北京理工大學

程序语言认知实验

班　　级：07111505

姓　　名：徐宇恒

学　　号：1120151839

目录

一. 实验目的 4
二. 实验内容 4
三. 实验环境 4
3.1. 计算机硬件配置 4
3.2. 程序设计语言环境配置 5
四. 实验过程 5
4.1. 实验思路 5
4.2. Intel x86 Assembly 5
4.2.1. 数据结构 5
4.2.2. 生成用例 6
4.2.3. 矩阵乘法 6
4.3. C++ 7
4.3.1. 数据结构 7
4.3.2. 生成用例 7
4.3.3. 矩阵乘法 7
4.4. Java 7
4.4.1. 数据结构 7
4.4.2. 生成用例 8
4.4.3. 矩阵乘法 8
4.5. Haskell 8
4.5.1. 数据结构 8
4.5.2. 矩阵乘法 9
4.6. Python 9
4.6.1. 生成用例 9
4.6.2. 矩阵乘法 10
五. 语言易用性及程序规模对比 10
5.1. 代码总行数对比如图 5–1 10
5.2. 核心代码对比如图 5-2 11
5.3. 运行时间对比如图 5-3 11
5.4. 运行时间与代码行数对比如图 5-4 12

六. 心得体会 12

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

使用 MIPS/X86 汇编、C++、Java、Haskell、Python 实现矩阵乘法。并对这几种语言的编程效率、程序规模、程序运行效率进行对比对比。

三. 实验环境

3.1. 计算机硬件配置

Model	**Intel(R) Core(TM)i7-6500U**
Frequency	2.5GHz
Cores	2
L1 cache	128KB
L2 cache	512KB
L3 cache	4.0MB

3.2. 程序设计语言环境配置

语言	编译器	源
X86 汇编	Masm32	http://www.masm32.com/
C++	Visual Studio 2017	https://www.visualstudio.com/
Java	jdk1.8.0_151	http://www.oracle.com/
Haskell	Haskell 8.2.2	https://www.haskell.org/
Python	Python 3.6	https://www.python.org/

四. 实验过程

4.1. 实验思路

- 生成两个 300*300 大小的随机数组。利用三层循环对矩阵进行求解
- 核心的数学公式：arr3[i][j] +=arr1[i][k] * arr2[k][j]
- 输出运算时间

4.2. Intel x86 Assembly

4.2.1. 数据结构

```
arr1    dword   90000   dup(?)
arr2    dword   90000   dup(?)
arr3    dword   90000   dup(0)
```

4. 2. 2. 生成用例

```
;*******初始化随机数组 arr1******
    mov ecx,    num
    mov edx,    0
    mov i,      edx
l1:
    rdtsc               ;获取 CPU 当前时钟周期作为种子
    mul a               ;线性同余算法 edx=(a*eax+b)%m
    add eax,    b
    div m
    mov eax,    edx
    mov edx,    0
    div d

    mov esi,    i
    add i,      4
    mov arr1[esi],  edx
    loop l1
```

4. 2. 3. 矩阵乘法

```
;arr3[i][j] +=arr1[i][k] * arr2[k][j]
mov ebx,   t1
mov eax,    arr1[ebx]
mov ebx,    t2
mul arr2[ebx]
mov ebx,    t3
add arr3[ebx], eax
```

4.3. C++

4.3.1. 数据结构

```
//随机数组arr1, arr2, 结果数组arr3
int arr1[maxn][maxn], arr2[maxn][maxn], arr3[maxn][maxn];
```

4.3.2. 生成用例

```
//初始化随机数组
    for (int i = 0; i < maxn; i++)
    {
        for (int j = 0; j < maxn; j++)
        {
            arr1[i][j] = rand() % 101;
            arr2[i][j] = rand() % 101;
        }
    }
```

4.3.3. 矩阵乘法

```
//矩阵乘法
    for (int i = 0; i < maxn; i++)
    {
        for (int j = 0; j < maxn; j++)
        {
            for (int k = 0; k < maxn; k++)
            {
                arr3[i][j] += arr1[i][k] * arr2[k][j];
            }
        }
    }
```

4.4. Java

4.4.1. 数据结构

```
//随机数组arr1, arr2, 结果数组arr3
int arr1[maxn][maxn], arr2[maxn][maxn], arr3[maxn][maxn];
```

4.4.2. 生成用例

```
//initialize
        Random rand = new Random();
        for(int i = 0; i < maxn; i++)
        {
            for(int j = 0; j < maxn; j++)
            {
                arr1[i][j] = rand.nextInt(100);
                arr2[i][j] = rand.nextInt(100);
            }
        }
```

