

编译原理与设计实验报告

姓名：卜梦煜 学号：1120192419 班级：07111905

1. 实验名称

程序设计语言认知实验

2. 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。

3. 实验内容

分别使用 C/C++、Java、Python 和 Haskell 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。

4. 实验环境

4.1 硬件配置

CPU 核数：8
CPU 主频：3.2GHz
内存：16GB
Cache：L1：512KB，L2：4MB，L3：16MB

4.2 软件配置

Visual Studio Code 1.64.2.0
C++：gcc 8.1.0
Python：python 3.9.7
Java：jdk-17.0.2
Haskell：ghc-8.0.2

5. 实现过程与步骤

四种语言在矩阵乘法功能的实现逻辑上是相似的，可以分为“输入模块”、“运算模块”、

“输出模块”、“计时模块”四个部分，输入模块从文件读入矩阵，运算模块进行矩阵乘法运算，输出模块将运算结果存入，计时模块记录程序前三模块运行时间，以毫秒为单位输出。

编写程序完成后，对编译型语言，C++ 需要利用 gcc 编译成.exe 文件，然后直接执行；Java 需要利用 jdk 编译生成.class 文件，然后在 JVM 运行。对解释型语言，Python 调用解释器 Python 解释执行文件代码。

Haskell 语言相比其他三种较特殊，既有编译器 ghc，也有解释器 ghci。编译运行时，调用编译器 ghc 编译生成.o 文件、.hi 文件、.exe 文件，然后直接执行.exe 文件；解释运行时，ghci 会解释.hs 文件，并加载 Main 模块。

