

编译原理实验一 实验报告

一、实验目的和内容

1.1 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。

1.2 实验内容

给定一个特定的功能,分别使用 C/C++、Java、Python、Haskell 和 MIPS/X86 汇编实现该功能,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。

本次实验选取**归并排序算法(Merge Sort)**作为本次实验实现的功能。

二、实现过程和步骤

2.1 实验环境

操作系统 : Ubuntu 14.04.5 LTS 64bit

内核版本 : 4.4.0-112-generic

CPU 型号 : Intel(R) Core(TM) M-5Y10c CPU @ 0.80GHz

CPU 核数 : 4 核

CPU 主频 : 0.80Ghz

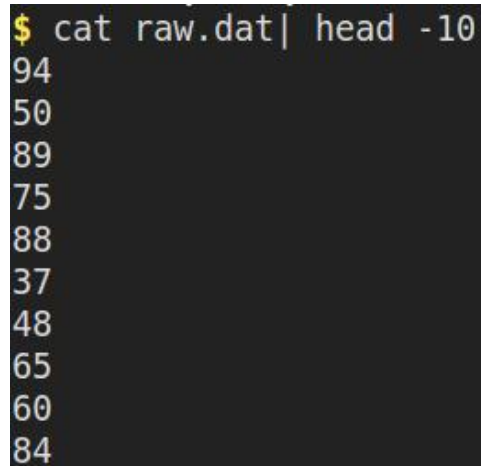
Cache 大小 : 4096KB

内存大小 : 4GB

2.2 实现过程

2.2.1 输入

输入为随机生成的 1000000 个介于 0 到 100 之间的随机整数，每个整数单独占一行，以纯文本的文件格式存储，文件名称为 raw.dat，如下图：



```
$ cat raw.dat | head -10
94
50
89
75
88
37
48
65
60
84
```

图 1 输入文件结构(节选)

在本次实验中，通过各语言对应 I/O 接口，读取数据文件。

注：raw.dat 文件应与编译后的可执行文件(C、汇编、Java、Haskell)或源代码 (Python、Haskell) 在同一目录下。

2.2.2 归并排序算法

Merge 函数：合并两组有序数组，并生成新的有序数组。

MergeSort 函数：递归的将输入的无序数组分裂成两组无序数组，并调用 Merge 函数。

具体代码实现见各语言源代码[1]。

2.2.3 输出

输出为**排序算法执行时间**。

此外，本实验中由各语言实现的程序，具有将排序完毕后的有序数据输出到文件的功能，格式同输入文件一致。为了占用额外的磁盘空间，该功能在源代码

中已经注释掉。

2.3 具体步骤

2.3.1 C 语言

使用编译器 gcc 编译源代码 mergesort.c , 生成机器可运行的可执行文件

```
gcc -g mergesort.c -o mergesort
```

注：本实验源代码中含有 makefile 文件，可使用 make 命令直接生成可执行文件

2.3.2 Java 语言

使用编译器 javac 编译源代码 mergesort.java

```
javac mergesort.java
```

使用 java 命令运行编译后生成的目标文件 mergesort.class

```
java mergesort
```

2.3.3 Python 语言

使用解释器 python 执行源代码 mergesort.py

```
python mergesort.py
```

2.3.4 Haskell 语言

Haskell 语言的执行方式可分为两种，解释执行与编译执行。

(1) 编译执行

使用编译器 ghc 编译源代码 mergesort.hs ,生成机器可运行的可执行文件

```
ghc mergesort.hs
```

(2) 解释执行

使用解释器 ghci 载入源代码 mergesort.hs

```
Prelude> :l mergesort.hs
```

在解释器中，通过调用 main 函数执行程序

```
Prelude Main> main
```

2.3.5 x86 汇编语言

通过汇编器 nasm 编译源代码 mergesort.s , 生成目标文件 mergesort.o

(32 位)

```
nasm -g -s mergesort.s -o mergesort.o -f elf32
```

通过编译器 gcc 编译目标文件 mergesort.o , 生成机器可运行的可执行文

件 (32 位)

```
gcc -g -m32 mergesort.o -o mergesort
```

注：本次实验机器支持 32 位环境。

三、运行效果截图

3.1 C 语言

```
Keep peace in mind.
22:26 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/C/bin
$ ./mergesort
cost: 0.321127s
22:26 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/C/bin
$ ./mergesort
cost: 0.319884s
22:26 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/C/bin
$ ./mergesort
cost: 0.322909s
22:26 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/C/bin
$ ./mergesort
cost: 0.320980s
22:26 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/C/bin
$ ./mergesort
cost: 0.321387s
22:26 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/C/bin
$ █
```

