

# 编译原理

## 程序设计语言实验报告

李文彬, 1120173001

2020 年 3 月 3 日

### 1 实验目的

为了通过本次实验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。

### 2 实验内容

针对给定的实现矩阵与向量相乘功能, 分别使用 C、Java、Python 和 Haskell 实现该功能, 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。例如分别使用上述几种语言实现一个简单的矩阵乘法程序, 输入两个矩阵, 输出一个矩阵, 并分析相应的执行效果。

### 3 实验环境及硬件背景

#### 3.1 实验软件环境介绍

C: Dev C++

Java: IntelliJ IDEA Community Edition 2018 x64

Python: Anaconda3 2019, Spider

Haskell: WinGHCi

#### 3.2 实验硬件环境介绍

- CPU:
- Cache:

## 4 实验具体过程及步骤

### 4.1 主体算法描述

利用三个 for 循环，根据输入矩阵和输入向量的对应位置上的每个数字按照矩阵相乘法则进行计算，并存储入输出矩阵之中。

#### 4.1.1 数据规模

前三种语言均使用的是 500\*500 矩阵与 500\*1 的列向量相乘。最后一种语言由于输入限制，使用的是 20\*20 矩阵和 20\*1 的列向量相乘。

### 4.2 各语言实现代码

#### 4.2.1 C

> 预处理导入库

```
#include<stdio.h>
#include<stdlib.h>
#include <time.h>

int main()
{
    int vector[600]={0};
    int input[600][600]={0};
    int output[600]={0};
    int i,j,k,m;

    clock_t start, finish; //定义第一次调用CPU时钟单位的实际，可以理解为定义一个计数器
    double Total_time; //定义一个double类型的变量，用于存储时间单位
    start = clock(); //获取进入要测试执行时间代码段之前的CPU时间占用值
```

图 1

> 生成 500\*500 随机数组和 500\*1 随机列向量

```
//随机生成500*500的输入矩阵
for (i = 0; i < 500; ++i)
{
    for (j = 0; j < 500; ++j)
    {
        input[i][j] = rand() % 1000;
    }
}

//随机生成500的列向量
for (j = 0; j < 500; ++j)
{
    vector[j] = rand() % 1000;
}
```

图 2

> 主体代码

```
for(int m=0;m<1;m++)
{
    //计算输出矩阵
    for(i=0;i<500;i++)
    for(j=0;j<500;j++)
    {
        .....
        output[i]=output[i]+vector[j]*input[i][j];
    }
}

finish = clock();//停止计时

Total_time = (double)(finish-start)/CLK_TCK; //计算时间
```

图 3

> 调用及结果显示

```
//输出结果矩阵
for(k=0;k<500;k++)
printf("%d\n",output[k]);

//输出所用时间
printf("本次处理所用时间为: %1f ms\n",1000*Total_time);

return 0;

}
```

图 4

#### 4.2.2 Java

> 预处理导入库

```
import java.io.File;
import java.util.Random;
import java.io.*;
```

图 5

> 生成 500\*500 随机数组和 500\*1 随机列向量

```

public class matrix
{
    //生成随机矩阵
    public static void matrix_rand(int [][]mat,int rows,int cols)
    {
        Random rand = new Random();
        for(int i=0;i<rows;i++)
        {
            for(int j=0;j<cols;j++)
            {
                mat[i][j] = rand.nextInt(100)+1;
            }
        }
    }
}

```

图 6

> 主体代码

```

public static void writeFile(int [][]A, String filename,int rows,int cols) {
    try {
        File writeName = new File(filename);
        writeName.createNewFile(); // 创建新文件
        try (FileWriter writer = new FileWriter(writeName);
        ) {
            for(int i=0; i<rows;i++)
            {
                for(int j=0;j<cols;j++)
                {
                    writer.write(String.valueOf(A[i][j]));
                    writer.write(" ");
                }
                writer.write("\n");
            }
            writer.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

图 7

> 调用及结果显示

```

//矩阵与向量相乘
public static void matrix(int [][]A, int [][]B, int [][]C)
{
    for(int i=0;i<500;i++)
    {
        for(int j=0;j<1;j++)
        {
            C[i][j] = 0;
            for(int k = 0; k < 500; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}

```

图 8

### 4.2.3 Python

> 预处理导入库

```

1 import numpy as np
2 from numpy import random
3 import time

```

图 9

> 生成 500\*500 随机数组和 500\*1 随机列向量

