实验一、语言认知实验报告

> *班级：07111801*
>
> *姓名：唐小娟*
>
> *学号：1120180207*

1. 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。

2. 实验内容

分别使用 C/C++、Java、Python 和 Haskell 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。在此实验中，为了方便，均采用n维方阵的矩阵乘法，方阵维度分别为：10、50、100、500、1000，运行时间以s为单位。

3. 实验环境

3.1 硬件配置信息

名称	参数
处理器	Intel(R) Core(TM) i7-10875H CPU
CPU主频	2.30GHz
RAM	16G
内核	8
逻辑处理器	16
L1缓存	512KB
L2缓存	2.0MB
L3缓存	16MB

3.2 程序设计语言的开发环境配置

语言	编译器/解释器版本
C++	gcc 8.1.0
Java	java 15.0.2
Haskell	ghc 8.10.4

语言	编译器/解释器版本
Python	Python 3.8.3

4. 实验过程

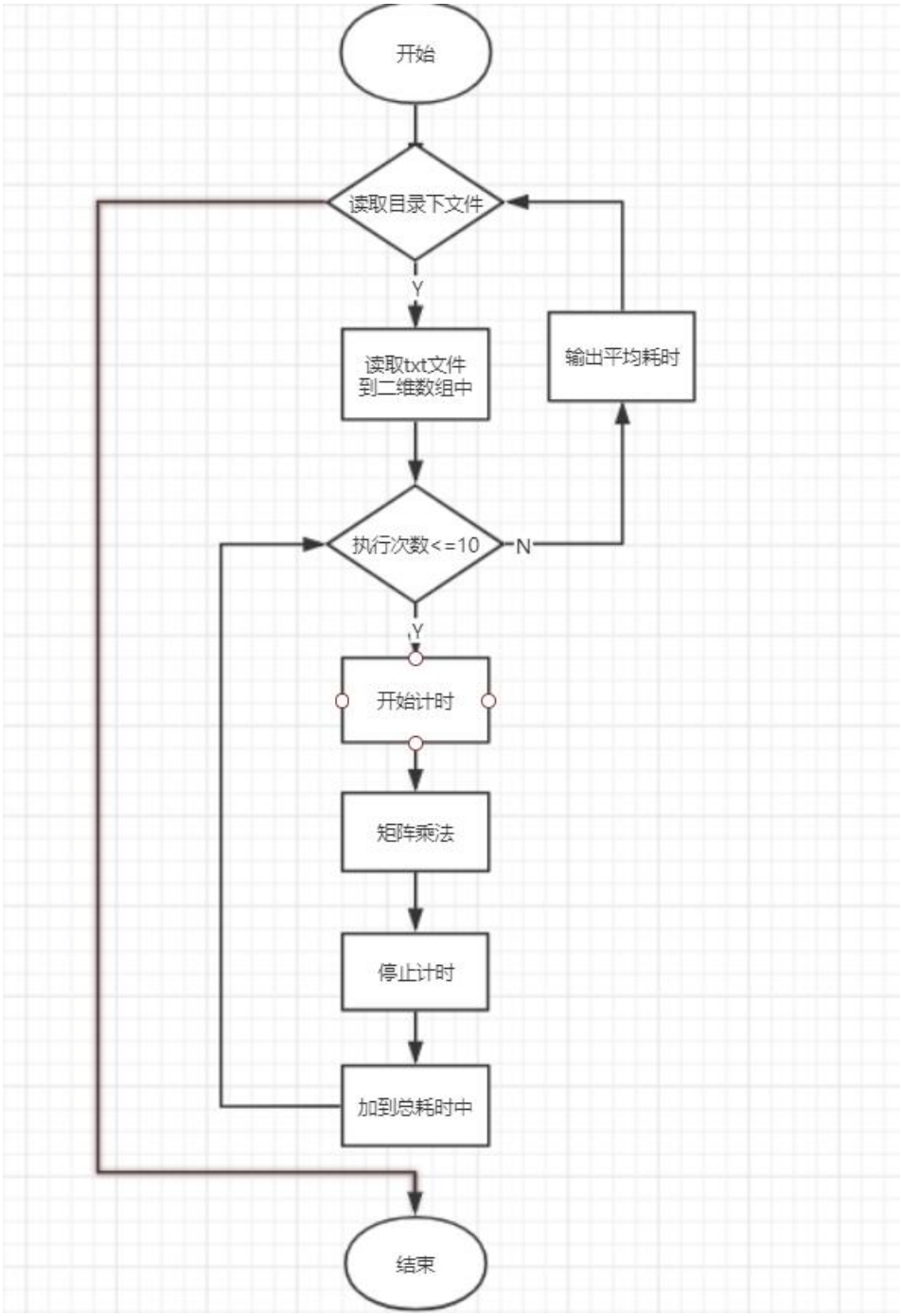
4.1 数据准备

实验数据均在`data`文件夹中，且是由`generate.py`的脚本生成的。

文件名	内容
matrix10.txt	维度为10的方阵
matrix50.txt	维度为50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

4.2 代码设计

4.2.1 流程图



4.2.2 流程说明

遍历`data`文件夹下的所有`matrix`数据，并且将`txt`内容读入到二维数组中，进行矩阵乘积运算，重复十次，记录每次运行时间，最终算出耗时平均值输出。

为了保证实验的可靠性，需要注意一下要点：

- 每组数据计算重复10次，取平均值减少实验误差；
- 在Java和C++中采用了动态数组读入`txt`文件，为了保证实验的合理性，计算的时候要把动态数组转化为普通数组；
- 计算过程不调用库；
- Haskell是惰性运算，为了保证测试时间的准确性，必须要严格强制计算结果。

4.2.3 矩阵运算

A矩阵[a,b]和B矩阵[b,c]运算得到C矩阵[a,c]，A中的每一行分别与B中的每一列相乘并计算总和得到C的一行数值，依次遍历A中的每一行，即可得到C矩阵的结果。

代码示例如下：

```
for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            ans[i][j] = 0;
            for (int k = 0; k < n; k++)
            {
                ans[i][j] += mat[i][k] * mat[k][j];
            }
        }
    }
```

