计算机组织与体系结构实习 Lab 1

处理器性能评测实验报告

1600011398 伍诗彬

一、 我们针对系统评测的不同角度会采用不同的评测程序。在目前已有的评测程序中，为下列评测目标找到某些合适的评测程序（列出即可）。

CPU 整点性能	SPEC CPU
CPU 浮点性能	Linpack/ SPEC CPU
计算机事务处理能力	TPC-C/ SpecJAppServer2001
嵌入式系统计算能力	Dhrystone/ Livermore Loops
2D 处理能力	Cinebench
3D 处理能力	3DMark
并行计算性能	NPB/ PERFECT club
系统响应速度	SPEC CPU
编译优化能力	SPEC jvm
操作系统性能	TPC-B
多媒体处理能力	Media Bench
IO 处理能力	DiskMark
浏览器性能	Acid2
网络传输速率	Netbench
Java 运行环境性能	SPEC jvm/ Java Grande
邮件服务性能	SPEC Mail
文件服务器性能	NpBench
Web 服务器性能	SPEC Web 99/2004
服务器功耗和性能	Netbench

二、 阅读文献（Reinhold P.Weicker, An Overview of Common Benchmarks, IEEE Computer, December 1990.）并回答下面的问题

1） 简述用于性能评测的 MIPS 指标之含义，以及它是如何被计算的。

MIPS 表面含义为机器每秒能够处理以百万为单位的指令数，是一个评测机器性能好快与处理器执行速度的指标。然而由于不同机器使用的 ISA 不同，程序的同一个计算命令在不同的 ISA 需要不同数量的指令，因此 MIPS 被赋予通用性更强的含义。而各个 benchmark 也有各自的 MIPS 算法。

“VAX MIPS”是 MIPS 当前比较流行的通用定义，在这种情况下，MIPS 只是给定机器相对于 VAX11/780 机器性能的一个评测指标。如果一台机器运行某个程序或一组程序的速度是 VAX11/780 的 X 倍，它被称为 X-MIPS 机器。而对于典型的程序，VAX11/780 每秒执行 100 万条指令。

2） 使用 linux 下的剖视工具（例如 gprof）对 dhrystone 和 whetstone 进行剖视，参考论文 Table 1 形式给出数据，你的结果和该论文是否一致，为什么？

whetstone 在 gprof 剖视下的结果：

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 40.15     21.39    21.39 14000000     0.00     0.00  PA
 31.49     38.17    16.78        1    16.78    43.83  main
  7.47     42.15     3.98 899000000     0.00     0.00  P3
  6.38     45.55     3.40                             __ieee754_log_avx
  5.91     48.70     3.15                             __ieee754_exp_avx
  3.15     50.38     1.68 616000000     0.00     0.00  P0
  1.74     51.31     0.93                             __cos_avx
  0.97     51.82     0.52                             __atan_avx
  0.84     52.27     0.45                             __profile_frequency
  0.61     52.60     0.33                             __sin_avx
  0.53     52.88     0.28                             __writev_nocancel
  0.27     53.02     0.15                             log
  0.26     53.16     0.14                             exp
  0.10     53.22     0.06                             sqrt
  0.08     53.26     0.04                             __mpexp_fma4
  0.04     53.28     0.02                             sloww2
  0.01     53.28     0.01                             atanMp.constprop.0
```

Procedure	Percent
Main Program	31.49
PA	40.15
P3	7.47
P0	3.15
User Code	82.26
Trigonometric functions	3.32
Other math functions	14.42
Library functions	17.74
Total	100

dhrystone 在 gprof 剖视下的结果:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 16.27      0.78     0.78        1     0.78     3.64  main
 14.08      1.46     0.68 100000000     0.00     0.00  Proc_1
 11.26      2.00     0.54                             __writev_nocancel
 10.22      2.49     0.49 100000000     0.00     0.00  Proc_8
  9.07      2.92     0.44                             __strcmp_sse2_unaligned
  7.93      3.30     0.38 100000000     0.00     0.00  Func_2
  5.84      3.58     0.28 100000000     0.00     0.00  Proc_3
  5.63      3.85     0.27 100000000     0.00     0.00  Proc_6
  4.38      4.06     0.21 300000000     0.00     0.00  Proc_7
  3.96      4.25     0.19 100000000     0.00     0.00  Proc_2
  3.13      4.40     0.15 200000000     0.00     0.00  Func_1
  2.82      4.54     0.14                             __profile_frequency
  2.40      4.65     0.12 100000000     0.00     0.00  Proc_4
  1.98      4.75     0.10 100000000     0.00     0.00  Proc_5
  0.52      4.77     0.03                             __strcmp_ssse3
  0.52      4.80     0.03                             frame_dummy
```

