CS205 C/ C++ Programming - Project2

> 樊顺
>
> 11811901

1. Part 1 Requirements

- redirect the output to the file.
- Calculate the matrix multiple as double and float separately.
- Improving the speed of calculating.

2. Part 2 The Implement of the Requirements

2.1. **arguments process**

For the input data, since I don't know the matrix size of the input file, I used the function: `int getNumber(string name);` to get the number in the input parameter to determine the size of the matrix in the input file. The code of this function show as follow:

```
int getNumber(string name)
{
    int len_s = name.size();
    int i=0, j=0;
    while (i < len_s)
    {
        if (name[i] >= '0'&& name[i] <= '9')
        {
            j = i;
            int len = 0;
            while (name[i] >= '0'&& name[i] <= '9')
            {
                i++;
                len++;
            }
            string s0 = name.substr(j, len);//获取子串
            int num=0;//数字字符串转换为整型数字
            stringstream s1(s0);
            s1 >> num;
```

```
            return num;
        }
        else
        {
        i++;
        }
    }
    return 0;
}
```

2.2. **redirect the output to the file**

In this part, I use the `<fstream>` to implement the redirect the output to the file. And when finish the output, close the stream. The code of output show as follow:

```
.....
ofstream file(out);
file.setf(ios::fixed);
for (int i = 0; i < result.size(); i++)
    {
        for (int j = 0; j < result[i].size(); j++)
        {
            file << result[i][j] << " ";
        }
        file << endl;
    }
file.close();
.....
```

2.3. **Calculate the matrix multiple as double and float separately**

In this task, I use the double type to store all input numbers in two vectors. After the calculate, the result also store as double in vector.

In the basic implement, I use the Two-dimensional array ***`vector<vector<double>>`*** to store the input matrix.

Due to storage structure, the calculate was always using the double to calculate. And when the result output to the file, I make sure that the precision of all output data is **`5`** decimal places.

2.4. **Improving the speed of calculating**

2.4.1. Brute force solution

- In this implement, I use three `for loop` to calculate two matrix multiple. The code is :

```
vector<vector<double>> calculate(
                                    vector<vector<double>> &a,
                                    vector<vector<double>> &b)
```

```
{
    vector<vector<double>> result;
    for (int i = 0; i < a.size(); i++)
    {
        vector<double> temp;
        for (int j = 0; j < b.size(); j++)
        {
            double res = 0;
            for (int k = 0; k < a.size(); k++)
            {
                res += a[i][k] * b[k][j];
            }
            temp.push_back(res);
        }
        result.push_back(temp);
    }
    return result;
}
```

- This method is the `basic and slowest` implementation. The time cost in calculate two matrix(the size both are 2048 * 2048) is `147s`.

```
./matmul mat-A-2048.txt mat-B-2048.txt out2048.txt
Read in First file used time :0.709465s
Read in second file used time :0.710928s
Calculate two matrix multiple used time :147.305s
Output the result to file used time :2.674s
```

2.4.2. Reverse cycle order

- In this Implementation, I change the cycle order, make the first loop the `k`, the loop `j`(which means this method value taken from the a array is first calculated with the data in the column of the b array.).

The code is :

```
vector<vector<double>> calculateReverse(
                                    vector<vector<double>> &a,
                                    vector<vector<double>> &b)
{
    vector<vector<double>> result;
    int size = a.size();
    for (int i = 0; i < size; i++)
    {
        vector<double> temp;
        for (int k = 0; k < size; k++)
        {
            double res = 0;
            double s = a[i][k];
            for (int j = 0; j < size; j++)
```

```
            {
                res += s * b[k][j];
            }
            temp.push_back(res);
        }
        result.push_back(temp);
    }
    return result;
}
```

- For this method, the performance of the program has been significantly improved, greatly reducing the time consumed by the calculation.

```
./matmul mat-A-2048.txt mat-B-2048.txt out2048.txt
Read in First file used time :0.704428s
Read in second file used time :0.699405s
Calculate two matrix multiple used time                                 :142.224s
Calculate two matrix multiple with Reverse cycle order used time        :32.7107s
Output the result to file used time :2.664s
Output the result to out2 file used time :2.586s
```

Analysis

In the analysis, it was found that the reasons for the inefficiency of matrix multiplication were on the one hand due to the complexity of the $O(n^3)$ algorithm, and on the other hand due to the discontinuity of memory access, which led to the low hit rate of the cache in the storage space.

Assume the definition of a jump uncle to evaluate the discontinuity of memory access.

> Assuming that the matrix is stored in rows.

For a loop like `ijk`, after each number in `result matrix` is counted, it jumps once, and jumps back to the beginning after the end of a line. For a matrix of size `n`, the number of hops is $n^3 + n^2 + n$ times.

For a loop like `ikj`, it calculates the matrix row by row, so the result matrix will only `jump once after each row is calculated`. During this period, the `b` matrix jumped `n times` in total, that is, during the calculation of one row, it jumped `n+1` times. The result matrix has a total of `n` rows, so the total number of jumps is $n^2 + n$, which greatly reduces the number of jumps and improves the operating efficiency.