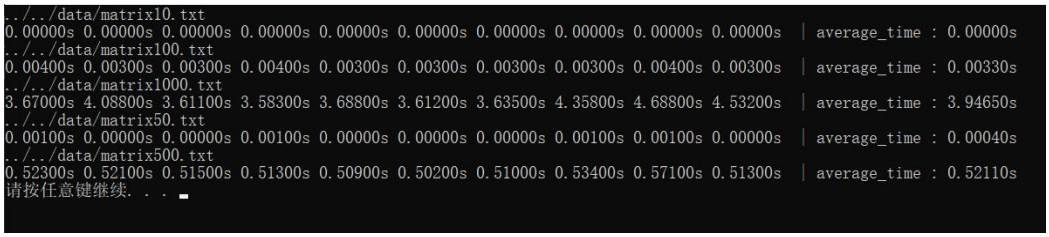
4.3 实验步骤

依次执行C++、Java、Python、Haskell的代码，观察执行结果，得到平均运行时间后统计结果，最后对这四种语言的特点进行分析。

5. 运行结果

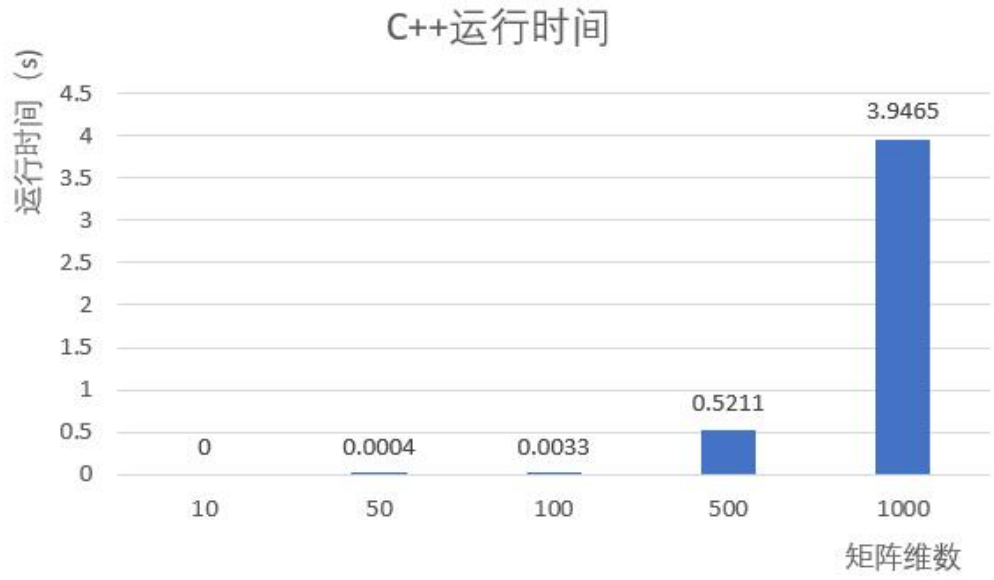
5.1 C++

运行结果



统计结果（单位：s）

矩阵维数	1	2	3	4	5	6	7	8	9	10	平均值
10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
50	0.00100	0.00000	0.00000	0.00100	0.00000	0.00000	0.00000	0.00100	0.00100	0.00000	0.00040
100	0.00400	0.00300	0.00300	0.00400	0.00300	0.00300	0.00300	0.00300	0.00400	0.00300	0.00330
500	0.52300	0.52100	0.51500	0.51300	0.50900	0.50200	0.51000	0.53400	0.57100	0.51300	0.52110
1000	3.67000	4.08800	3.61100	3.58300	3.68800	3.61200	3.63500	4.35800	4.68800	4.53200	3.94650