Procedure	Percent
Main Program	16.27
User Procedures	59.55
User Code	75.82
Other Library functions	14.59
strcmp	9.59
Library functions	24.18
Total	100

从实验结果可以发现，whetstone 的剖视结果和论文差异较大，而 dhrystone 的剖视结果和论文相差不大。Whetstone 最大差异在于库函数运行时间占比与论文相较大幅减小。通过静态库以及 gprof，发现库函数都带有 avx 的后缀，表明 math 库函数使用了 Intel 近些年新增的 AVX 指令集来完成计算，相比于 1990 年数学函数运算效率大大提高。

在 User Code 部分比重的变化，发现 P0 从 11.6%降至 3.15%，阅读 P0 源码发现是一个轮换函数，而 CPU 的 Cache 增大将其全部缓存，因此效率提高。由于现代 CPU 的预取预测等技术的优化，也导致了 P3、PA 的占时比变化。

3） 论文中讨论了处理器之外可能对性能造成影响的因素，请分别使用两种不同的语言（例如 C 和 Java）使用同一算法实现快速排序、矩阵乘法、求 Ackermann 函数，验证文中的观点。（请保留你的程序，我们在后面可能还会用到它）

Program	Run Time(s)	
	C++ 7.3.0	Java 8
Quick Sort(1M)	0.9	1.4
Matrix Multiple(1000x1000)	3.3	4.3
Ackermann(3, 12) (recursion)	2.6	4.1

三、 性能评测

基于某个给定的计算机系统平台，使用 dhrystone、whetstone、SPEC CPU2000 开展评测、分析、研究并给出报告。

处理器性能评测报告

一、 评测目标

在同一机器上对 dhrystone, whetstone, SPEC CPU2000 int fp 三种 benchmark 进行性能评测，并对评测方案进行改进。

二、 评测环境

项目	详细指标和参数
处理器型号及相关参数（频率、架构、缓存等）	CPU 个数：8 CPU 核数：8 CPU 型号：Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHZ 架构：x86_64 缓存大小：20480kB 内核频率：2699.984
内存	总内存大小： 125.88G
外存	总容量：459G 已用：237G
操作系统及其版本	Ubuntu 16.04 64-bits
编译器版本（及编译参数）	gcc version 5.4.0
库函数及其版本	glibc version 2.23

三、 评测步骤及结果分析

1. 在 linux 下基于 dhrystone-2.1 所提供的 Makefile 编译 dhrystone
2. 分别采用 10^8、$3*10^8$、$5*10^8$、$7*10^8$、$9*10^8$ 为输入次数，运行编译生成的两个程序，记录、处理相关数据并做出解释。

输入次数	gcc_dry2	cc_dry2
10^8	12738854.0	15625000.0
$3*10^8$	12261580.0	15859030.0
$5*10^8$	12658228.0	16146395.0
$7*10^8$	12586155.0	15951386.0
$9*10^8$	12239347.0	16085791.0

观察数据我们可以发现随着输入次数线性增长，可执行文件 gcc_dry2 每秒运行的 dhrystone 数量大致稳定在 $1.25*10^7$ 左右，可执行文件 cc_dry2 每秒运行的 dhrystone 数量大致稳定在 $1.6*10^7$ 左右。横向对比我们发现，输入次数的增长对同一可执行程序的吞吐量几乎没有影响，说明输入次数的范围在操作系统能够稳定运行的范围内，因此运行时间随着输入规模线性

增长。纵向对比 gcc_dry2 和 cc_dry2，cc_dry2 的吞吐量要明显高于 gcc_dry2，因为 cc_dry2 的 OPTIMIZE 选项是 O4，而 gcc_dry2 的是 O1，因此 cc_dry2 的编译优化比 gcc_dry2 更高，因此程序运行速度更快。