5.1 C++

输入模块：利用 fstream 库打开文件“data.txt”，读入矩阵，以 int 型二维数组存储。

运算模块：按照矩阵乘法定义编写相应代码，复杂度为 $O(n^3)$ 。

输出模块：利用 fstream 库打开文件“result_C++.txt”，写入运算结果。

计时模块：利用 ctime 库记录程序开始时间、结束时间，相减得到程序运行时间。

5.2 Python

为更好地感受语言本身的效率，没有使用优化较好的 Numpy 库的 Matmul 函数，而是使用 Python 基本的列表操作进行运算。

输入模块：利用内置的 open() 函数打开文件“data.txt”，读入矩阵，以列表形式存储。

运算模块：按照矩阵乘法定义编写相应代码，复杂度为 $O(n^3)$ 。

输出模块：利用 open() 函数打开文件“result_python.txt”，写入运算结果。

计时模块：利用 time 模块记录程序开始时间、结束时间，相减得到程序运行时间。

5.3 Java

输入模块：利用 java.util.Scanner 类打开文件“data.txt”，读入矩阵，以 int 型二维数组存储。

运算模块：按照矩阵乘法定义编写相应代码，复杂度为 $O(n^3)$ 。

输出模块：利用 java.io 类打开文件“result_java.txt”，写入运算结果。

计时模块：利用 System 类记录程序开始时间、结束时间，相减得到程序运行时间。

5.4 Haskell

输入模块：利用 readFile 函数打开文件“data.txt”，编写函数读入矩阵，并处理成 Int 类型存储。

运算模块：按照矩阵乘法定义编写相应函数，复杂度为 $O(n^3)$ 。

输出模块：将结果加工成字符串，利用 writeFile 函数将运算结果写入文件“result_haskell.txt”。

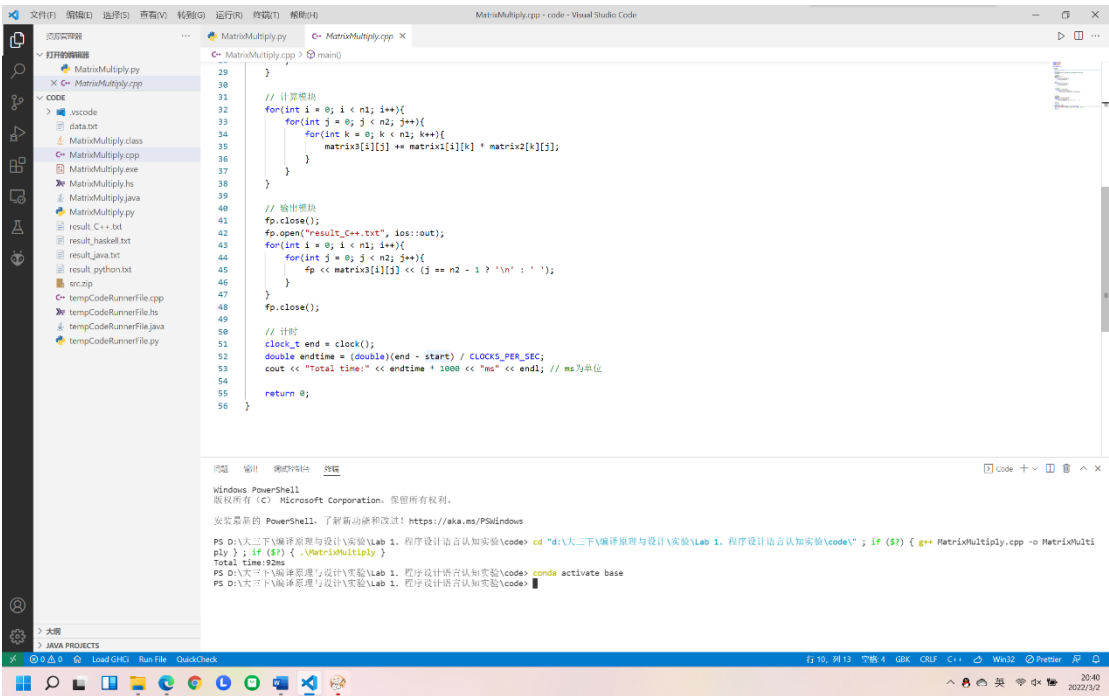
计时模块：利用 Data.Time.Clock 库记录程序开始时间、结束时间，利用 diffUTCTime 函