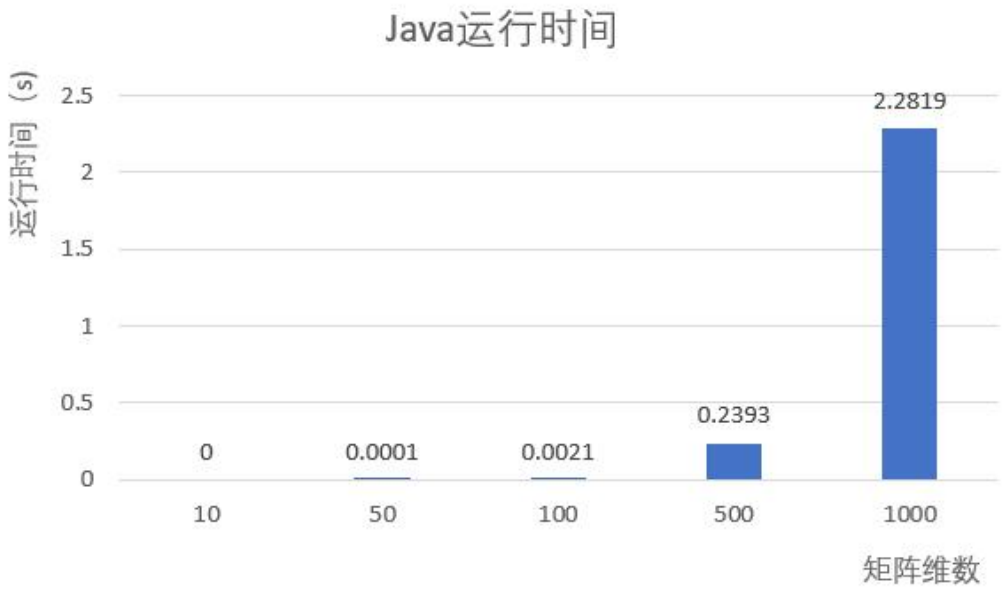
5.2 Java

运行结果

```
..\..\data\matrix10.txt
0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s  | average_time: 0.00000s
..\..\data\matrix100.txt
0.01200s 0.00100s 0.00100s 0.00100s 0.00100s 0.00100s 0.00100s 0.00100s 0.00100s 0.00100s  | average_time: 0.00210s
..\..\data\matrix1000.txt
2.25800s 2.35200s 2.18100s 2.26200s 2.63100s 2.17900s 2.17200s 2.31600s 2.19900s 2.26900s  | average_time: 2.28190s
..\..\data\matrix50.txt
0.00000s 0.00000s 0.00100s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s  | average_time: 0.00010s
..\..\data\matrix500.txt
0.25900s 0.24800s 0.24300s 0.24100s 0.23300s 0.24000s 0.23300s 0.23000s 0.23300s 0.23300s  | average_time: 0.23930s
```

统计结果（单位：s)

矩阵位数	1	2	3	4	5	6	7	8	9	10	平均值
10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
50	0.00000	0.00000	0.00100	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00010
100	0.01200	0.00100	0.00100	0.00100	0.00100	0.00100	0.00100	0.00100	0.00100	0.00100	0.00210
500	0.25900	0.24800	0.24300	0.24100	0.23300	0.24000	0.23300	0.23000	0.23300	0.23300	0.23930
1000	2.25800	2.35200	2.18100	2.26200	2.63100	2.17900	2.17200	2.31600	3.19900	2.26900	2.28190