4.4.3. 矩阵乘法

```
//mul
        for(int i = 0; i < maxn; i++)
        {
            for(int j = 0; j < maxn; j++)
            {
                for(int k = 0; k < maxn; k++)
                {
                    arr3[i][j] += arr1[i][k] * arr2[k][j];
                }
            }
        }
```

4.5. Haskell

4.5.1. 数据结构

```
//使用Haskell 中的两级嵌套列表存储矩阵，例如：
a = [[1, 2],
     3, 4]]
```

4.5.2. 矩阵乘法

```haskell
-- 标准输入获得矩阵
data Matrix = Matrix {getMartix :: [[Int]]}
-- 矩阵数乘
smul :: Int -> Matrix -> Matrix
i `smul` Matrix xs = Matrix $ map (map (*i)) xs
-- 矩阵加法
plus ::  Matrix -> Matrix -> Matrix
Matrix xs `plus` Matrix ys = Matrix $ zipWith (zipWith (+))
xs ys
-- 矩阵减法
sub :: Matrix -> Matrix -> Matrix
Matrix xs `sub` Matrix ys = Matrix $ zipWith (zipWith (-)) xs
ys
-- 矩阵转置
transpose :: Matrix -> Matrix
transpose (Matrix ([]:_)) = Matrix []
transpose (Matrix xs) =
    Matrix $ map head xs : getMartix (transpose $ Matrix
$ map tail xs)
-- 矩阵乘法
mmul :: Matrix -> Matrix -> Matrix
Matrix [] `mmul` _ = Matrix []
Matrix (x:xs) `mmul` ys =
    Matrix $ (mul_ x $ getMartix (transpose ys)) : getMartix
(Matrix xs `mmul` ys)
mul_ :: [Int] -> [[Int]] -> [Int]
mul_ _ [] = []
mul_ x (y:ys) = sum (zipWith (*) x y) : mul_ x ys
```

4.6. Python

4.6.1. 生成用例

```python
#随机数组arr1, arr2, 结果数组arr3
arr1 = np.random.randint(0, 100, size = [maxn, maxn])
arr2 = np.random.randint(0, 100, size = [maxn, maxn])
arr3 = np.random.randint(0, 100, size = [maxn, maxn])
```

4.6.2. 矩阵乘法

```
//随机数组arr1, arr2, 结果数组arr3
for i in range(maxn):
    for j in range(maxn):
        for k in range(maxn):
            arr3[i][j] += arr1[i][k] * arr2[k][j]
```

五. 语言易用性及程序规模对比

5.1. 代码总行数对比如图 5 - 1

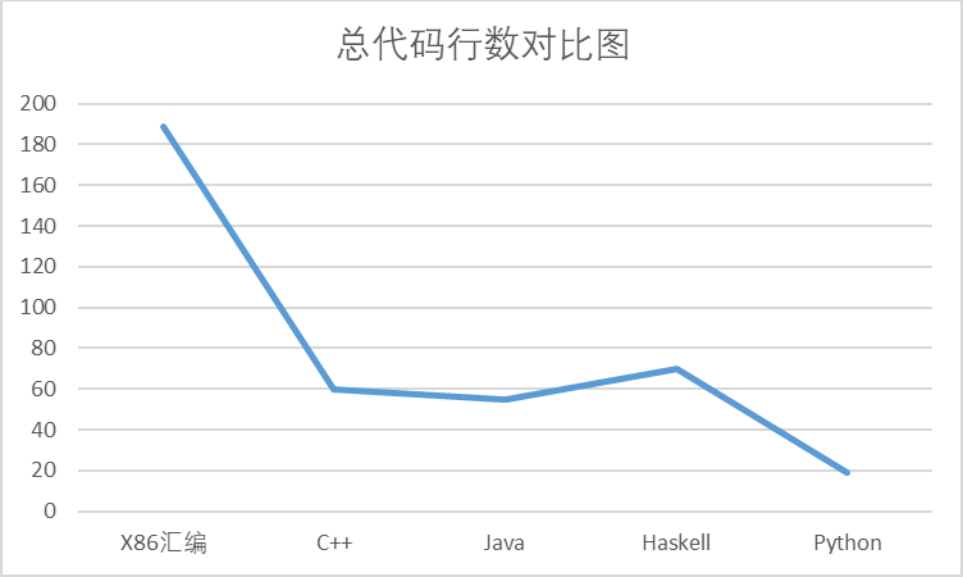

图 5 – 1

不难看出程序的代码规与语言的等级成反比，语言越高级，代码规模越小。Python 与汇编的代码相差十数倍之多。可见语言越高级，编码方式越精简，更加适合作为项目开发的工具。

