中山大学计算机院本科生实验报告

（2024 学年秋季学期）

课程名称：高性能计算程序设计　　**批改人：**

实验	**通用矩阵乘法**	专业（方向）	**信息与计算科学**
学号	**22336044**	姓名	**陈圳煌**
Email	**2576926165@qq.com**	完成日期	**2024.09.18**

1.实验目的

通过 python、java、C 三种语言实现矩阵乘法，分析不同语言以及不同优化下运行速度的差异。

2.实验过程和核心代码

分别实现 C 语言、Python 以及 Java 的矩阵乘法

C 语言：

```
#include<iostream>
#include<cstdlib>
#include<ctime>
#include<chrono>
#include<iomanip>
#include<random>
using namespace std;

int main(){
    int M,N,K;
    printf("PUT IN NUM:\n");
    scanf("%d %d %d",&M,&N,&K);
    //分配空间
    double **A=new double*[M];
    double **B=new double*[N];
    double **C=new double*[M];
    for(int i=0;i<M;i++){
        A[i]=new double[N];
    }
    for(int i=0;i<N;i++){
        B[i]=new double[K];
    }
    for(int i=0;i<M;i++){
        C[i]=new double[K];
    }
```

```
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<> dis(0.0,1.0);
//初始化
for(int i=0;i<M;i++){
    for(int j=0;j<N;j++){
        A[i][j]=dis(gen);
    }
}

for(int i=0;i<N;i++){
    for(int j=0;j<K;j++){
        B[i][j]=dis(gen);
    }
}

for(int i=0;i<M;i++){
    for(int j=0;j<K;j++){
        C[i][j]=0.0;
    }
}
//开始时间
auto start=chrono::high_resolution_clock::now();
//矩阵运算
for(int i=0;i<M;i++){
    for(int j=0;j<K;j++){
        for(int l=0;l<N;l++){
            C[i][j]+=A[i][l]*B[l][j];
        }
    }
}
//结束时间
auto end=chrono::high_resolution_clock::now();
chrono::duration<double> time_=end-start;

printf("Used time: %.6f s",time_.count());
//释放空间
for(int i=0;i<M;i++){
    delete []A[i];
}
for(int i=0;i<N;i++){
```

```
        delete []B[i];
    }
    for(int i=0;i<M;i++){
        delete []C[i];
    }
    delete A;
    delete B;
    delete C;
}
```

Python:

```
import time
import random
M,N,K=map(int,input().split())
#初始化
A=[[random.uniform(0,1) for j in range(N)] for i in range(M)]
B=[[random.uniform(0,1) for j in range(K)] for i in range(N)]
C=[[0 for j in range(K)] for i in range(M)]
#开始时间
time1=time.time()
#矩阵运算
for i in range(M):
    for l in range(N):
        for j in range(K):
            C[i][j]+=A[i][l]*B[l][j]
#结束时间
time2=time.time()
print("Used time:",(time2-time1).__round__(6),"s")
```

Java:

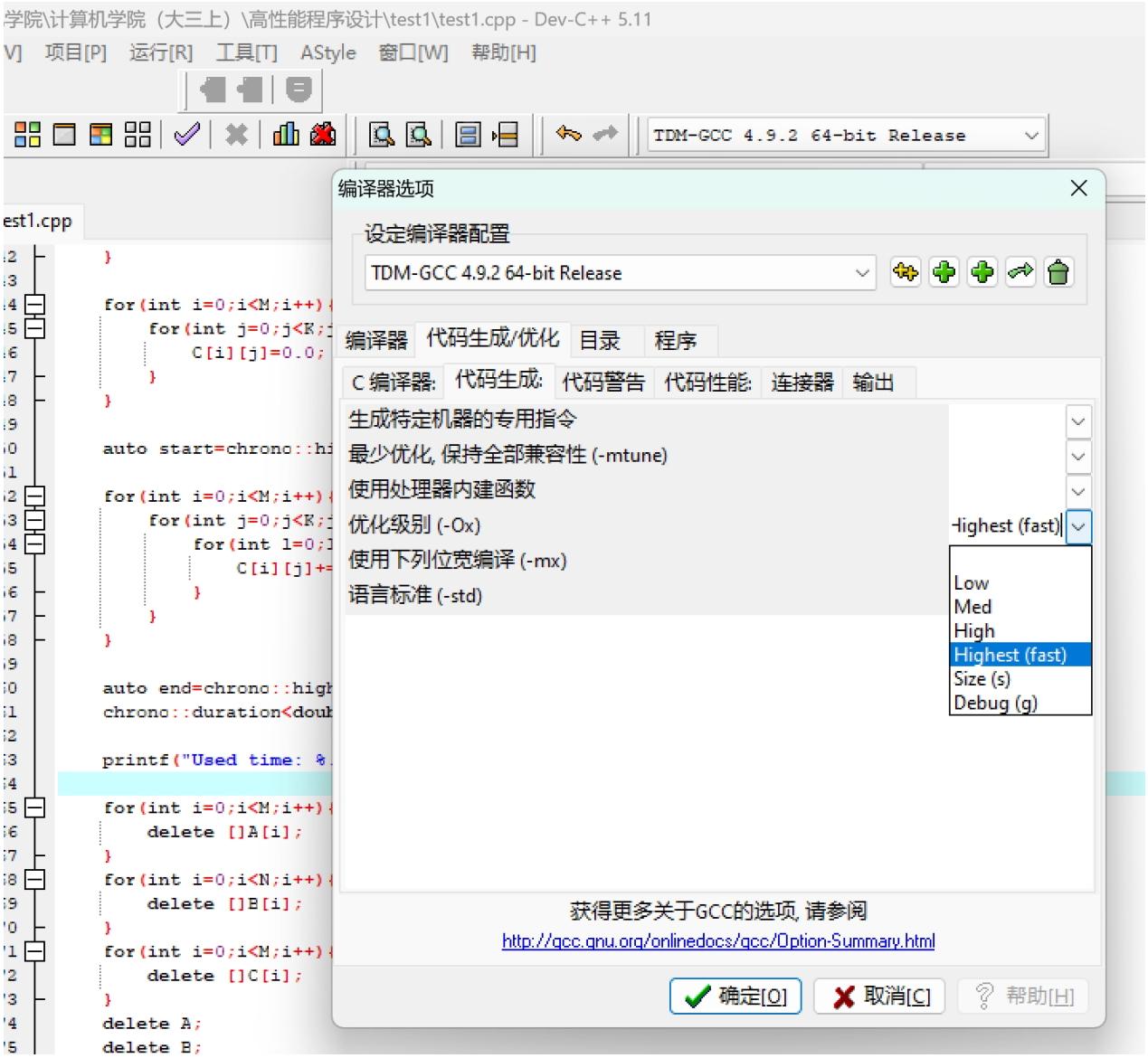
```
import java.util.Scanner;
import java.text.DecimalFormat;

public class test1 {
    public static void main(String[] args){
        Scanner reader=new Scanner(System.in);
        int M=reader.nextInt();
        int N=reader.nextInt();
        int K=reader.nextInt();
        //分配空间
        double A[][]=new double[M][N];
        double B[][]=new double[N][K];
```

```
        double C[][]=new double[M][K];
        //初始化
        for(int i=0;i<M;i++){
            for(int j=0;j<N;j++){
                double num1=Math.random();
                A[i][j]=num1;
            }
        }

        for(int i=0;i<N;i++){
            for(int j=0;j<K;j++){
                double num2=Math.random();
                B[i][j]=num2;
            }
        }

        for(int i=0;i<M;i++){
            for(int j=0;j<K;j++){
                C[i][j]=0.0;
            }
        }
        //开始时间
        double startTime=System.nanoTime();
        //矩阵运算
        for(int i=0;i<M;i++){
            for(int j=0;j<K;j++){
                for(int l=0;l<N;l++){
                    C[i][j]+=A[i][l]*B[l][j];
                }
            }
        }
        //结束时间
        double endTime=System.nanoTime();
        double Usedtime=endTime-startTime;

        DecimalFormat df=new DecimalFormat("#.######");
        System.out.println("Used time: "+df.format(Usedtime/1e9)+" s");
    }
}
```

以上代码均使用 O(n^3)的时间复杂度，可以观察到循环顺序从外到内分别遍历 i、j、l，并没有充分利用局部性原理。已知计算机中程序一般是按照顺序

执行的，访问 C[i][j]后，如果访问临近单元 C[i][j+1]会大大缩短运行时间，因此我们可以改变循环的顺序变成 i、l、j，可以在后续的结果中发现，改变顺序后运行时间有一定减少。