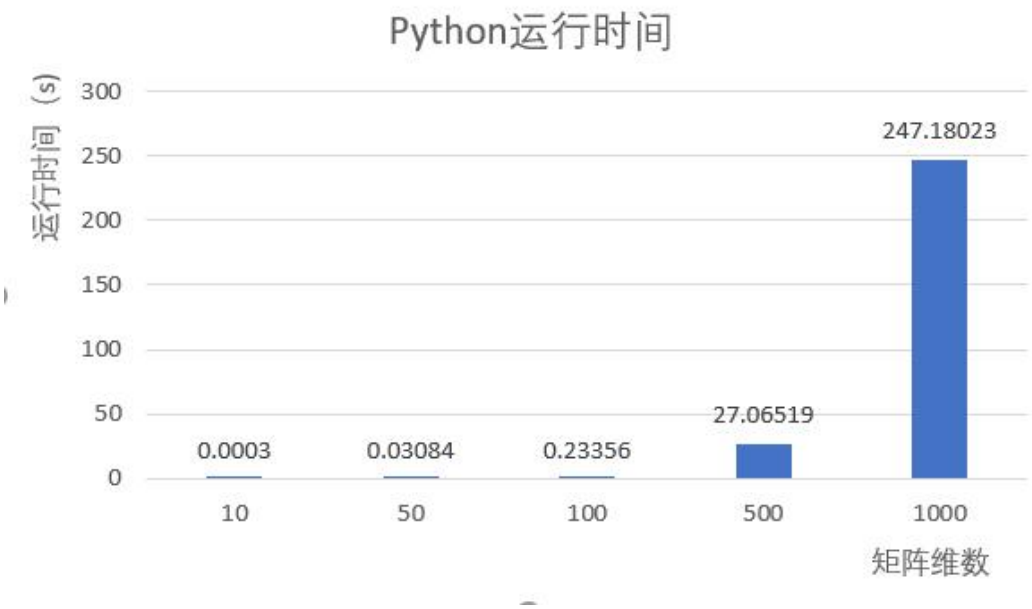
5.3 Python

运行结果

```
../../data/matrix10.txt
0.00000s 0.00000s 0.00100s 0.00000s 0.00000s 0.00100s 0.00000s 0.00000s 0.00000s 0.00100s  | average_time : 0.00030s
../../data/matrix100.txt
0.23884s 0.21995s 0.23738s 0.23737s 0.23347s 0.22649s 0.23038s 0.23117s 0.24036s 0.24019s  | average_time : 0.23356s
../../data/matrix1000.txt
251.24170s 250.52425s 251.93903s 221.22979s 223.38170s 251.66832s 259.49687s 247.92154s 254.64178s 259.75730s  | average_time : 247.18023s
../../data/matrix50.txt
0.04387s 0.02992s 0.02992s 0.02992s 0.02892s 0.02948s 0.02895s 0.02853s 0.02892s  | average_time : 0.03084s
../../data/matrix500.txt
30.38136s 28.26790s 25.82366s 26.72285s 26.67691s 27.26415s 26.15719s 26.64301s 26.35745s 26.35737s  | average_time : 27.06519s
```

统计结果（单位：s）

矩阵维数	1	2	3	4	5	6	7	8	9	10	平均值
10	0.00000	0.00000	0.00100	0.00000	0.00000	0.00100	0.00000	0.00000	0.00000	0.00100	0.00030
50	0.04387	0.02992	0.02992	0.02992	0.02992	0.02892	0.02948	0.02895	0.02853	0.02892	0.03084
100	0.23884	0.21995	0.23738	0.23737	0.23347	0.22649	0.23038	0.23117	0.24036	0.24019	0.23356
500	30.38136	28.26790	25.82366	26.72285	26.67691	27.26415	26.15719	26.64301	26.35745	26.35737	27.06519
1000	251.24170	250.52425	251.93903	221.22979	223.38170	251.66832	259.49687	247.92154	254.64178	259.75730	247.18023



