

编译原理实验报告



实验一、 程序设计语言认知实验报告

姓 名： 刘宇诺

班 级： 计算机 20-1

学 号： 20201210207

任课教师： 韩连金

目录

- 一、实验目的 3
- 二、实验内容 3
- 三、实验环境 3
- 四、数据准备 3
- 五、实验过程设计 4
- 六、程序设计及实现 4
- 七、实验结果及分析 8
- 八、实验心得体会 9

一、实验目的

了解程序设计语言的发展历史,了解不同程序设计语言的各自特点:感受编译执行和解释执行两种不同的执行方式,初步体验语言对编译器设计的影响,为后续编译程序的设计和开发奠定良好的基础。

二、实验内容

分别使用 C、Java、Python 实现矩阵乘法,对采用这几种语言实现的编程效率,程序的规模,程序的运行效率进行对比分析。方便起见,本次实验采用的都是 n 维方阵的矩阵乘法。同时为了保证实验的合理性,所有矩阵乘法均未采用其他库。

本次实验的方阵维度分别为:10、100、500、1000,为了减少误差,每组数据均重复十次并取时间的平均值。

三、实验环境

Linux 系统 (Ubuntu、Debian、Gentoo 等系统均可), gcc 7.3.0, g++ 7.3.0, Java JDK 1.8, Python 3.6(以上编译器\解释器均可使用更高版本)。

四、数据准备

本次实验数据均在 data 文件夹下,由该文件夹下的 produce_data.py 脚本生成,生成的数据范围为 $[-100, 100)$ 。

如需更多实验数据，可自行修改脚本内容进行数据生成。

文件名	含义
matrix10.txt	维度为 10 的方阵
matrix100.txt	维度为 100 的方阵
matrix500.txt	维度为 500 的方阵
matrix1000.txt	维度为 1000 的方阵

五、实验过程设计

为保证每一种编程语言的测试数据相同所以采用从同一文件读取矩阵数据的方式进行输入。

不用的编程语言的读入输入快慢不同所以测试时的计算的时间只计算矩阵计算函数运行的时间而不是整个程序运行的时间。

程序设计思想：设计一个可以计算方阵矩阵相乘的函数，这样可以利用这个函数进行多次试验；编写一个测试函数，函数实现对矩阵相乘计算函数的运行时间的计算，最后返回一个本次计算的时间；main 函数中先进行数据的读入，然后循环十次，每次把测试函数中返回来的时间记录到总时间中，且测试函数会打印出每次测试的时间，十次测试完成后计算并且打印出这个矩阵计算的十次平均值。

六、程序设计及实现

C++代码

```
#include <iostream>
#include <fstream>
#include <time.h>

using namespace std;

const int N = 1000;

int a[N][N];
int res[N][N];

void matrix()
```

```

{
for (int i = 0; i < N; i ++ )
{
    for (int j = 0; j < N; j ++ )
    {
        int tmp = 0;
        for (int k = 0; k < N; k ++ )
        {
            tmp += a[i][k] * a[k][j];
        }
        res[i][j] = tmp;
    }
}
}

double test()
{
time_t start, stop;
start = clock();
matrix();
stop = clock();
double t = (double)(stop - start) / CLOCKS_PER_SEC;
printf("%.4fs\n", t);
return t;
}

int main()
{
ifstream fin("D:/xjuexp/matrix1000");
for (int i = 0; i < N; i ++ )
{
    for (int j = 0; j < N; j ++ )
    {
        fin >> a[i][j];
    }
}
fin.close();

double sum = 0;
for (int i = 0; i < 10; i ++ )
{
    sum += test();
}
printf("avg: %.4fn", sum / 10.0);