数得到程序运行时间。

6. 运行效果截图

本实验的输入矩阵为 300*300 的方阵，由随机数生成。

(1) C++运行 92ms，运行截图：

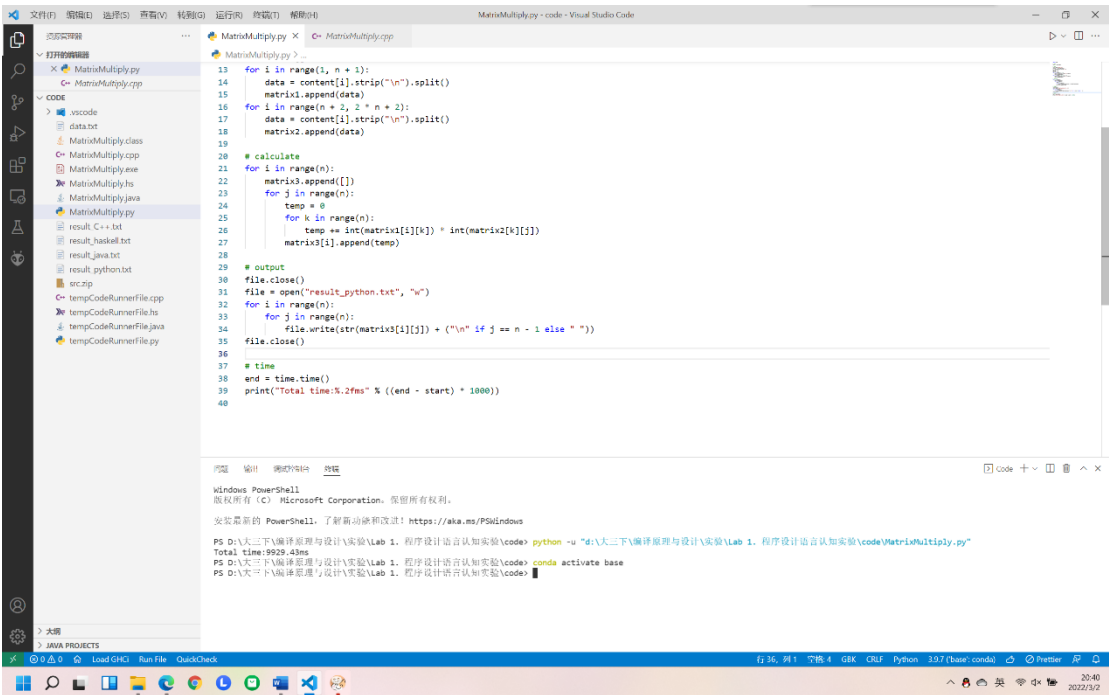


The screenshot shows the Visual Studio Code editor with a C++ file named `MatrixMultiply.cpp`. The code implements a naive matrix multiplication algorithm for two 300x300 matrices. It uses nested loops for the multiplication and a `clock_t` to measure the execution time. The output is written to `result.cpp.txt`. The terminal window at the bottom shows the command `g++ MatrixMultiply.cpp -o MatrixMulti` and the output `Total time:92ms`.

```
29 // 计算乘积
30 for(int i = 0; i < n1; i++){
31     for(int j = 0; j < n2; j++){
32         for(int k = 0; k < n2; k++){
33             matrix3[i][j] += matrix1[i][k] * matrix2[k][j];
34         }
35     }
36 }
37 // 输出乘积
38 fp.close();
39 fp.open("result.cpp.txt", ios::out);
40 for(int i = 0; i < n1; i++){
41     for(int j = 0; j < n2; j++){
42         fp << matrix3[i][j] << " ";
43     }
44     fp << "\n";
45 }
46 fp.close();
47 // 计时
48 clock_t end = clock();
49 double endtime = (double)(end - start) / CLOCKS_PER_SEC;
50 cout << "Total time:" << endtime * 1000 << "ms" << endl; // ms为单位
51 return 0;
52 }
```

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> cd "D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code" ; if (\$?) { g++ MatrixMultiply.cpp -o MatrixMulti
Total time:92ms
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> conda activate base
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code>

(2) Python 运行 9.93s，运行截图：



The screenshot shows the Visual Studio Code editor with a Python file named `MatrixMultiply.py`. The code implements a naive matrix multiplication algorithm for two 300x300 matrices. It uses nested loops for the multiplication and the `time` module to measure the execution time. The output is written to `result.python.txt`. The terminal window at the bottom shows the command `python -u "D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code\MatrixMultiply.py"` and the output `Total time:9929.43ms`.

```
13 for i in range(1, n + 1):
14     data = content[i].strip("\n").split()
15     matrix1.append(data)
16 for i in range(n + 2, 2 * n + 2):
17     data = content[i].strip("\n").split()
18     matrix2.append(data)
19 # calculate
20 for i in range(n):
21     matrix3.append([0] * n)
22     for j in range(n):
23         temp = 0
24         for k in range(n):
25             temp += int(matrix1[i][k]) * int(matrix2[k][j])
26         matrix3[i].append(temp)
27 # output
28 file.close()
29 file = open("result_python.txt", "w")
30 for i in range(n):
31     for j in range(n):
32         file.write(str(matrix3[i][j]) + (" \n" if j == n - 1 else " "))
33     file.close()
34 # time
35 end = time.time()
36 print("Total time:%.2fms" % ((end - start) * 1000))
37 
```

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> python -u "D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code\MatrixMultiply.py"
Total time:9929.43ms
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> conda activate base
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code>

(3) Java 运行 185ms，运行截图：

```
28 // 计算结果
29 int [][] matrix3 = new int[n1][n2];
30 for(int i = 0; i < n1; i++){
31     for(int j = 0; j < n2; j++){
32         matrix3[i][j] = 0;
33         for(int k = 0; k < n1; k++){
34             matrix3[i][j] += matrix1[i][k] * matrix2[k][j];
35         }
36     }
37 }
38
39 // 输出结果
40 File file_out = new File("result_java.txt");
41 OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream(file_out));
42 BufferedWriter bw = new BufferedWriter(osw);
43 for (int i = 0; i < n1; i++) {
44     StringBuffer sb = new StringBuffer();
45     for (int j = 0; j < n2; j++) {
46         sb.append(matrix3[i][j] + " ");
47     }
48     bw.write(sb.toString());
49 }
50 bw.close();
51
52 // 计时
53 long end = System.currentTimeMillis();
54 System.out.printf("Total time:%dms", end - start);
55 return;
56 }
57
58 }
```

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！<https://aka.ms/PSWindows>
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> cd "d:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code"; if (\$?) { javac MatrixMultiply.java }; if (\$?) { java MatrixMultiply.java }; if (\$?) { }
Total time:185ms
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> conda activate base
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> █

