计算机组织与体系结构实习 **Lab 1.1**

姓名：张煌昭

学号：**1400017707**

学院：元培学院

邮箱：zhang_hz@pku.edu.cn

电话：**17888838127**

第一部分 **Benchmark**调研

针对系统评测的不同目标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合适的评测程序（列出即可）。

测评目标	测评程序	测评目标	测评程序
CPU整点性能	CINT2017	CPU浮点性能	CFP2017
计算机事务处理能力	TPC-C	嵌入式系统计算能力	CoreMark
2D处理能力	3DMark11	3D处理能力	3DMark11
并行计算性能	Linpack	系统响应速度	Loadrunner
编译优化能力	Specint2017	操作系统性能	Sysbench
多媒体处理能力	PassMark SoundCheck	I/O处理能力	Iozone
浏览器性能	wrk	网络传输速率	Netperf
Java运行环境性能	SPECjbb2005	邮件服务性能	Servermark
文件服务器性能	Servermark	Web服务器性能	Servermark

第二部分 论文阅读

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。

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？如有不同，请说明原因。

3. 论文中讨论了**CPU**之外可能对性能造成影响的几种因素。请阐述文中不同编程语言对程序性能影响的观点，并分别使用两种不同的语言（例如**C**和**Java**）使用相同算法实现快速排序、矩阵乘法、求**Ackermann**函数，验证以上观点。

1. MIPS可以指下面的任何一种MIPS指标，因而使用时需要注明使用的是哪一种MIPS。

Native MIPS指的是Millions of Instructions per Second，即每秒执行的百万指令数。由于RISC的出现，使用相同速度的RISC和CISC执行相同的程序，由于RISC的精简指令系统的设计，其Native MIPS将是CISC的数倍。因而Native MIPS这一指标对于用户来说，几乎没有意义。

Peak MIPS指的是处理器的峰值MIPS，即最高的每秒执行的百万指令数。由于对于绝大部分处理器而言，一个指令最少需要在一个时钟周期执行，因而一般情况下Peak MIPS等于处理器频率。因而这一指标，与性能几乎无关。

EDN MIPS，Dhrystone MIPS等指的是运行特定的程序而得到的Native MIPS或者VAX MIPS（请见下一段）。

VAX MIPS指的是一个同标准机器比较的分数。当规定程序，规定语言，规定编译器时，比较待测机器与VAX 11/780的运行时间的比值，由于默认VAX 11/780为1MIPS，因而可以用比值代表待测机器的MIPS。

2. 使用gprof工具，对Whetstone进行剖视，过程与结果如下。

在Makefile（请参见附件测评报告）的基础之上，加入LFLAGS = -pg -static进行编译，其中-pg注明gprof工具，-static参数使得所有动态库静态链接，从而可以使用gprof查看所有函数（包括库函数）。 make得到可执行文件后，使用./wet3 10000000命令运行-O3优化编译的Whetstone，之后再用gprof wet3 gmon.out -p命令查看各个函数运行占时。

gprof剖视结果如下表。

Procedure	Percent (%)	Paper (%)
PA	43.20	1.9
main	26.83	18.9
P3	8.78	14.4
P0	2.68	11.6
User code	81.49	46.8
__atan_avx	1.32	--
__cos_avx	1.24	--
__sin_avx	0.77	--
Trigonometric library functions	3.33	21.6
exp	0.52	--
log	0.38	--
sqrt	0.12	--
__exp1_avx	0.04	--
__ieee754_log_avx	6.51	--
__ieee754_exp_avx	6.34	--
Other math library functions	13.91	31.7
Sloww2	0.05	--
__profile_frequency	0.65	--
__writev_nocancel	0.55	--
AtanMp.constprop.0	0.01	--
Other functions	0.71	--
Total	99.44 (100)	100.1 (100)

Notice: Due to rounding, total percentage is nearly below 100%.

"Paper" means results in the paper; "--" means not-mentioned

结果与论文中结果不同。猜测认为，如此的差异是由于计算机能力造成的——论文作者所处的时代，浮点型运算能力远远不如现代计算机，因而导致当时的论文结果中库函数运算占比很高。

使用gprof工具，对Dhrystone进行剖视，过程与结果如下。

在Makefile（请参见附件测评报告）的基础之上，加入LFLAGS = -pg -static进行编译，。 make得到可执行文件后，使用./gcc_dry2 100000000命令运行Dhrystone，之后再用gprof gcc_dry2 gmon.out -p命令查看各个函数运行占时。

gprof剖视结果如下表。

Procedure	Percent (%)	Paper (%)
Proc_1	13.08	--
Proc_7	8.58	--
Proc_8	7.22	--
Proc_3	3.54	--
Proc_4	3.13	--
Proc_5	2.86	--
Proc_2	2.72	--
Proc_6	1.23	--
Func_2	5.99	--
Func_1	3.68	--
User procedures	52.03	65.7
main	25.34	18.3
User Code	77.37	84.0
__strcmp_sse2_unaligned	9.40	--
__strcmp_ssse3	0.68	--
strcmp	10.08	8.0
__writev_nocancel	8.99	--
strcpy	8.99	8.1
__profile_frequency	2.45	--
frame_dummy	1.09	--
Other functions	3.54	--
Total	99.98 (100)	100.1 (100)

Notice: Due to rounding, total percentage is nearly below 100%.