```

```
return 0;
}
```

Java 代码

```
package compilers;

import java.awt.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class exp1 {
    public static void main(String[] args) throws IOException {
        BufferedReader bf = new BufferedReader(new FileReader("D:\\xjuexp\\matrix1000.txt"));
        String textLine = new String();
        String str = new String();

        while ((textLine = bf.readLine()) != null) {
            str += textLine + ",";
        }

        String[] numbers = str.split(",");
        int[][] a = new int[numbers.length][numbers.length];
        String[] stmp = null;
        for(int i = 0; i < numbers.length; i++) {
            stmp = numbers[i].split(" ");
            for(int j = 0; j < numbers.length; j++) {
                a[i][j] = Integer.parseInt(stmp[j]);
            }
        }

        Tool tool = new Tool();
        tool.exp(a);
    }
}

class Tool {
    void matrix(int a[], int res[][] ) {
        for (int i = 0; i < a.length; i ++ ) {
            for (int j = 0; j < a[0].length; j ++ ) {
                int tmp = 0;
                for (int k = 0; k < a[0].length; k ++ ) {
```

```

        tmp += a[i][k] * a[k][i];
    }
    res[i][j] = tmp;
}
}

double test(int a[], int res[]) {
    long start = System.nanoTime();
    matrix(a, res);
    long end = System.nanoTime();
    return (double) (end - start) / 1000000000L;
}

void exp(int a[]) {
    double sum = 0;
    int[] res = new int[a.length][a[0].length];
    for (int i = 0; i < 10; i++) test(a, res); // 先计算 10 次

    for (int i = 0; i < 10; i++) {
        double t = test(a, res);
        System.out.printf("%.4fs\n", t);
        sum += t;
    }
    System.out.printf("avg: %.4fs", (sum / 10));
}
}

```

Python 代码

```

import time
import numpy as np

N = 1000
a = [[0 for col in range(N)] for row in range(N)]
b = [[0 for col in range(N)] for row in range(N)]

def readfile():
    Efield = []
    with open("matrix1000.txt", 'r') as file_to_read:
        while True:
            lines = file_to_read.readline()
            if not lines:
                break

```

```

        E_tmp = [int(i) for i in lines.split()]
        Efield.append(E_tmp)
    Efield = np.array(Efield)
    return Efield

def mul():
    readfile()
    start = time.time()
    for d in range(0, N):
        for e in range(0, N):
            b[d][e] = 0
            for c in range(0, N):
                b[d][e] += (a[d][c] * a[c][e])
    end = time.time()
    t = end - start
    print("%.4f %t" % t)
    return t

def main():
    sum = 0
    for i in range(0, 10):
        sum += mul()
    print('avg=%.4f % (sum / 10));

if __name__ == '__main__':
    main()

```

七、实验结果及分析

C++											
矩阵	1	2	3	4	5	6	7	8	9	10	平均值
10	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
100	0.0010	0.0030	0.0020	0.0030	0.0020	0.0020	0.0020	0.0030	0.0020	0.0030	0.0023
500	0.2990	0.2920	0.2880	0.2990	0.2920	0.2960	0.2930	0.3010	0.2910	0.2910	0.2942
1000	2.3930	2.3760	2.3690	2.3660	2.3710	2.3750	2.3730	2.3760	2.3770	2.3860	2.3762

Java 为了避免这个误差，先计算十次不计入，让后再计算十次记录并求算平均值。

Java											
矩阵	1	2	3	4	5	6	7	8	9	10	平均值
10	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
100	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008
500	0.2259	0.2185	0.2256	0.2274	0.2267	0.2263	0.2198	0.2203	0.2195	0.2211	0.2231
1000	1.7148	1.6664	1.7121	1.7218	1.6860	1.6877	1.6701	1.6544	1.6770	1.6903	1.6881

Python											
矩阵	1	2	3	4	5	6	7	8	9	10	平均值
10	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
100	0.1070	0.1107	0.1126	0.1150	0.1120	0.1100	0.1100	0.1187	0.1100	0.1080	0.1104
500	17.0421	16.7066	16.9417	16.8498	16.8972	16.5321	16.8220	16.4450	16.5084	16.4937	16.7238
1000	142.1412	138.9009	138.7187	137.5370	139.0604	139.2212	143.2431	141.3080	143.4216	142.7620	140.6314