```

input = random.randint(100,500,size=(500,500))
vector = random.randint(100,500,size=(500,1))
output = np.zeros((500,1))

np.savetxt("input.txt",input)
np.savetxt("vector.txt",vector)

```

图 10

> 主体代码

```

start_time = time.time()

for m in range(1):
    for i in range(500):
        for j in range(1):
            for k in range(500):
                output[i][j] += input[i][k]*vector[k][j]

finish_time = time.time()

```

图 11

> 调用及结果显示

```

np.savetxt("output.txt",output)
print("计算结束，结果存入输出文件中!")

result=1000*(finish_time-start_time)

print("本次处理所用时间:%f ms"%(result))

```

图 12

#### 4.2.4 Haskell

> 预处理导入库

```

import Data.Time.Clock

```

图 13

> 主体代码

```

matMul x y =
    let
        col m = [x|x:xs <- m]
        rights m = [xs|x:xs <- m,length(xs) > 0 ]
        rowMulMat r [] = []
        rowMulMat r m = sum(zipWith (*) r (col m)):(rowMulMat r (rights m))
    in case x of
        [r]      -> [rowMulMat r y]
        (r:rs)   -> (rowMulMat r y):(matMul rs y)

```

图 14

> 后续操作

```
main do
  start<-getCurrentTime
  operation
  end<-getCurrentTime
  print $ diffUTCTime    end start
```

图 15

## 5 运行效果截图

### 5.1 C

#### I First Running

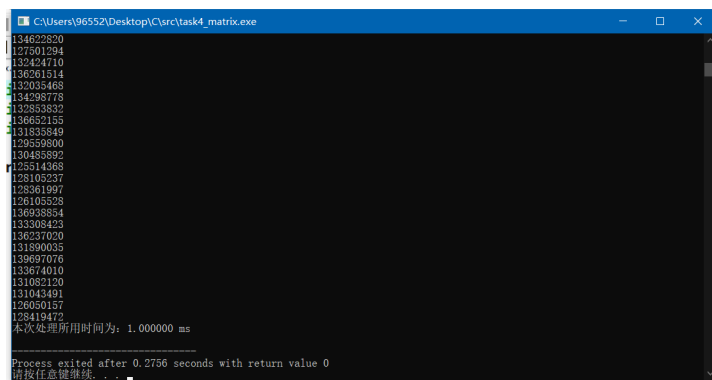


图 16

#### II Second Running

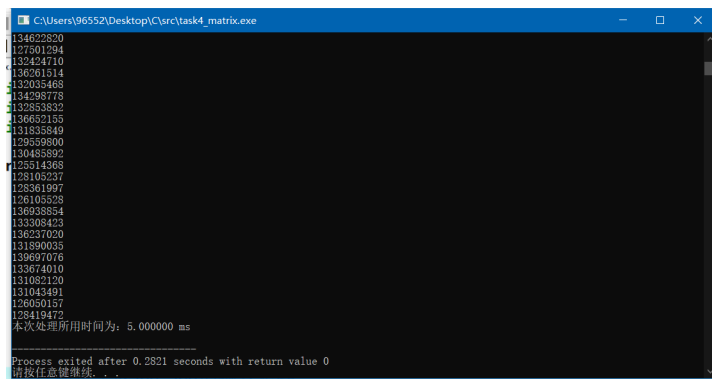
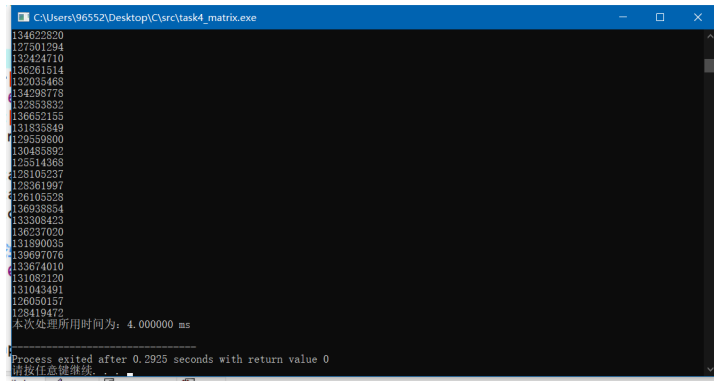


图 17

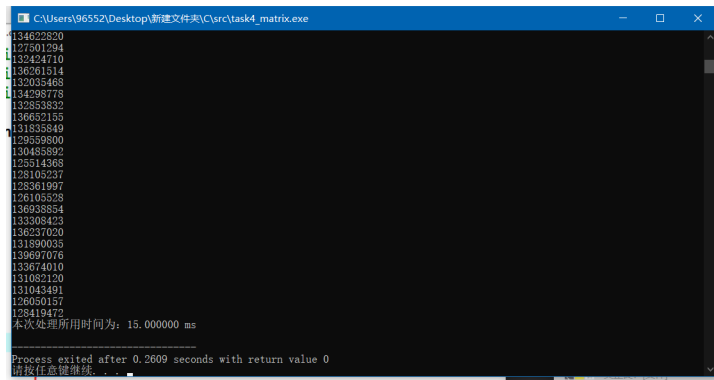
### III Third Running



