌生，只知道通过thread传入函数指针可以实现并行，所以改写了原程序。将矩阵的读入，写入和计算都包装成了函数，方便线程的调用。

```
vector<vector<double>> file_Reading(string filename)
{
    vector<vector<double>> doubletempA;
    ifstream inflie;
    string str1;
    inflie.open(filename);
    while (getline(inflie, str1)) //读到空白处停止
    {
        istringstream input(str1);
        vector<double> tmp;
        double a;
        while (input >> a)
        {
            tmp.push_back(a);
        }
        doubletempA.push_back(tmp);
    }
    inflie.close();
    return doubletempA;
}
```

注：因为我利用<*fstream*>中的`ofstream`将数据写入文档，又把写入过程包装成了一个函数，第二次调用文本编写函数时会将原来编写的文档覆盖。如果按照要求在原`txt`继续写入数据，需要先定位到文档的空白行。当进行2048 x 2048矩阵的写入数据时，定位到空白行需要一定的时间，会导致测量结果不精确，所以在这里我没有按照老师的要求，将输出文件改成了两个不同的文件。

以double为例：

```cpp
void double_file_Writing(double **double_matrix, int size, string filename)
{
    ofstream outfile("double_" + filename);
    outfile << "Double version :" << endl;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            outfile << double_matrix[i][j] << " "; ///设置为最大精度
        }
        outfile << endl;
    }
    outfile << endl
            << endl
            << endl
            << endl
            << endl
            << endl
            << endl;
    outfile << flush;
    outfile.close();
}
```

多线程的具体实现：

子线程：

```cpp
thread task01 = thread(thread1, size, doubletempA, doubletempB, filename);
thread task02 = thread(thread2, size, doubletempA, doubletempB, filename);
task01.join();
task02.join();
```

为了方便调用编写的线程函数：

```cpp
void thread1(int size, vector<vector<double>> doubletempA, vector<vector<double>> doubletempB, string filename)
{
    clock_t floatStart = clock();
    float **float_Answer = FastMatMul(size, doubletempA, doubletempB);
    float_file_Writing(float_Answer, size, filename);
    clock_t floatEnd = clock();
    delete float_Answer;
    cout << "float类型矩阵的输出时长：  " << (double)(floatEnd - floatStart) / CLOCKS_PER_SEC << " s" << endl;
}
```

以2048 x 2048规格的矩阵输出时长为例（都采用了方法一中的乘法算法）：

提速前时间：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-2048.txt mat-B-2048.txt out204
8.txt
float类型矩阵的输出时长：  58.3515 s
double类型矩阵的输出时长：  52.6459 s
```

（注：总时长约120s）

提速后时间：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-2048.txt mat-B-2048.txt out204
8.txt
double类型矩阵的输出时长：  111.97 s
float类型矩阵的输出时长：  120.104 s
总时长：  121.897 s
```