三种语言对同一组数据分别测试 10 次求取平均之后，速度的快慢是 Java > C++ > Python，Java 是编译-解释型语言，C++是编译型语言，Python 是解释型语言，可以看出解释型语言是相对编译型语言慢很多的。Java 的编译-解释型语言的特点是可以运行在多样的环境中。Python 作为解释型语言其跨平台性优于 C++。

八、实验心得体会

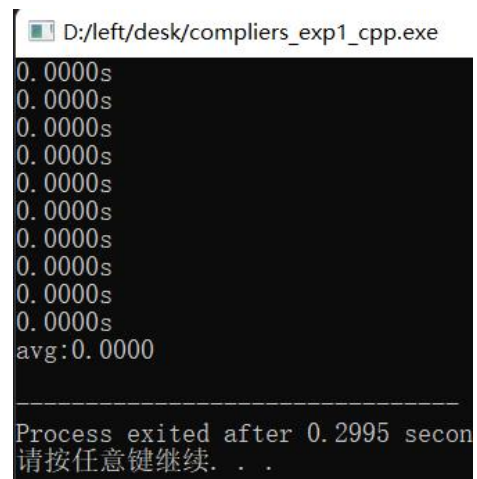
C++使用比较熟练没有遇到什么困难，在一次运行中循环十次得到一组测试数据，并求取平均值。在 Java 组的测试中，在一次运行中前两次计算得出的数据和后边得到的数据差别较大，而且除了前两次数据后边的数据都是稳定在一个数值左右，所以为了避免这个误差，先计算十次不计入，让后再计算十次记录并求算平均值。Python 计算 1000*1000 的矩阵时用的时间太长了，也体现了解释型语言的编译型高级语言在速度上的差别。

在追求速度的运算上用 C++和 Java 更好一些，在更大量的数据运算上应该是 C++的性能更优于 Java，Acm 比赛中对于大量数据的读入和运算 C++是明显优于 Java 的。在需要更好跨平台的软件时 Java 更好，既有一次编译无限次运行的优点又有 JVM 可以有更好的跨平台性能。而 Python 作为解释型语言更多的使用于数据处理，爬虫和人工智能等领域。

实验截图：

c++组的

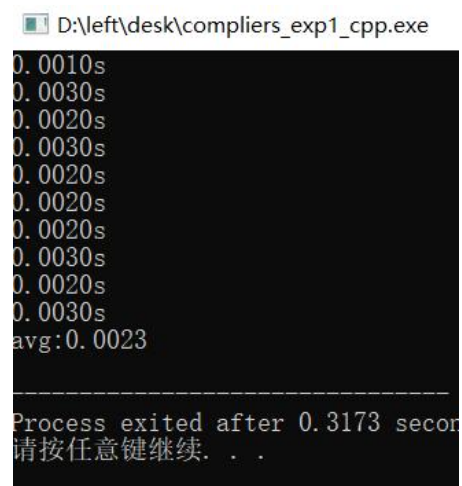
10



```
D:\left\desk\compliers_exp1_cpp.exe
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
avg:0.0000

-----
Process exited after 0.2995 seconds
请按任意键继续. . .
```

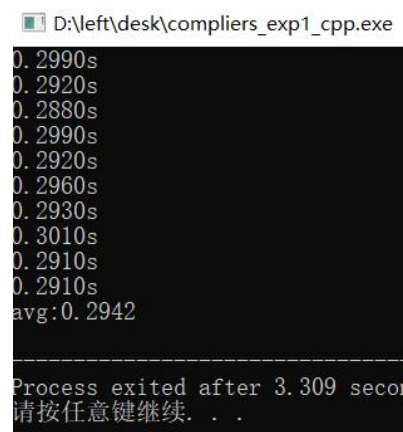
100



```
D:\left\desk\compliers_exp1_cpp.exe
0.0010s
0.0030s
0.0020s
0.0030s
0.0020s
0.0020s
0.0020s
0.0030s
0.0020s
0.0030s
avg:0.0023

-----
Process exited after 0.3173 seconds
请按任意键继续. . .
```