3. 对 dhrystone 代码做少量修改，使其运行结果不变但“性能”提升。
4. 采用 dhrystone 进行评测有哪些可改进的地方？对其做出修改、评测和说明。

通过阅读 dhrystone 源码，我们可以发现程序本身由若干个 Proc 函数组成，在运行过程种反复调用，因此提出猜想将 Proc 函数均设为内联函数可能能提升测评程序性能。

实验结果如下：

输入次数	gcc_dry2	gcc_dry2(inline)
10^8	12738854.0	14742015.0
$3*10^8$	12261580.0	14925372.0
$5*10^8$	12658228.0	15864622.0
$7*10^8$	12586155.0	15317286.0
$9*10^8$	12239347.0	15004168.0

我们发现加上 inline 之后 gcc_dry2 的吞吐量达到 $1.5*10^7$ 左右，接近 cc_dry2 的吞吐量，有明显的“性能”提升，说明函数调用开销占用整体开销的很大部分，我们的猜想得到验证。

5. 在 linux 下使用编译器分别采用-O0、-O2、-O3 选项对 whetstone 程序进行编译并执行，记录评测结果。
6. 分别采用 10^6、10^7、10^8、10^9 为输入次数，运行编译生成的可执行程序，记录、处理相关数据并做出解释。

	-O0	-O2	-O3
10^6	Duration: 87sec	Duration: 42sec	Duration: 41sec

	1149.4MIPS	2381.0MIPS	2439.0MIPS
10^7	Duration: 850sec 1176.5MIPS	Duration: 334sec 2994.0MIPS	Duration: 329sec 3039.5MIPS
10^8	Duration: 8519sec 1173.8MIPS	Duration: 3453sec 2896.0MIPS	Duration: 3354sec 2981.5MIPS
10^9	Duration: 85102sec 1171.6MIPS	Duration: 33508sec 28506.5MIPS	Duration: 33508sec 2895.0MIPS

从结果我们可以看出，程序运行时间随着输入规模十倍增长也近似十倍增长，而在 10^7 输入规模吞吐量达到峰值。而-O0 优化和-O2、-O3 差异较大，O2、O3 差异不大。

7. 进一步改进 whetstone 程序性能（例如新的编译选项），用实验结果回答。

	-O3	-std=gnu99	-static
10^6	Duration: 41sec 2439.0MIPS	Duration: 41sec 2439.0MIPS	Duration: 28sec 3571.4MIPS
10^7	Duration: 329sec 3039.5MIPS	Duration: 336sec 2976.2MIPS	Duration: 207sec 4830.9MIPS
10^8	Duration: 3354sec 2981.5MIPS	Duration: 3375sec 2963.0MIPS	Duration: 2078sec 4812.3MIPS
10^9	Duration: 33508sec 2895.0MIPS	Duration: 33554sec 2876.5MIPS	Duration: 20652sec 4804.7MIPS

从结果我们可以看出，-std=gnu99 编译选项几乎没有优化效果，而禁止使用动态链接库的编译选项则大幅提升了性能，但是会导致编译出来的目标文件一般都很大。

8. 完成 SPEC CPU2000 的安装。
9. 修改自己的 config 文件，分别用低强度优化（例如 O2）和高强度优化（例如 O3）完成完整的 SPEC CPU2000 的评测，提交评测报告文件。

优化等级	评测结果(MIPS)	
	INT	FP
low-opt	2611	3171
high-opt	2671	3202

详细评测结果见报告文件

四、小结

在本次 lab 中，我们理解了对计算机进行性能评测的基本指标，了解目前流行的 benchmark 评测程序以及其着重测试计算机的哪些性能。在对 common benchmark 的文献进行阅读之后，我们利用 Dhrystone、Whetstone 以及 SPEC CPU2000 三个具体评测程序对我们的计算机进行了实际评测。通过实验结果以及对 Dhrystone 与 Whetstone 评测程序以及编译选项的修改也让我们理解了这计算机性能在不同编译环境下的差异，这启发了我们需要结合自己的思考对自身的应用程序进行适用于计算机的优化处理以达到提高运行效率的目的。