5.2. 核心代码对比如图 5-2

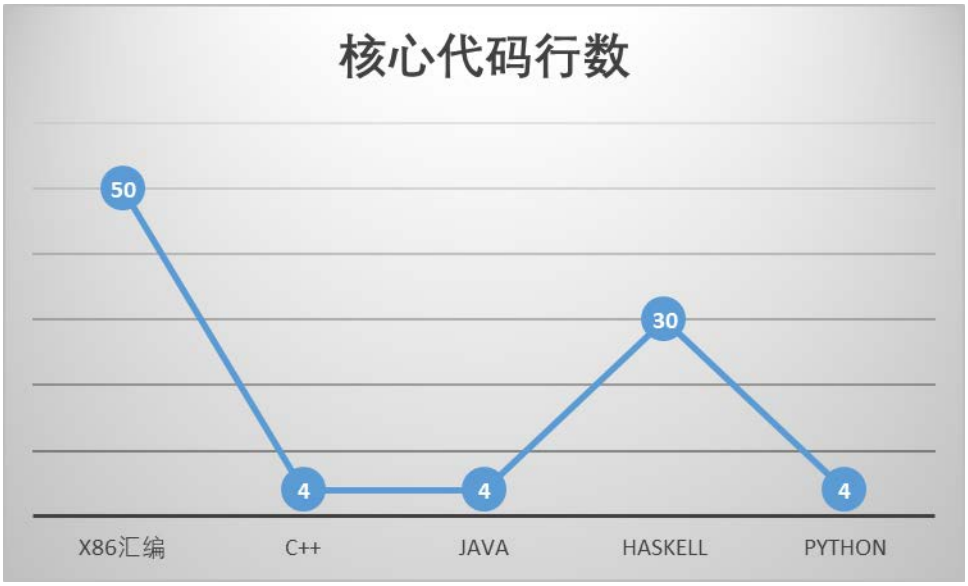

图 5－2

对于核心代码，，C++，java，python 均为三重循环结构，没有明显的差别。X86 汇编由于寄存器的使用限制导致在进行每一层循环时都要反复进行读写内存，更改寄存器内容等操作导致代码量急剧增加。而 haskell 内部并没有循环结构，只能通过递归实现循环，编码逻辑性要求最强，且实现复杂。