3.2 Java 语言

```
Keep peace in mind.
22:27 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Java/bin
$ java mergesort
cost: 0.254000s
22:27 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Java/bin
$ java mergesort
cost: 0.255000s
22:27 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Java/bin
$ java mergesort
cost: 0.257000s
22:27 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Java/bin
$ java mergesort
cost: 0.262000s
22:27 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Java/bin
$ java mergesort
cost: 0.253000s
22:27 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Java/bin
$ █
```

3.3 Python 语言

```
Keep peace in mind.
22:28 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Python/src
$ ./mergesort.py
cost: 6.76310420036s
22:28 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Python/src
$ ./mergesort.py
cost: 6.78115487099s
22:28 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Python/src
$ ./mergesort.py
cost: 6.70948505402s
22:28 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Python/src
$ ./mergesort.py
cost: 6.74358415604s
22:28 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Python/src
$ ./mergesort.py
cost: 6.83230304718s
22:29 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Python/src
$ █
```

3.4 Haskell 语言

3.4.1 编译执行结果

```

Keep peace in mind.
22:29 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Haskell/bin
$ ./mergesort
5.481827s
22:29 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Haskell/bin
$ ./mergesort
5.481787s
22:30 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Haskell/bin
$ ./mergesort
5.465715s
22:30 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Haskell/bin
$ ./mergesort
^[[A
5.492214s
22:30 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Haskell/bin
$ ./mergesort
5.472779s

```

3.4.2 解释执行结果

```

Keep peace in mind.
22:30 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/Haskell/bin
$ ghci
GHCi, version 7.6.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> :l ../../src/mergesort.hs
[1 of 1] Compiling Main                ( ../../src/mergesort.hs, interpreted )
Ok, modules loaded: Main.
*Main> main
Loading package array-0.4.0.1 ... linking ... done.
Loading package deepseq-1.3.0.1 ... linking ... done.
Loading package time-1.5.0.1 ... linking ... done.
8.283018s
*Main> main
8.331515s
*Main> main
8.161973s
*Main> main
8.251592s
*Main> main
8.175229s
*Main> 

```

3.5 x86 汇编语言


```
Keep peace in mind.
22:29 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/x86_Asm/bin
$ ./mergesort
cost: 333496ns\n%
22:29 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/x86_Asm/bin
$ ./mergesort
cost: 333217ns\n%
22:29 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/x86_Asm/bin
$ ./mergesort
cost: 334531ns\n%
22:29 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/x86_Asm/bin
$ ./mergesort
cost: 334608ns\n%
22:29 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/x86_Asm/bin
$ ./mergesort
cost: 333340ns\n%
22:29 taqini@q /home/taqini/Desktop/CompilersTheory/ep1/x86_Asm/bin
$ █
```

四、语言易用性和程序规模对比分析

表 1 各编程语言规模及易用性对比

编程语言	程序规模	语言易用性
c	适中	一般
java	适中	一般
python	较小	容易
haskell	极小	困难
asm	较大	较难

五、程序运行性能对比分析

表 2 程序性能对比

编程语言	运行时间/s	性能
c	0.32126	较好
java	0.25620	好
python	6.76592	差
haskell(解释)	8.24067	极差
haskell(编译)	5.47886	较差
asm	0.33384	较好

六、实验心得体会

Python 语言简练、易学但是效率不及 C 和 Java。

Haskell 难学、不易懂，适合专业性较高的人使用和欣赏，反正我发誓以后再也不碰这语言了。

总的来说，编译执行的语言效率比解释执行的语言高，程序规模越小的语言，性能越差。