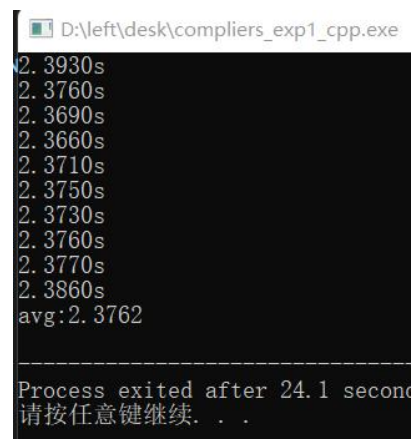
500



```
D:\left\desk\compliers_exp1_cpp.exe
0.2990s
0.2920s
0.2880s
0.2990s
0.2920s
0.2960s
0.2930s
0.3010s
0.2910s
0.2910s
avg:0.2942

-----
Process exited after 3.309 seconds
请按任意键继续. . .
```

10000

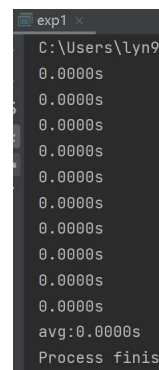


```
D:\left\desk\compliers_exp1_cpp.exe
2.3930s
2.3760s
2.3690s
2.3660s
2.3710s
2.3750s
2.3730s
2.3760s
2.3770s
2.3860s
avg:2.3762

Process exited after 24.1 second
请按任意键继续. . .
```

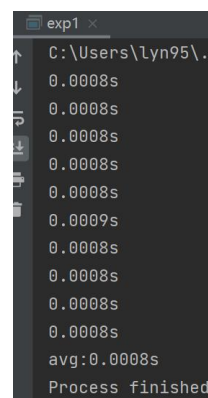
Java 组的

10



```
exp1
C:\Users\lyn9
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
0.0000s
avg:0.0000s
Process finished
```

100



```
exp1
C:\Users\lyn95\
0.0008s
0.0008s
0.0008s
0.0008s
0.0008s
0.0009s
0.0008s
0.0008s
0.0008s
0.0008s
0.0008s
avg:0.0008s
Process finished
```

500

```
exp1 x
C:\Users\lyn95\.jdk
0.2259s
0.2185s
0.2256s
0.2274s
0.2267s
0.2263s
0.2198s
0.2203s
0.2195s
0.2211s
avg:0.2231s
Process finished with
```

1000

```
exp1 x
C:\Users\lyn95\.jdk
1.7148s
1.6664s
1.7121s
1.7218s
1.6860s
1.6877s
1.6701s
1.6544s
1.6770s
1.6903s
avg:1.6881s
Process finished with
```

Python 组的

10

```
main x
D:\project\Python\test\venv\Scr
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
avg=0.0000
Process finished with exit code
```

100

```
main x
D:\project\Pytho
0.1070
0.1107
0.1126
0.1150
0.1120
0.1100
0.1100
0.1087
0.1100
0.1080
avg=0.1104
Process finished
```

500

```
n: main x
D:\project\Python\test\
17.0421
16.7066
16.9417
16.8494
16.8972
16.5321
16.8220
16.4450
16.5084
16.4937
avg=16.7238
Process finished with e
```

1000

```
main x
D:\project\Python\test\
142.1412
138.9009
138.7187
137.5370
139.0604
139.2212
143.2431
141.3080
143.4216
142.7620
avg=140.6314
Process finished with e
```