

03 | 通过你的CPU主频，我们来谈谈“性能”究竟是什么？

2019-04-29 徐文浩

深入浅出计算机组成原理

[进入课程 >](#)



讲述：徐文浩

时长 12:41 大小 11.62M



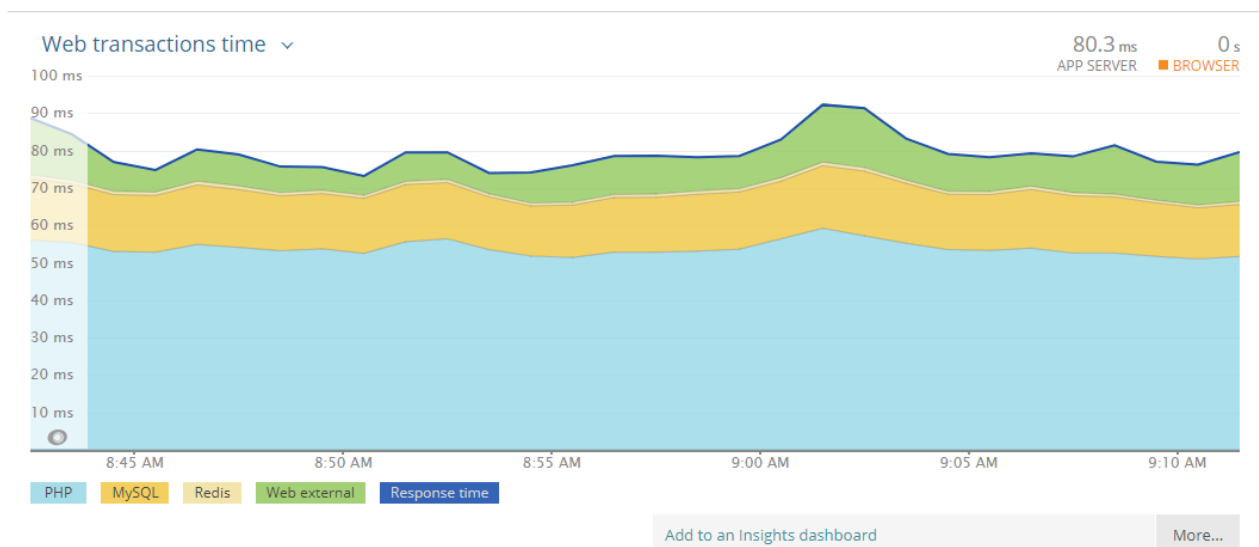
“性能”这个词，不管是在日常生活还是写程序的时候，都经常被提到。比方说，买新电脑的时候，我们会说“原来的电脑性能跟不上了”；写程序的时候，我们会说，“这个程序性能需要优化一下”。那么，你有没有想过，我们常常挂在嘴边的“性能”到底指的是什么呢？我们能不能给性能下一个明确的定义，然后来进行准确的比较呢？

在计算机组成原理乃至体系结构中，“性能”都是最重要的一个主题。我在前面说过，学习和研究计算机组成原理，就是在理解计算机是怎么运作的，以及为什么要这么运作。“为什么”所要解决的事情，很多时候就是提升“性能”。

什么是性能？时间的倒数

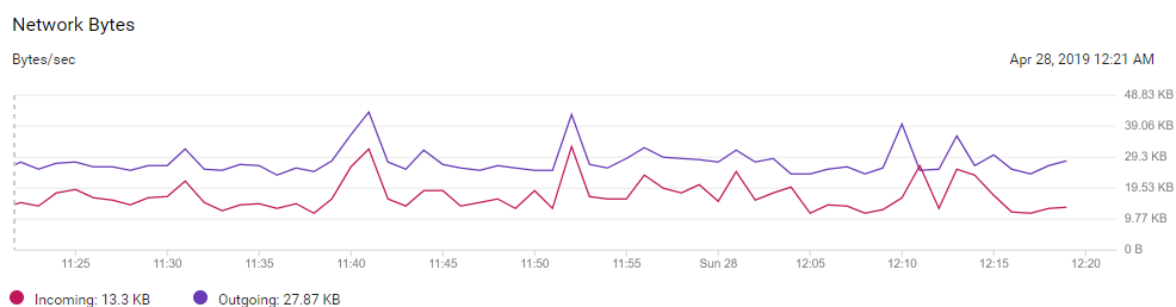
计算机的性能，其实和我们干体力劳动很像，好比是我们要搬东西。对于计算机的性能，我们需要有个标准来衡量。这个标准中主要有两个指标。

第一个是**响应时间**（Response time）或者叫执行时间（Execution time）。想要提升响应时间这个性能指标，你可以理解为让计算机“跑得更快”。