```
C:\Users\96552\Desktop\src\task4_matrix.exe
134622820
127501294
132424710
136261514
132035468
134298778
132853832
136652155
131835849
129559800
130485892
125514368
128105237
128361997
126105528
136938854
133308423
136237020
131890035
139697076
133674010
131082120
131043491
126050157
128419472
本次处理所用时间为: 4.000000 ms
Process exited after 0.2925 seconds with return value 0
请按任意键继续...
```

图 18

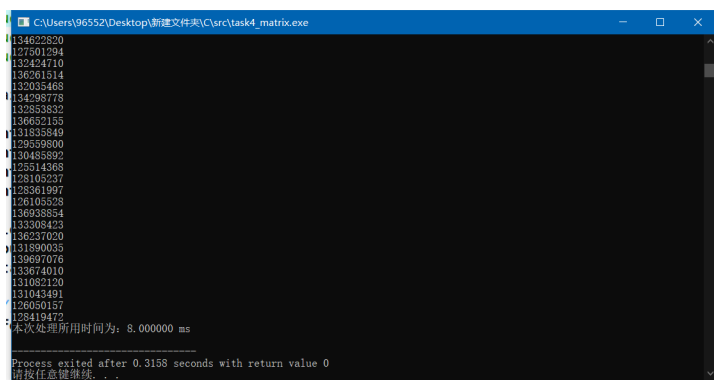
### IV Fourth Running



```
C:\Users\96552\Desktop\新建文件夹(C)\src\task4_matrix.exe
134622820
127501294
132424710
136261514
132035468
134298778
132853832
136652155
131835849
129559800
130485892
125514368
128105237
128361997
126105528
136938854
133308423
136237020
131890035
139697076
133674010
131082120
131043491
126050157
128419472
本次处理所用时间为: 15.000000 ms
Process exited after 0.2609 seconds with return value 0
请按任意键继续...
```

图 19

### V Fifth Running



```
C:\Users\96552\Desktop\新建文件夹(C)\src\task4_matrix.exe
134622820
127501294
132424710
136261514
132035468
134298778
132853832
136652155
131835849
129559800
130485892
125514368
128105237
128361997
126105528
136938854
133308423
136237020
131890035
139697076
133674010
131082120
131043491
126050157
128419472
本次处理所用时间为: 8.000000 ms
Process exited after 0.3158 seconds with return value 0
请按任意键继续...
```

图 20

## 5.2 Java

### I First Running



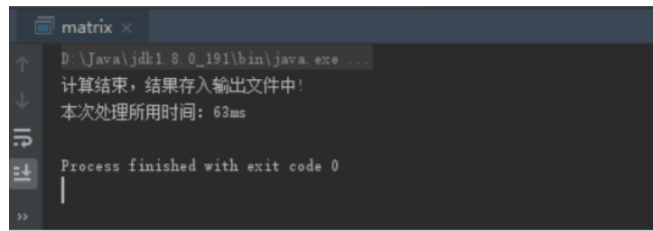


图 21

## II Second Running

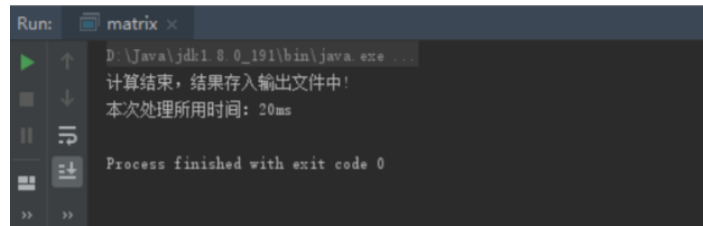


图 22

## III Third Running

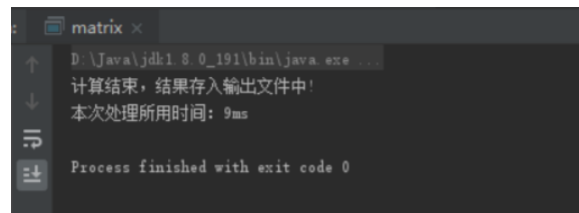


图 23

## IV Fourth Running

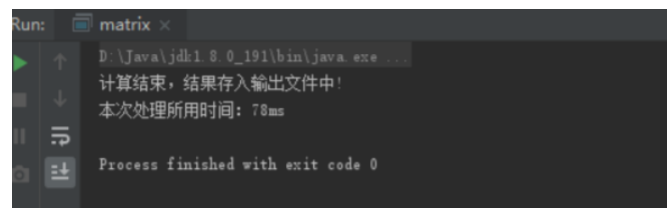


图 24

## V Fifth Running



图 25

## 5.3 Python

### I First Running