5.3. 运行时间对比如图 5-3

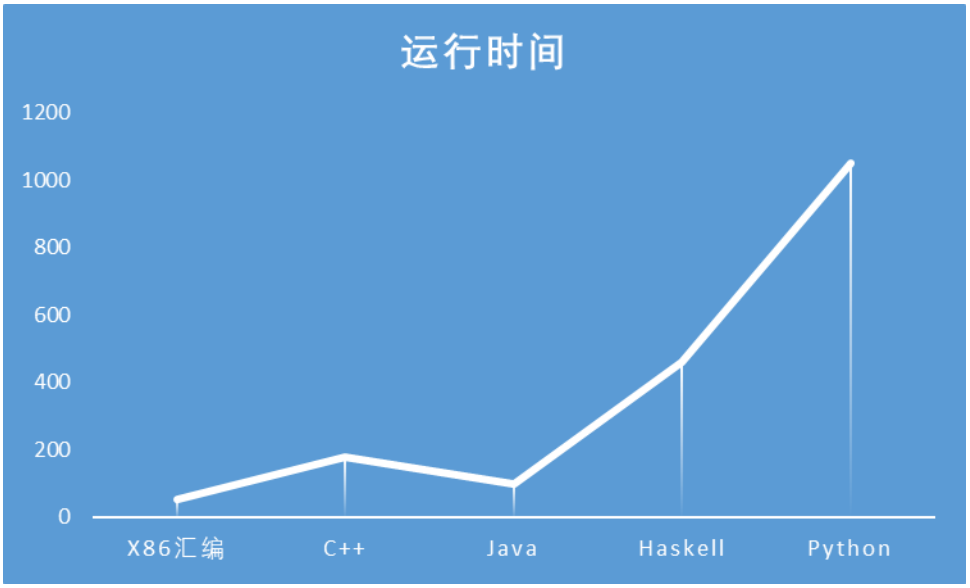

图 5－3

语言越高级，代码越精简，运行时间也越长。运行速度也相差数十倍之多。可见在速度方面，越接近底层的语言速度越快，适合用在对计算速度有极致要求的地方。

5.4. 运行时间与代码行数对比如图 5-4

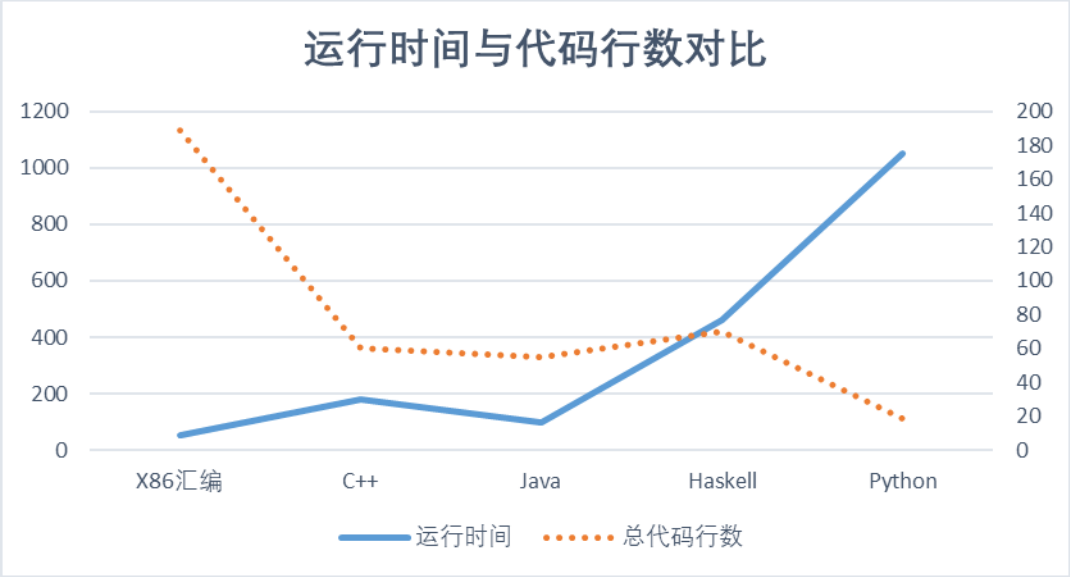

图 5－4

对比可知。低级语言和高级语言均有优势和劣势。不能武断的说高级语言就一定比低级语言好。不可否认，高级语言精简，结构清晰，对程序员更加友好，能极大的提升开发效率。但低级语言更加接近机器，对机器更加友好，运算速度更快。当然也存在像 Haskell 这样的数学编程工具语言，在日常开发中很少见到，但却是一个逻辑性极强的数学工具。

六. 心得体会

在本次实验中使用了 5 种不同的语言实现矩阵乘法运算。各具特点。

汇编是直接面向硬件的语言，所以对于机器而言最容易理解，相反的对于开发者是最不友好的。这也导致该汇编语言的优缺点十分显著。优点是与机器指令一一对应，运行速度是所有语言中最快的，一般用在某些对计算速度有极致要求的程序的核心部分，如 windows 部分内核。其运算速度较高级语言快数倍，但其编码量一般是高级语言代码量的十数倍不止，且难以阅读。

C++是在 C 语言的基础上引入了面向对象的成分。即保持了 C 语言速度快的优势同时提升了模块开发的能力，更有利于团队开发。至今仍高居编程语言使用排行版前列。

Java 是一种纯面向对象的语言。强调的是对象在类的封装以及接口方面有着严格的要求，利于工程开发。Java 是一种跨平台开发的语言。JDK 并不是将源代码直接编译为可执行文件，而是生成对应的字节码。在不同开发平台上安装对应的虚拟机即可执行。也就是所谓的一次编程可移植。java 虚拟机不仅提供跨平台的支持，而且也对代码进行了大量的优化，执行速度上也令人满意。Sum 公司也为 Java 开发了许多第三方库，有着良好的生态环境，以及海量的学习资源。

Haskell 是一种标准化的，通用的纯函数编程语言，有非限定性语义和强静态类型。作为数学工具，他的执行速度并不太高。而且 Haskell 没有循环结构，只能通过递归实现，很难上手，不适合作为开发工具使用。

Python 是当下最活跃最流行的语言。简单易懂，即使没有编程经验的人也能很快上手，降低了学习编程的门槛。而且 python 有无数的码农不停的开发优化第三方库，在机器学习，人工智能方面应用广泛。但正是由于它的简洁导致其运算速度缓慢，但是在第三方库的优化下，调用库函数仍能获得不俗的计算速度。

每种语言都有自己的优势与劣势。所以对不同的问题我们应该灵活选择合适的语言