图中是我们实际系统里性能监测工具 NewRelic 中的响应时间，代表了每个外部的 Web 请求的执行时间

第二个是**吞吐量**（Throughput）或者带宽（Bandwidth），想要提升这个指标，你可以理解为让计算机“搬得更多”。



服务器使用的网络带宽，通常就是一个吞吐量性能指标

所以说，响应时间指的就是，我们执行一个程序，到底需要花多少时间。花的时间越少，自然性能就越好。

而吞吐率是指我们在一定的时间范围内，到底能处理多少事情。这里的“事情”，在计算机里就是处理的数据或者执行的程序指令。

和搬东西来做对比，如果我们的响应时间短，跑得快，我们可以来回多跑几趟多搬几趟。所以说，缩短程序的响应时间，一般来说都会提升吞吐率。

除了缩短响应时间，我们还有别的方法吗？当然有，比如说，我们还可以多找几个人一起来搬，这就类似现代的服务器都是 8 核、16 核的。人多力量大，同时处理数据，在单位时间内就可以处理更多数据，吞吐率自然也就上去了。


提升吞吐率的办法有很多。大部分时候，我们只要多加一些机器，多堆一些硬件就好了。但是响应时间的提升却没有那么容易，因为 CPU 的性能提升其实在 10 年前就处于“挤牙膏”的状态了，所以我们得慎重地来分析对待。下面我们具体来看。

我们一般把性能，定义成响应时间的倒数，也就是：

$$\text{性能} = 1 / \text{响应时间}$$

这样一来，响应时间越短，性能的数值就越大。同样一个程序，在 Intel 最新的 CPU Coffee Lake 上，只需要 30s 就能运行完成，而在 5 年前 CPU Sandy Bridge 上，需要 1min 才能完成。那么我们自然可以算出来，Coffee Lake 的性能是 1/30，Sandy Bridge 的性能是 1/60，两个的性能比为 2。于是，我们就可以说，Coffee Lake 的性能是 Sandy Bridge 的 2 倍。

过去几年流行的手机跑分软件，就是把多个预设好的程序在手机上运行，然后根据运行需要的时间，算出一个分数来给出手机的性能评估。而在业界，各大 CPU 和服务器厂商组织了一个叫作**SPEC** (Standard Performance Evaluation Corporation) 的第三方机构，专门用来指定各种“跑分”的规则。

 SPEC® CPU2017 Floating Point Rate Result <small>Copyright 2017-2018 Standard Performance Evaluation Corporation</small>	
ASUSTeK Computer Inc. ASUS RS700-E9(Z11PP-D24) Server System (2.70 GHz, Intel Xeon Gold 6150)	SPECrate2017_fp_base = 199 SPECrate2017_fp_peak = 201
CPU2017 License: 9016 Test Sponsor: ASUSTeK Computer Inc. Tested by: ASUSTeK Computer Inc.	Test Date: Dec-2017 Hardware Availability: Jul-2017 Software Availability: Sep-2017

Benchmark result graphs are available in the PDF report.

Hardware								Software							
CPU Name: Intel Xeon Gold 6150								OS: SUSE Linux Enterprise Server 12 (x86_64) SP2							
Max MHz.: 3700								Kernel 4.4.21-69-default							
Nominal: 2700								Compiler: C/C++: Version 18.0.0.128 of Intel C/C++ Compiler for Linux;							
Enabled: 36 cores, 2 chips, 2 threads/core								Fortran: Version 18.0.0.128 of Intel Fortran Compiler for Linux							
Orderable: 1, 2 chip(s)								Parallel: No							
Cache L1: 32 KB I + 32 KB D on chip per core								Firmware: Version 0601 released Oct-2017							
L2: 1 MB I+D on chip per core								File System: btrfs							
L3: 24.75 MB I+D on chip per chip								System State: Run level 3 (multi-user)							
Other: None								Base Pointers: 64-bit							
Memory: 768 GB (24 x 32 GB 2Rx4 PC4-2666V-R)								Peak Pointers: 64-bit							
Storage: 1 x 960 GB SATA SSD								Other: None							
Other: None															
Results Table															
Benchmark	Base							Peak							
	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	
503.bwaves_r	72	1529	472	1529	472	1530	472	72	1526	473	1526	473	1526	473	
507.cactuBSSN_r	72	504	181	505	181	505	180	72	505	180	505	180	505	180	
508.namd_r	72	383	179	381	180	383	179	72	382	179	382	179	383	179	
510.parest_r	72	1669	113	1684	112	1700	111	72	1687	112	1699	111	1695	111	
511.povray_r	72	603	279	618	272	596	282	72	521	323	524	321	525	320	
519.1bm_r	72	678	112	677	112	677	112	72	677	112	678	112	677	112	
521.wrf_r	72	772	209	784	206	787	205	72	788	205	785	205	785	205	
526.blender_r	72	495	222	495	222	495	222	72	487	225	487	225	488	225	
527.cam4_r	72	547	230	550	229	550	229	72	543	232	542	232	542	232	
538.imagick_r	72	509	352	509	352	510	351	72	511	351	511	351	511	351	
544.nab_r	72	399	303	398	304	400	303	72	396	306	396	306	395	307	
549.fotonik3d_r	72	1985	141	1986	141	1986	141	72	1985	141	1981	142	1985	141	
554.roms_r	72	1263	90.6	1268	90.2	1269	90.2	72	1271	90.0	1270	90.1	1271	90.0	

