中山大学计算机院本科生实验报告

（2022 学年秋季学期）

课程名称：并行程序设计　　批改人：

实验	**并行程序设计（0）：通用矩阵乘法**	专业（方向）	**计算机科学与技术（人工智能与大数据方向）**
学号	**20337025**	姓名	**崔璨明**
Email	**897369624@qq.com**	完成日期	**2023/3/5**

1. 实验目的

（1）用 C/C++语言实现通用矩阵乘法：

输入：M , N, K 三个整数（512 ~2048）

问题描述：随机生成 M*N 和 N*K 的两个矩阵 A,B,对这两个矩阵做乘法得到矩阵 C。

输出：A,B,C 三个矩阵以及矩阵计算的时间。

采用多种语言实现矩阵乘法，通过编译器优化，填写对应的表格。

（2）根据表格中的要求，采用 python、java 语言来实现通用矩阵乘法。

（3）对 C 语言实现的通用矩阵乘法进行优化。

2. 实验过程和核心代码

实验在 linux 环境下进行，虚拟机系统为 ubuntu18.04。

2.1 C 语言实现

首先输入 M，N，K 三个整数，然后调用 init_Mat()函数对 A，B，C 三个矩阵进行初始化，A 矩阵初始化为 M*N 的区间为[0,100]的浮点数矩阵，B 矩阵初始化为 N*K 的区间为[0,100]的浮点数矩阵，C 矩阵为 M*K 的全零矩阵。关键代码如下：

```
01. void init_Mat(int M,int N,int K){
02. 
03.     srand(233);
04. 
05.     for (int m=0;m<M;m++){
06.         for(int n=0;n<N;n++){
07.             A[m][n]=(double)(rand()%1000/10.0);
08.             printf("%f",A[m][n]);
09.         }
10.     }
11. 
12.     for (int n=0;n<N;n++){
13.         for(int k=0;k<K;k++){
14.             B[n][k]=(double)(rand()%1000/10.0);
15.         }
16.     }
17. 
18.     for (int m=0;m<M;m++){
19.         for(int k=0;k<K;k++){
20.             C[m][k]=0;
21.         }
22.     }
23. }
```

然后根据矩阵乘法公式进行计算并计时，公式如下：

$$C_{ij}=\sum_{k=0}^{n}a_{ik}b_{kj}$$

该部分的关键代码如下：

```
start_time=clock();
for(int m=0;m<M;m++){
    for(int k=0;k<K;k++){
        for(int n=0;n<N;n++){
            C[m][k]+=A[m][n]*B[n][k];
        }
    }
}
end_time=clock();
```

2.2 python 实现

python 的实现思路和上述的 C 语言实现相同，在此就不再赘述。这里我采用的储存矩阵的数据结构是二维列表，速度可能较 numpy 慢。具体代码如下：

```
import random
import time

M,N,K=input("input three integer(512 ~2048):\n").split()

M=int(M)
N=int(N)
K=int(K)

x=[[100.0*random.random()
                for row in range(N)]
                for col in range(M)]
y=[[100.0*random.random()
                for row in range(K)]
                for col in range(N)]
z=[[0.0 for row in range(K)]
                for col in range(M)]

print("matrix_1:\n",x)
print("matrix_2\n",y)

start = time.clock()
for m in range(M):
    for k in range(K):
        for n in range(N):
            z[m][k]+=x[m][n]*y[n][k]
end = time.clock()

print("result:\n",z)
print("using time:",1000*(end-start),'ms\n' )
```

2.3 java 实现

java 的实现思路和上述的 C 语言实现相同，在此也不再赘述。具体代码如下（只展示了关键部分）：

```
public static void main(String[] args) {
...
    double[][] a =new double[x][y];
    double[][] b =new double[y][z];
    double[][] c =new double[x][z];

        Random random = new Random();

    System.out.println();
        System.out.println("matrix_1:");
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                a[i][j] =100* random.nextDouble();
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
```

```
18.
19.     System.out.println();
20.         System.out.println("matrix_2:");
21.         for (int i = 0; i < b.length; i++) {
22.             for (int j = 0; j < b[i].length; j++) {
23.                 b[i][j] = 100* random.nextDouble();
24.                 System.out.print(b[i][j] + " ");
25.             }
26.         System.out.println();
27.         }
28.
29.     long start=System.currentTimeMillis();
30.
31.     for (int i = 0; i < a.length; i++) {
32.         for (int j = 0; j < a[i].length; j++) {
33.             for (int k = 0; k < b[j].length; k++) {
34.                 c[i][k] += a[i][j] * b[j][k];
35.             }
36.         }
37.     }
38.     long end=System.currentTimeMillis();
39. ...
40. }
```