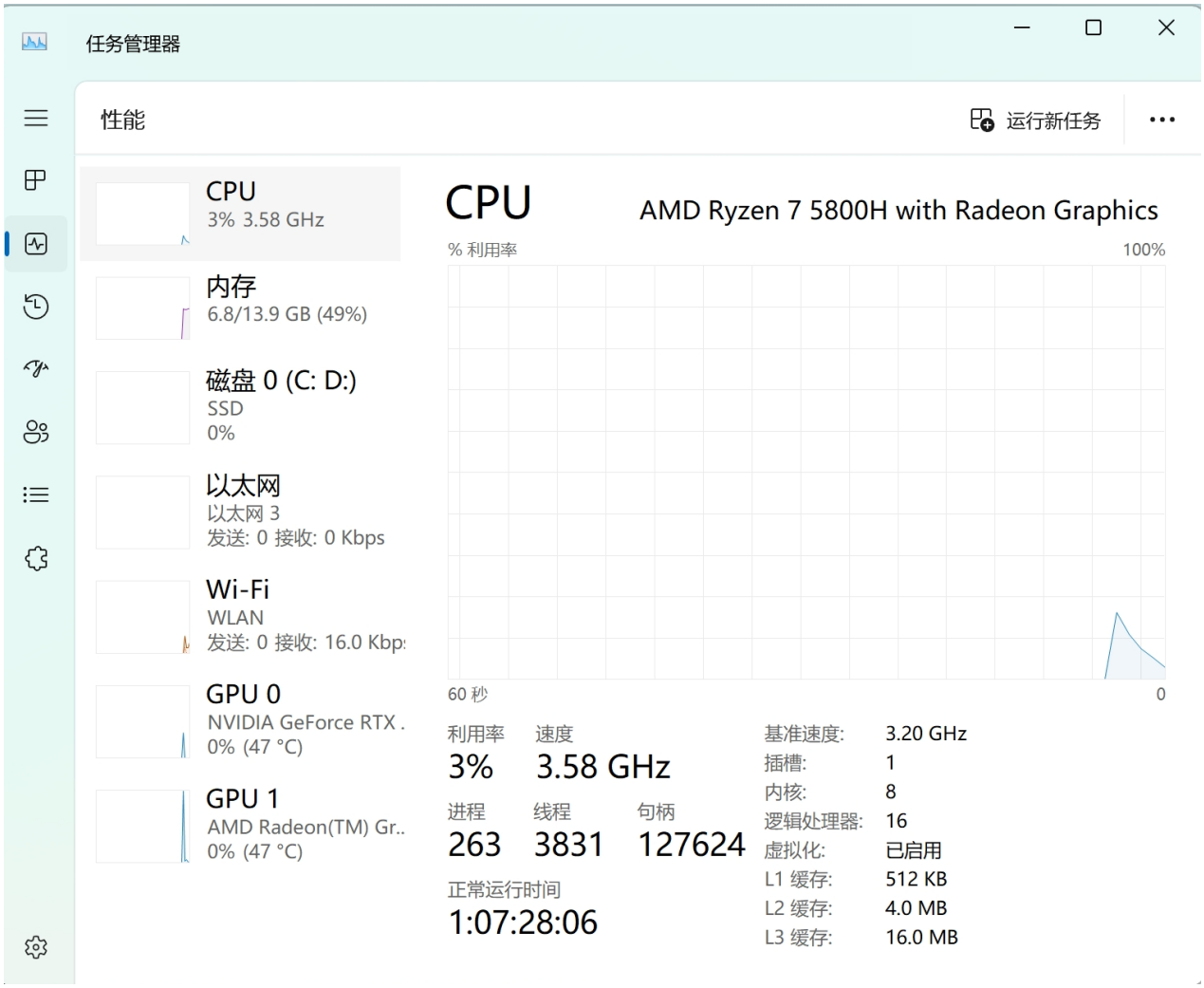
对于 C 语言的代码，可以通过编译优化，提高运行速度（如下图）：



对于 Python 中的 numpy 库，发现运行矩阵乘法的速度远远高于 C 语言的编译优化，原因可能在于 numpy 库中的运算实现可能是基于 C 语言，并且是经过多种优化方式后才进行封包，所以运行速度很快。

3.实验结果

实验所用计算机CPU信息如下：3.2GHz*16=51.2GFLPS



设定A、B、C都是800X800的矩阵。以下单位都为秒。调整循环顺序都由循环i、j、l -> i、l、j。浮点计算次数为2*(800^3)=1.024e9

版本	实现	运行时间(s)	相对加速比(相对前一版本)	绝对加速比(相对版本1)	浮点性能(GFLOPS)	达到峰值性能的百分比
1	Python	107.730488	1.000	1.000	0.00951	0.0186%
2	Python 调整循环顺序	112.849389	0.955	0.955	0.00907	0.0177%

3	Java	1.942460	58.096	55.461	0.52717	1.0296%
4	Java 调整循环顺序	1.448411	1.341	74.378	0.70698	1.3808%
5	C	1.780343	0.814	60.511	0.57517	1.1234%
6	C 调整循环顺序	1.282528	1.388	83.999	0.79842	1.5594%
7	+编译优化	0.358626	3.576	300.398	2.85534	5.5768%
8	Python 的 numpy 运算	0.029033	12.352	3710.622	35.27021	68.8871%

4.实验感想

在这次实验中，我对计算性能方面的知识有了一些基本的了解，学习了多种代码优化方案，并且自学 Java 完成矩阵乘法的实现。当然也遇到了很多问题，首先在编写 C 语言代码时，由于太久没有实践，在最开始时出现了矩阵规模超过 300 就会崩溃的问题，后来觉得应该是计算机性能不足，不能够一次性开辟很大的空间，后来修改成指针实现，经过验证，能够成功运行。总的来说，这次的实验给我带来了很大的收获。