"Paper" means results in the paper; "--" means not-mentioned

结果与论文中结果基本相同。用户代码占比约80%，库函数（strcmp，strcpy）各占比约8%~10%，由于整形运算速度和库函数运行速度目前基本只与处理器频率相关，因而其占比没有很大变化。

3. CPU外对性能的影响因素还有：编程语言，编译器，动态库，高速缓存以及内存大小。

编程语言的影响：特定编程语言的某些特性会直接影响程序性能，此外，语言执行的机制也是会直接影响（比如python程序在解释器上执行理论上会比C/C++编译出的程序慢）。

编译器的影响：不同的编译选项会直接影响程序性能。比如gcc/g++编译器，优化选项-O0到-O3性能一般情况不减甚至增加；而-ffast-math和-ffloat-store等参数会针对浮点运算进行某些优化；-register编译选项会要求运算只使用寄存器...这些选项都会影响到程序性能。

动态库的影响：通过剖视可以发现，程序执行的过程中总会有一部分时间需要调用和执行库函数（例如，Whetstone和Dhrystone中都有将近20%的时间在执行库函数）。不同的库会有不同的实现方式，因而性能上会有不同的影响。

cache大小的影响：如果程序的全部text和data都可以放入cache中，那么其性能一般情况会高于没有全部放入cache的程序。同理，物理内存大小也会产生影响。

下面使用python和c两种编程语言分别实现快速排序，矩阵乘法，求Ackermann函数，共三种算法进行测试。在不同语言上实现的算法相同，测试输入也相同：

快速排序的测试输入均默认为如下序列：

$$[1,2,3,4,5,10,9,8,7,6,11,12,13,14,15,20,19,18,17,16]$$
$$\overset{sort}{\Rightarrow}[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]$$

矩阵乘法的测试输入均为如下的矩阵乘法：

$$\begin{Bmatrix}1&2&3&4&5\\6&7&8&9&10\\11&12&13&14&15\\16&17&18&19&20\\21&22&23&24&25\end{Bmatrix}\times\begin{Bmatrix}15&14&13\\12&11&10\\9&8&7\\6&5&4\\3&2&1\end{Bmatrix}\overset{matmul}{=}\begin{Bmatrix}105&90&75\\330&290&250\\444&490&425\\780&690&600\\1005&890&775\end{Bmatrix}$$

Ackermann函数的测试输入均为$m=3,\ n=3$，如下。

$$\begin{aligned}A(3,3)&=A(2,A(3,2))=A(2,A(2,A(3,1)))=A(2,A(2,A(2,A(3,0))))\\&=A(2,A(2,A(2,A(2,1))))=A(2,A(2,A(2,A(1,A(2,0)))))\\&=A(2,A(2,A(2,A(1,A(1,1)))))=A(2,A(2,A(2,A(1,A(0,A(1,0))))))\\&=A(2,A(2,A(2,A(1,A(0,A(0,A(0,1)))))))\\&=A(2,A(2,A(2,A(1,A(0,A(0,2)))))=A(2,A(2,A(2,A(1,A(0,3)))))\\&=A(2,A(2,A(2,A(1,4))))=A(2,A(2,A(2,A(0,A(1,3)))))\\&=\cdots\cdots\\&=61\end{aligned}$$

算法	编程语言		不同轮数的平均用时 / Sec				
		轮数	1000	10000	100000	1000000	10000000
快速排序	C		$7.030*10^{-7}$	$7.382*10^{-6}$	$9.744*10^{-6}$	$6.827*10^{-7}$	$5.841*10^{-7}$
	Python		$4.232*10^{-5}$	$3.430*10^{-5}$	$2.689*10^{-5}$	$2.333*10^{-5}$	$2.257*10^{-5}$
		轮数	1000	10000	100000	1000000	10000000
矩阵乘法	C		$6.700*10^{-7}$	$1.076*10^{-6}$	$8.588*10^{-7}$	$5.200*10^{-7}$	$5.210*10^{-7}$
	Python		$3.341*10^{-5}$	$2.425*10^{-5}$	$2.537*10^{-5}$	$2.326*10^{-5}$	$2.575*10^{-5}$
		轮数	10	100	1000	10000	100000
阿克曼函数	C		$9.900*10^{-6}$	$1.004*10^{-5}$	$1.626*10^{-5}$	$1.345*10^{-5}$	$8.340*10^{-6}$
	Python		$3.969*10^{-4}$	$5.419*10^{-4}$	$3.241*10^{-4}$	$3.289*10^{-4}$	$3.309*10^{-4}$

发现C语言在以上三个算法的实现，都比Python的实现性能快，用时均基本少一个数量级。

第三部分 性能评测

1. 基于你的计算机系统，使用**dhrystone**、**whetstone**开展评测、分析、研究并给出报告。报告格式见附件。

2. 基于问题一的调研结果，选择其中至少**1**项，使用相应的评测程序进行评测，并给出评测结果和简要说明。

3. (选做**)** 提供一个用于物体检测识别的**Convolutional Neural Network**（简称**CNN**）的前向计算过程。模型以一张**448×448**大小的**RGB**三通道图片作为输入，输出结果为一个**Tensor**，其中包含物体位置和类别的信息。

测评报告请见附件。