可以看出，程序的确是有两个子线程运行的，因为总时长和两种类型矩阵的输出时长很接近。

理论上双线程的时长应该是比单线程的时长短的，但出乎意料的是，分别输出两种矩阵的时长反而增长了，导致双线程执行该程序的总时长和单线程执行该程序的总时长差别不大。

这可能是内存占用也可能是CPU的原因，我在这里只比较了时长，并没有对原因做具体分析。总之，在没有对多线程程序进行化前，本方法（只使用双线程）的效果不明显。

4.4 内存寻址优化2：矩阵分块：

参考文章：https://zhuanlan.zhihu.com/p/342923482

矩阵分块计算是线性代数中一个处理矩阵乘法的常用方法。

矩阵分块能够加速矩阵乘法的原因与改变循环顺序加速的原因基本相同，都是内存寻址方面的优化，帮助程序高效利用缓存（cache）。

代码：

```
double **double_block_MatMul(vector<vector<double>> A, vector<vector<double>> B, int row)
{
    int size = row;
    double **answer;
    answer = new double *[size];   //更换循环顺序：https://zhuanlan.zhihu.com/p/146250334
    for (int i = 0; i < size; i++) //动态内存申请
    {
        answer[i] = new double[size];
    }

    int block_size = 7;
    for (int bi = 0; bi < row; bi += block_size)
    {
        for (int bj = 0; bj < row; bj += block_size)
        {
            for (int bk = 0; bk < row; bk += block_size)
            {
                for (int i = bi; i < min(bi + block_size, row); ++i)
                {
                    for (int j = bj; j < min(bj + block_size, row); ++j)
                    {
                        for (int k = bk; k < min(bk + block_size, row); ++k)
                        {
                            answer[i][j] += A[i][k] * B[k][j];
                        }
                    }
                }
            }
        }
    }
    return answer;
}
```

提速前总时长：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-2048.txt mat-B-2048.txt out204
8.txt
总时长:  397.292 s
float类型矩阵的输出时长:  212.002 s
double类型矩阵的输出时长:  183.623 s
```

提速后总时长：

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-2048.txt mat-B-2048.txt out204
8.txt
总时长:  217.215 s
float类型矩阵的输出时长:  108.461 s
double类型矩阵的输出时长:  106.854 s
```

可以看出提速效果明显，总时长缩短了接近一半，但是矩阵分块的效果并没有第一种内存寻址优化的方法明显，原因是矩阵分块的循环过多，六层循环使得导致时间复杂度大大增加。

Part 4 - 遇到的困难及解决方法

1. 如何打开txt文件并读取其中的数据：

   1.先把6个存储了矩阵的txt文件放在CPP文件所在的同一个目录。

   2.再从命令行传参，即使用 `int main(int *argc*, char ***argv*)` 传递文件的名字。

   3.利用 `<fstream>` 中的ifstream和getline( )函数获取每一行所对应的字符串，因为矩阵中的元素通过空格分隔，可以直接利用 `<sstream>` 中的istringstream读取矩阵中的元素，再将转化为浮点数的字符串存储在 `vector<vector< double >>` 容器中。

2. 如何存储超大的矩阵而不出现栈溢出错误：

   在计算2048 X 2048的矩阵时，如果直接使用 `double matrix[ 2048 ][ 2048 ]` 或 `float matrix[ 2048 ][ 2048 ]` ，则会在计算过程中出现栈溢出错误。这样的 `matrix` 变量属于临时变量，存储在栈中，临时变量是系统来决定空间的分配和回收的。当临时变量存储了很大的数据时，程序大概率会发生栈溢出错误（StackOverflowError）。