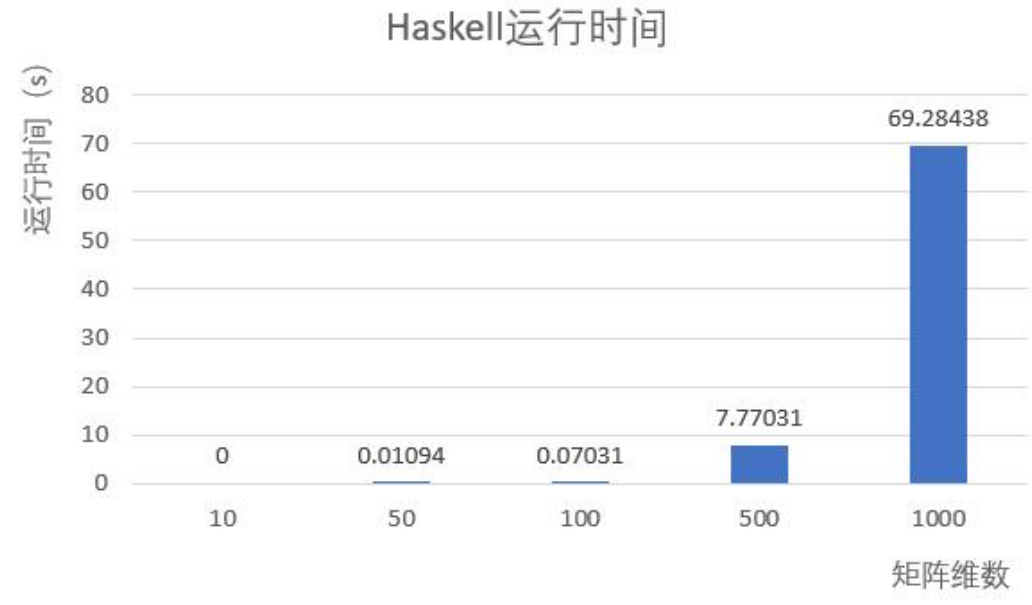
5.4 Haskell

运行结果

```
8.10938s 7.62500s 8.29688s 7.51562s 8.15625s 7.73438s 7.53125s 7.65625s 7.54688s 7.53125s   | average_time: 7.77031s
0.00000s 0.01562s 0.01562s 0.01562s 0.00000s 0.01562s 0.01562s 0.01562s 0.00000s 0.01562s   | average_time: 0.01094s
67.93750s 67.21875s 66.92188s 69.25000s 70.04688s 70.40625s 71.73438s 69.23438s 70.28125s 69.81250s   | average_time: 69.28438s
0.07812s 0.06250s 0.07812s 0.06250s 0.07812s 0.06250s 0.07812s 0.06250s 0.07812s 0.06250s   | average_time: 0.07031s
0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s 0.00000s   | average_time: 0.00000s
process over
```

统计结果（单位：s）

矩阵维数	1	2	3	4	5	6	7	8	9	10	平均值
10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
50	0.00000	0.01562	0.01562	0.01562	0.00000	0.01562	0.01562	0.01562	0.00000	0.01562	0.01094
100	0.07812	0.06250	0.07812	0.06250	0.07812	0.06250	0.07812	0.06250	0.07812	0.06250	0.07031
500	8.10938	7.62500	8.29688	7.51562	8.15625	7.73438	7.53125	7.65625	7.54688	7.53125	7.77031
1000	67.93750	67.21875	66.92188	69.25000	70.04688	70.40625	71.73438	69.23438	70.28125	69.81250	69.28438



6. 语言性能分析

6.1 语言易用性

由于本人对C++和Python比较熟悉，对Java和Haskell是完全没有接触的，在学习Haskell和Java的过程中，我认为Haskell的学习难度要远远大于Java。首先函数式编程的思想和以往的命令式编程思想完全不同，思维转变有些困难；其次它模式匹配、柯里函数、Monads都让我有些摸不着头脑，总之要改变以往的思维模式去接触这样一种函数式编程语言，学习曲线十分陡峭，个人认为它的易用性最差。

但是Java是一种完全面向对象的语言，而且和C++十分类似，由于大二已经掌握了面向对象编程的思想，所以Java上手起来比Haskell要容易很多。除此之外，Java没有C++的指针，提供了对内存的自动管理，从而大大减少了出错的概率；同时它的高移植性更是使代码运行带来了极大的方便。因此个人认为Java的易用性优于C++。

而Python相比较是一种十分灵活的接近自然语言的语言，学习比较容易。

总之个人认为这四门语言的易用性排序为：Python>Java>C++>Haskell

6.2 程序规模

程序规模主要考察分析使用不同语言实现同一种功能所得到的程序规模大小。所以根据程序实现，功能主要有：矩阵乘法、读取文件。

6.2.1 矩阵乘法

Python、Java、C++都是通过三个for循环进行求值运算，核心代码规模相差不大，而Haskell不用for循环，直接采用zipWith函数使得代码十分简洁，只需要一行代码。从矩阵乘法这一角度来说，Haskell的规模最小。