```
In [1]: runfile('C:/Users/96552/Desktop/Python/src/task4_matrix.py', wdir='C:/Users/96552/Desktop/Python/src')
计算结束，结果存入输出文件中!
本次处理所用时间: 272.281170 ms
```

图 26

### II Second Running

```
In [2]: runfile('C:/Users/96552/Desktop/Python/src/task4_matrix.py', wdir='C:/Users/96552/Desktop/Python/src')
计算结束，结果存入输出文件中!
本次处理所用时间: 260.321140 ms
```

图 27

### III Third Running

```
In [3]: runfile('C:/Users/96552/Desktop/Python/src/task4_matrix.py', wdir='C:/Users/96552/Desktop/Python/src')
计算结束，结果存入输出文件中!
本次处理所用时间: 265.290260 ms
```

图 28

### IV Fourth Running

```
In [4]: runfile('C:/Users/96552/Desktop/Python/src/task4_matrix.py', wdir='C:/Users/96552/Desktop/Python/src')
计算结束，结果存入输出文件中!
本次处理所用时间: 307.159424 ms
```

图 29

### V Fifth Running

```
In [5]: runfile('C:/Users/96552/Desktop/Python/src/task4_matrix.py', wdir='C:/Users/96552/Desktop/Python/src')
计算结束，结果存入输出文件中!
本次处理所用时间: 243.378639 ms
```

图 30

## 5.4 Haskell

### I First Running

```
%Main: start <getCurTime  
%Main: matMul[[1,2,3,4,5,6,7,8,9,10],[1,2,3,4,5,6,7,8,9,10],[1,2,3,4,5]  
[770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770],  
[770], [770], [770], [770], [770], [770], [770], [770]]  
%Main: end<getCurTime  
%Main: print $ diffUTCtime end start  
0.2809323s  
%Main:
```

图 31

## II Second Running

```
%Main) start <-getCurrentTime
%Main) matMul[[1,2,3,4,5,6,7,8,9,10],[1,2,3,4,5,6,7,8,9,10],[1,2,3,4,5]
%Main) [[770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770]]
%Main) end<-getCurrentTime
%Main) print $ diffUTCTime end start
0.3512964s
%Main)
```

图 32

### III Third Running

```

Main> start <-getCurrentTime
Main> matMul[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [1, 2, 3, 4, 5]
[[770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770]]
Main> end<-getCurrentTime
Main> print $ diffUTCTime end start
0.6029441s
Main>

```

图 33

#### IV Fourth Running

```
%Main> start <getCurTime
%Main> matMul [1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10], [1,2,3,4,5]
[770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770], [770]
%Main> end<getCurTime
%Main> print $ diffUTCTime end start
0.4066527s
%Main>
```

图 34

## V Fifth Running

[illegible]

图 35

## 6 语言易用性对比

语言	使用体验	易用性
C	较易上手，且由于学习时间较长，使用较为熟练	较易
Java	因为之前未接触过，但与 python 较为类似，经过资料查阅之后也相对较易上手	一般
Python	各种库功能强大，经过资料查阅周，较易写出代码	一般
Haskell	未接触过的语言形式，且语义较为难懂，上手相对较难	较难

图 36

## 7 程序规模对比分析

语言	代码行数	程序规模
C	58 行	一般
Java	83 行	较大
Python	27 行	较小
Haskell	13 行	较小

图 37

## 8 程序运行性能对比分析

语言	运行时间	运行性能
C	10.4ms	最优
Java	40.4ms	一般
Python	1377.5ms	较差
Haskell	9.98s（包含手动进行数据导入等操作时间）	最差

图 38

## 9 心得体会

本次实验让我更深入的理解了不同语言自身的特点，也更加深入的感受到了不同语言的代码规模与运行效率之间的差异。而对于 java 和 haskell 这两种语言的完全没有接触过的我来说，一边查阅资料一边进行代码编写更是一个不小的挑战，大大提升了自己的自学能力、资料查阅能力和有用信息筛选能力。同时，通过这次实验，我也认识对于已掌握 (或者说应该扎实掌握的)C 语言还有部分盲区，比如系统时间的获取，在这次实验前我从未使用过，但经过这次实验我已经掌握了 C 语言中对于部分代码块运行时

间的测量，这次实验使我更加扎实的掌握了已学过的知识，也让我感受到自己遇到问题，努力之后解决问题的喜悦。实践出真知，只有经过严谨的实验，才能让我们更好的掌握所学知识，也能更加深入的理解一些原理。