   通过动态内存分配可以使得创建的二维数组存储在堆中，堆中的变量一般由程序员分配，在程序员手动操作或程序结束后释放，从而避免栈溢出错误。

   利用指针实现动态内存申请：

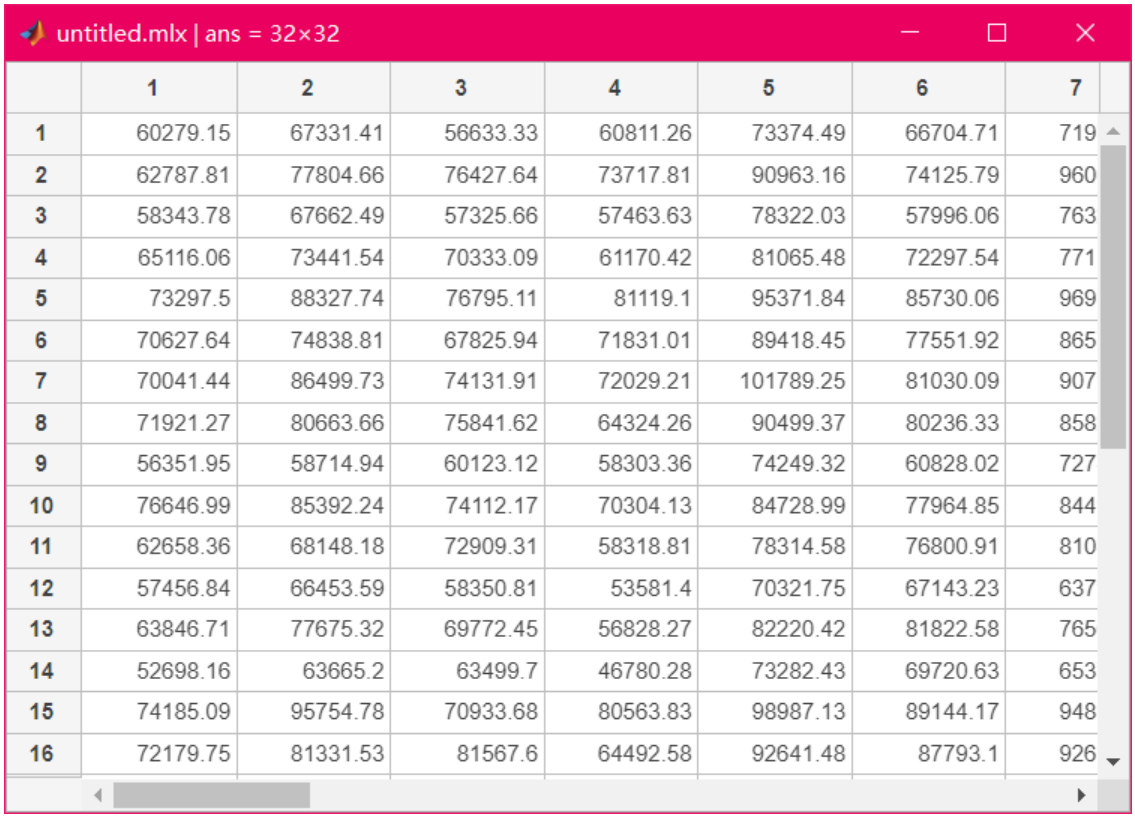
```
int size = row;
double **answer;
answer = new double *[size];
for (int i = 0; i < size; i++) //动态内存申请
{
    answer[i] = new double[size];
}
```

3. 如何保证计算结果一定是正确的：

   浮点数的值实际上都是不完全精确的，而是一个近似值，很难验证矩阵计算结果是否精确。

   1.考虑使用MATLAB自带的矩阵计算器进行验证：

   (32 X 32矩阵计算的结果验证)



	1	2	3	4	5	6	7
1	60279.15	67331.41	56633.33	60811.26	73374.49	66704.71	719
2	62787.81	77804.66	76427.64	73717.81	90963.16	74125.79	960
3	58343.78	67662.49	57325.66	57463.63	78322.03	57996.06	763
4	65116.06	73441.54	70333.09	61170.42	81065.48	72297.54	771
5	73297.5	88327.74	76795.11	81119.1	95371.84	85730.06	969
6	70627.64	74838.81	67825.94	71831.01	89418.45	77551.92	865
7	70041.44	86499.73	74131.91	72029.21	101789.25	81030.09	907
8	71921.27	80663.66	75841.62	64324.26	90499.37	80236.33	858
9	56351.95	58714.94	60123.12	58303.36	74249.32	60828.02	727
10	76646.99	85392.24	74112.17	70304.13	84728.99	77964.85	844
11	62658.36	68148.18	72909.31	58318.81	78314.58	76800.91	810
12	57456.84	66453.59	58350.81	53581.4	70321.75	67143.23	637
13	63846.71	77675.32	69772.45	56828.27	82220.42	81822.58	765
14	52698.16	63665.2	63499.7	46780.28	73282.43	69720.63	653
15	74185.09	95754.78	70933.68	80563.83	98987.13	89144.17	948
16	72179.75	81331.53	81567.6	64492.58	92641.48	87793.1	926

2.利用网页在线矩阵计算器验证：

网站：http://www.yunsuan.info/matrixcomputations/solvematrixmultiplication.html

(以32 x 32矩阵前八列的部分值为例)