2.4 调整循环顺序

以 C 语言实现的矩阵乘法为例进行优化，调整循环：

```
01. start_time=clock();
02. for(int m=0;m<M;m++){
03.     for(int n=0;n<N;n++){
04.         for(int k=0;k<K;k++){
05.             C[m][k]+=A[m][n]*B[n][k];
06.         }
07.     }
08. }
09. end_time=clock();
```

2.5 编译优化

C/C++编译器提供了一系列的编译选项，以下不同等级的编译选项为不同编译优化参数：

优化等级	含义
-O0	在不影响编译速度的前提下，尽量采用一些优化算法降低代码大小和可执行代码的运行速度。
-O1	同上。
-O2	牺牲部分编译速度，除了执行-O1 所执行的所有优化之外，还会采用几乎所有的目标配置支持的优化算法，用以提高目标代码的运行速度。
-O3	该选项除了执行-O2 所有的优化选项之外，一般都是采取很多向量化算法，提高代码的并行执行程度，利用现代 CPU 中的流水线，Cache 等。

3. 实验结果

统一采用 M=600,N=700,K=800 的输入来进行测试。

c 语言实现：

```
.89 1884002.77 1827637.62 1837921.37 1838010.00 1881300.37 1855991.49 1817596.92
 1744222.41 1879169.90 1822994.84 1816744.65 1902017.92 1888474.40 1818500.35 18
83251.21 1844507.63 1821764.60 1849781.65 1804431.99 1884031.26 1920266.66 18540
29.57 1892912.05 1830982.75 1789145.11 1795353.94 1812913.40 1840489.13 1855059.
93 1823138.16 1801449.59 1944166.42 1760324.72 1783308.46 1780233.31 1834782.59

using time: 4812.074000  ms
cui@cui-VirtualBox:~/parall/lab1$
```

python 实现：

```
844825.7056416539, 1820600.032457769, 1840218.9216365581, 1750954.699279489, 179
9701.7765008858, 1753282.2994128717, 1811159.9713085904, 1833477.7006516883, 177
2178.6054021153, 1765435.04265395, 1769433.3898474828, 1797495.1484649568, 17830
62.7868117657, 1777590.6751179371, 1768512.2927845104, 1809966.1862275382, 18384
65.4924103399, 1785571.6889660296, 1750391.370759397, 1738755.085346037, 1842256
.2024530172, 1791514.410975016, 1767317.2652243269, 1878932.490881888, 1722914.6
84111853, 1787166.9963620184, 1808736.6638874612, 1776606.2895416908, 1799782.26
49537504, 1749897.0643098978]]
using time: 110146.429 ms
```

java 实现：

```
3897 1688006.0968221573 1735670.764157566 1760339.4938341165 1693077.4299698472
1696134.9863400038 1732452.5233838018 1714653.8501297696 1646956.935741885 15847
00.9994326422 1713059.8190890318 1677429.386977579 1673004.9126475519 1741041.79
645063 1680395.8031893815 1735523.3982684459 1697612.453195415 1687462.339235138
8 1827267.7990156903 1692784.4216341255 1759007.5162718885 1729013.0905249072 18
21825.8190851717 1658697.7902017129 1744272.4092384118 1703005.452903149 1709805
.7927661445 1653417.9624656292 1719566.2112615719
using time:428 ms

cui@cui-VirtualBox:~/parall/lab1$
```

c 语言+调整循环顺序：

```
5 1923616.42 1948932.94 1897769.02 1958253.02 1925729.32 1854447.69 1883289.18 1
914956.46 1801378.27 1838625.05 1842103.42 1855233.17 1812769.84 1877422.58 1801
195.45 1948367.59 1779312.57 1858715.20 1877089.79 1819311.61 1841365.12 1817686
.89 1884002.77 1827637.62 1837921.37 1838010.00 1881300.37 1855991.49 1817596.92
 1744222.41 1879169.90 1822994.84 1816744.65 1902017.92 1888474.40 1818500.35 18
83251.21 1844507.63 1821764.60 1849781.65 1804431.99 1884031.26 1920266.66 18540
29.57 1892912.05 1830982.75 1789145.11 1795353.94 1812913.40 1840489.13 1855059.
93 1823138.16 1801449.59 1944166.42 1760324.72 1783308.46 1780233.31 1834782.59

using time: 1440.436000  ms
cui@cui-VirtualBox:~/parall/lab1$
```

c 语言+编译优化：

```
914956.46 1801378.27 1838625.05 1842103.42 1855233.17 1812769.84 1877422.58 1801
195.45 1948367.59 1779312.57 1858715.20 1877089.79 1819311.61 1841365.12 1817686
.89 1884002.77 1827637.62 1837921.37 1838010.00 1881300.37 1855991.49 1817596.92
 1744222.41 1879169.90 1822994.84 1816744.65 1902017.92 1888474.40 1818500.35 18
83251.21 1844507.63 1821764.60 1849781.65 1804431.99 1884031.26 1920266.66 18540
29.57 1892912.05 1830982.75 1789145.11 1795353.94 1812913.40 1840489.13 1855059.
93 1823138.16 1801449.59 1944166.42 1760324.72 1783308.46 1780233.31 1834782.59

using time: 1409.803000  ms
cui@cui-VirtualBox:~/parall/lab1$
```