6.2.2 读取文件

Python读取txt数据到list中，可直接读取而不需要设置专门的文件类，同时可直接分隔读取的内容到list中，代码大概需要6行左右。

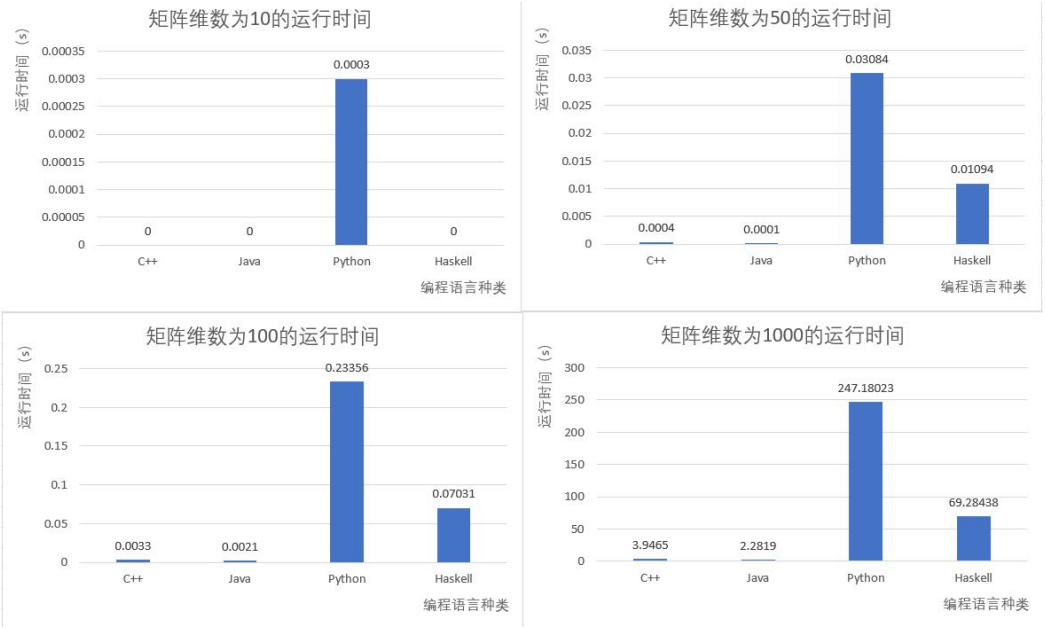
Haskell在遍历目录下文件时需要采用递归模式，而读取文件内容采用map函数以及words来分割字符串并转化为数字，代码需要3行。

相比较而言，Java和C++需要设置专门的文件类来读取，同时因为不知道矩阵的大小，需要设置动态数组，在计算矩阵乘法时，还需要将动态数组的值辅到普通数组中。除此之外，Java还设置专门的myMatrix类，这部分的代码量为18行。C++这部分的代码量大约为11行。

6.2.3 总体规模

根据6.2.1和6.2.2的探究，总之来说，四门语言的程序规模排序为：Java>C++>Python>Haskell。

6.3 程序运行性能



根据上图可知：

1、当方阵维数很小时，四种语言的运行时间相差不大；但维数越大，运行时间相差就越来越大了。

2、C++由于接近底层，性能是显而易见的，但是这里Java竟然比C++的运行效率略高，经过我查阅资料得：C++的编译器不如Java的编译器，因为java的编译器能针对CPU指令集进行优化，而C++的静态编译器难以做到。

3、总而言之：运行效率排序为：Java>C++>Haskell>Python

7. 心得体会

在这次实验中，由于haskell是从没有接触过的编程语言，首先配环境就花费了一大半天的时间，再者haskell的资料在网上相对比较少，所以在学习的过程中确实花了不少精力。尤其是haskell的惰性运算，在进行统计执行时间的时候，时间一直为0，后来查资料得知，haskell有一个特性就是只有在确实需要时才求值。这样导致的问题是调用函数计算矩阵，如果后面没有使用这个结果，计算过程就不会真正执行导致执行时间统计一直为0，为了解决这个问题，我使用了`deepseq`，这个函数可以对矩阵中的每一个元素强行求值。

再者，我加深了解释型语言和编译型语言的理解。解释型语言用解释器边解释边执行；而编译器是将整个文件翻译成目标代码后再执行。haskell语言既是编译型语言又是解释型语言。编译则采用ghc，解释则用ghci。

haskell编程语言是函数式编程语言，与以往的命令式语言不一样，非常具有代表性，让我见识到不同语言的美妙之处，开阔了我的眼界，转变了我的思维方式，为以后代码的编写奠定基础。四门语言之间的比较，让我对编程语言有了新的认识，意识到现在我所掌握的编程技术还远远不够，我需要不断学习，拓展自己的程序设计能力。同时编译器的设计会对代码的执行性能会造成影响，也为之后编译原理的学习也打下了基础，激发了我对接下来课程的好奇心和学习热情。