您所输入问题的解C=A*B如下：

第1列	第2列	第3列	第4列	第5列	第6列	第7列	第8列
60279.1500	67331.4100	56633.3300	60811.2600	73374.4900	66704.7100	71979.1400	71427.7100
62787.8100	77804.6600	76427.6400	73717.8100	90963.1600	74125.7900	96060.6600	73474.1500
58343.7800	67662.4900	57325.6600	57463.6300	78322.0300	57996.0600	76328.6300	62307.8100
65116.0600	73441.5400	70333.0900	61170.4200	81065.4800	72297.5400	77137.3400	67358.1600
73297.5000	88327.7400	76795.1100	81119.1000	95371.8400	85730.0600	96954.8200	83247.3200
70627.6400	74838.8100	67825.9400	71831.0100	89418.4500	77551.9200	86571.2100	77988.9100
70041.4400	86499.7300	74131.9100	72029.2100	101789.2500	81030.0900	90794.5600	82540.6900
71921.2700	80663.6600	75841.6200	64324.2600	90499.3700	80236.3300	85823.8200	71942.0000
56351.9500	58714.9400	60123.1200	58303.3600	74249.3200	60828.0200	72748.4700	66695.4400
76646.9900	85392.2400	74112.1700	70304.1300	84728.9900	77964.8500	84497.3800	80480.6600
62658.3600	68148.1800	72909.3100	58318.8100	78314.5800	76800.9100	81081.4900	78672.7500
57456.8400	66453.5900	58350.8100	53581.4000	70321.7500	67143.2300	63793.9700	73180.8400
63846.7100	77675.3200	69772.4500	56828.2700	82220.4200	81822.5800	76564.2800	77162.2500
52698.1600	63665.2000	63499.7000	46780.2800	73282.4300	69720.6300	65374.1600	53936.3400
74185.0900	95754.7800	70933.6800	80563.8300	98987.1300	89144.1700	94885.3400	84644.4700
72179.7500	81331.5300	81567.6000	64492.5800	92641.4800	87793.1000	92618.1700	82303.7100
73823.1500	75522.0200	76492.0300	62361.9300	85690.9300	78020.8800	84408.2800	73596.9700

4. 如何证明double类型数比float类型数的计算结果更精确

先找到两种规格相同但类型不同的矩阵的对应位置的浮点数，再打印出这两个浮点数在内存中的存储形式。

用指针分别找到利用两个浮点数的地址，并将地址存储的二进制数打印出来，再比较两种浮点数的二进制的有效位数（非0位数）来比较精度。

其中double类型数的存储形式表示存在问题，解决前参考了博客中的函数：https://blog.csdn.net/mydriverc2/article/details/84628950

```
marsy@DESKTOP-62OEA15:/mnt/d/CPP/Project02$ ./matmul mat-A-32.txt mat-B-32.txt out32.txt
float计算结果：  60279.2
float存储形式结果：
[10100011101111101011000111]
double计算结果：  60279.1
double存储形式结果：
[ccccccccce46eed40]
[1100110011001100110011001100110011100100011011101110110101000000]
```

5. 如何实现相同参数但输出类型不同的“类函数重载”

将参数的顺序调换，实现模拟函数重载

```
double **FastMatMul(vector<vector<double>> A, vector<vector<double>> B, int row);
float **FastMatMul(int row, vector<vector<double>> A, vector<vector<double>> B);
double **SlowMatMul(vector<vector<double>> A, vector<vector<double>> B, int row);
float **SlowMatMul(int row, vector<vector<double>> A, vector<vector<double>> B);
```

6. 如何使用多线程编程：

将需要用子线程运行的程序包装成一个新的函数从而直接使用线程调用函数。

```
thread task01 = thread(thread1, size, doubletempA, doubletempB, filename);
thread task02 = thread(thread2, size, doubletempA, doubletempB, filename);
task01.join();
task02.join();
```

注：如果使用线程，在编译时要注释

我使用的命令：`g++ -std=c++11  -Wl,--no-as-needed  source.cpp  -pthread -o matmu`