c 语言+编译优化+调整循环顺序：

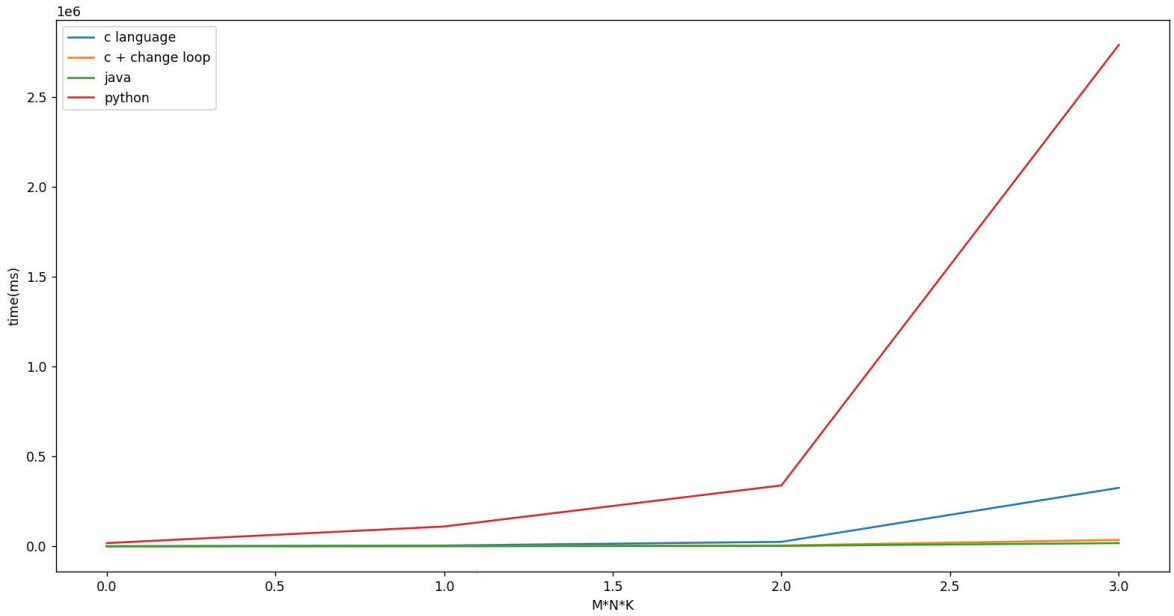
```
.89 1884002.77 1827637.62 1837921.37 1838010.00 1881300.37 1855991.49 1817
 1744222.41 1879169.90 1822994.84 1816744.65 1902017.92 1888474.40 1818500
83251.21 1844507.63 1821764.60 1849781.65 1804431.99 1884031.26 1920266.66
29.57 1892912.05 1830982.75 1789145.11 1795353.94 1812913.40 1840489.13 18
93 1823138.16 1801449.59 1944166.42 1760324.72 1783308.46 1780233.31 18347

using time: 1327.239000  ms
cui@cui-VirtualBox:~/parall/lab1$
```

在我的虚拟机上，时钟频率是 2295.616MHz，CPU 数量是 4，每个 CPU 中的核数是 1，超线程数量为 1，CPU 为 AMD 锐龙 7 3750H 移动处理器，其浮点计算单元为 8，则浮点峰值性能估计为 73.46GFLOPS。

填写表格如下：

版本	实现	运行时间（ms）	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	110146.43	1	1	0.0061	0.0083%
2	Java	428	257	257	1.57	2.1%
3	C	4812.07	0.08	23	0.139	0.19%
4	+调整循环顺序	1440	3.34	76	0.467	0.64%
5	+编译优化	1327.24	1.08	83	0.506	0.69%

除此之外，我还进行了多次实验，在每次实验中将 M，N，K 设置为不同的值（分别为 300*400*500，600*700*800，1024*1024*1024，2048*2048*2048），绘制图像如下：



4. 实验感想（200 字以内）

这是第一次并行程序设计的实验，通过这次实验，我对矩阵乘法的实现和如何优化有了更加深入的了解。除此之外，在本次实验中也遇到了一些问题，如计算时间的模块放置位置不对导致时间计算错误、对 java 语言的不熟悉导致花费了较多的时间在 debug 上等等。

这次实验让我学到了很多，也让我明白，在程序设计的过程中，除了要保证程序的正确性，还要兼顾程序的性能和开销。在我以往的学习过程中，我更多是关注前者而往往会忽略了后者，在以后的学习工作中我会对程序正确性和性能给予同等的关注。