(4) Haskell 运行 3.14s，运行截图：

```
17 start <- getCurrentTime
18
19 -- 输入矩阵
20 let file_in = "data.txt"
21 content <- readFile file_in
22 let line = lines content
23 n = read (head line) :: Int
24 a = tail $ take (n + 1) line
25 b = drop (n + 2) line
26
27 -- 运算结果
28 let matrix1 = readMatrix a
29 matrix2 = readMatrix b
30 matrix3 = matrixMultiply matrix1 matrix2
31
32 -- 输出结果
33 -- unwords函数将输入列表连接成字符串
34 let temp = map (unwords . map show) matrix3
35 out = unlines temp
36 file_out = "result_haskell.txt"
37 writeFile file_out out
38
39 --计时
40 end <- getCurrentTime
41 print $ diffUTCTime end start
42
```

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！<https://aka.ms/PSWindows>
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> cd "d:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code"; if (\$?) { stack runhaskell MatrixMultiply.hs }; if (\$?) { }
3.137123s
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> conda activate base
PS D:\大三下\编译原理与设计\实验\Lab 1. 程序设计语言认知实验\code> █

7. 语言易用性与程序规模对比分析

7.1 语言易用性

学习难度方面。由于之前学过 C++、Python、Java，没学过 Haskell，网络上关于 C++、Python、Java 的教程远比 Haskell 的教程详细的多，并且 Haskell 为此前没有接触过的函数式编程。因此，从学习难度与学习成本上看，Haskell 的学习难度最大，学习成本最高，函数

式编程思想、Haskell 语法、Haskell 编译环境的安装都需要较多时间。

语言编程效率方面。从执行时间来看,由快到慢顺序依次为 C++、Java、Haskell、Python。

7.2 程序规模

考虑代码行数, C++为 43 行, Python 为 29 行, Java 为 47 行, Haskell 为 25 行。因此本实验中, C++、Java 程序规模较大, Python、Haskell 程序规模较小。

8. 程序运行性能对比分析

8.1 运行结果

实验条件: 输入矩阵为随机数生成 300*300 的方阵, 重复运行 10 次取平均值, 作为程序运行时间。实验中用到的 Python 为原生 Python, 并未使用 Numpy 库加速。

实验结果如下:

	C++/ms	Python/s	Java/ms	Haskell/s
1	92	10.12	175	2.53
2	91	10.17	181	3.23
3	92	9.95	178	3.23
4	91	10.23	176	3.24
5	90	10.25	176	3.22
6	92	10.01	175	3.19
7	91	10.5	178	2.89
8	92	10.13	177	3.12
9	91	10.56	173	2.95
10	91	10.5	177	2.74
平均	91.30	10.24	176.60	3.03

由结果可知, 运行速度由快到慢依次为 C++、Java、Haskell、Python。

8.2 结果分析

1. 编译型语言比解释型语言快。C++、Java、Haskell 为编译型语言, 程序运行计时并未包括编译二进制文件所需时间, 因此速度上比边解释边执行的 Python 更快。

2. C++比 Java 快。 C++编译生成的为.exe 文件为二进制文件, 可直接在机器上运行; 而 Java 编译生成的.class 文件需要在 JVM 虚拟机上运行, 因此 C++运行速度比 Java 快。

3. C++比 Haskell 快。查阅资料得知, 这与 CPU 在函数返回方面的优化有关。gcc 会调用指令集里现有的返回指令 ret, CPU 的分支预测可以准确预测返回地址, 流水线不会出现停滞情况; 而 ghc 并不能调用指令集原生的返回指令, 而是使用 jmp *指令完成, 导致 CPU 不能准确预测返回地址, 流水线经常停滞。这就导致 Haskell 运行速度比 C++慢。

9. 实验心得体会

通过这次实验, 我有如下收获:

1. 对解释型语言、编译型语言的执行过程有了更深刻的认识，尤其是不同编译型语言生成的中间文件，以及中间文件的运行方式。例如 C++ 与 Haskell 编译生成的.exe 文件是可以直接在机器上运行的，而 Java 编译生成的.class 文件必须在 JVM 虚拟机上运行。

2. 对四种语言有了更深入的了解，包括代码效率、语法特性、库的调用、运行环境配置等。程序运行速度不仅与运行方式有关，更与程序设计语言本身有关。例如，Haskell 函数式编程在表示循环时常用递归实现，而不恰当的递归可能导致运行时间大大增加；Java 对各种操作都有封装完整的模块，使用前需多了解，查阅相关文档。

3. 了解了函数式编程的思想与基本方法，能够编写简单代码。