一份 SPEC 报告通常包含了大量不同测试的评分

SPEC 提供的 CPU 基准测试程序，就好像 CPU 届的“高考”，通过数十个不同的计算程序，对于 CPU 的性能给出一个最终评分。这些程序丰富多彩，有编译器、解释器、视频压缩、人工智能国际象棋等等，涵盖了方方面面的应用场景。感兴趣的话，你可以点击[这个链接](#)看看。

计算机的计时单位：CPU 时钟

虽然时间是一个很自然的用来衡量性能的指标，但是用时间来衡量时，有两个问题。

第一个就是时间不“准”。如果用你自己随便写的一个程序，来统计程序运行的时间，每一次统计结果不会完全一样。有可能这一次花了 45ms，下一次变成了 53ms。


为什么会不准呢？这里面有好几个原因。首先，我们统计时间是用类似于“掐秒表”一样，记录程序运行结束的时间减去程序开始运行的时间。这个时间也叫 Wall Clock Time 或者

Elapsed Time，就是在运行程序期间，挂在墙上的钟走掉的时间。

但是，计算机可能同时运行着好多个程序，CPU 实际上不停地在各个程序之间进行切换。在这些走掉的时间里面，很可能 CPU 切换去运行别的程序了。而且，有些程序在运行的时候，可能要从网络、硬盘去读取数据，要等网络和硬盘把数据读出来，给到内存和 CPU。所以说，**要想准确统计某个程序运行时间，进而去比较两个程序的实际性能，我们得把这些时间给刨除掉。**

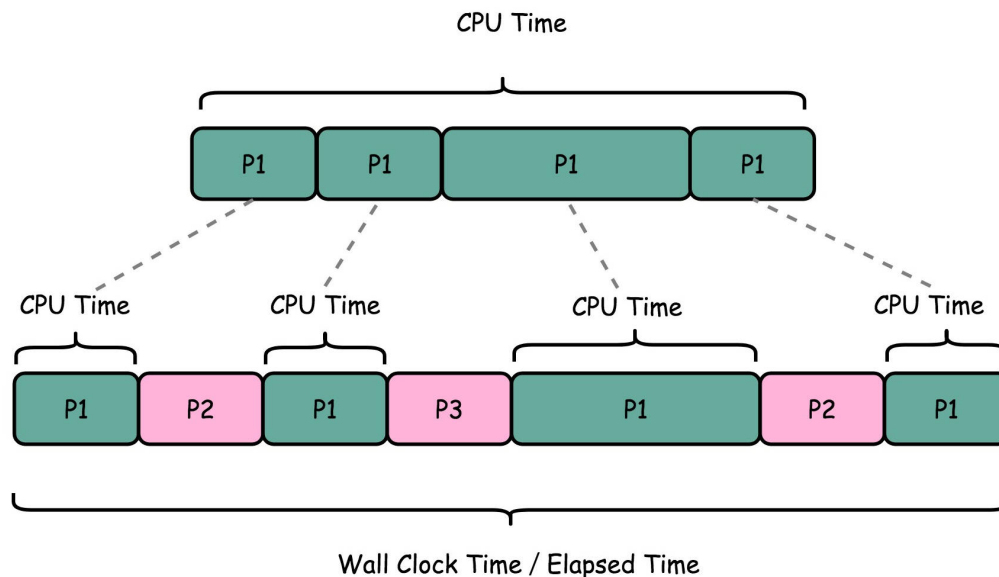
那这件事怎么实现呢？Linux 下有一个叫 time 的命令，可以帮我们统计出来，同样的 Wall Clock Time 下，程序实际在 CPU 上到底花了多少时间。

我们简单运行一下 time 命令。它会返回三个值，第一个是**real time**，也就是我们说的 Wall Clock Time，也就是运行程序整个过程中流逝掉的时间；第二个是**user time**，也就是 CPU 在运行你的程序，在用户态运行指令的时间；第三个是**sys time**，是 CPU 在运行你的程序，在操作系统内核里运行指令的时间。而**程序实际花费的 CPU 执行时间（CPU Time）**，就是 **user time 加上 sys time**。

 复制代码

```
1 $ time seq 1000000 | wc -l
2 1000000
3
4
5 real    0m0.101s
6 user    0m0.031s
7 sys     0m0.016s
```

在我给的这个例子里，你可以看到，实际上程序用了 0.101s，但是 CPU time 只有 $0.031 + 0.016 = 0.047s$ 。运行程序的时间里，只有不到一半是实际花在这个程序上的。



程序实际占用的 CPU 时间一般比 Elapsed Time 要少不少

其次，即使我们已经拿到了 CPU 时间，我们也不一定可以直接“比较”出两个程序的性能差异。即使在同一台计算机上，CPU 可能满载运行也可能降频运行，降频运行的时候自然花的时间会多一些。

除了 CPU 之外，时间这个性能指标还会受到主板、内存这些其他相关硬件的影响。所以，我们需要对“时间”这个我们可以感知的指标进行拆解，把程序的 CPU 执行时间变成 CPU 时钟周期数（CPU Cycles）和时钟周期时间（Clock Cycle）的乘积。

程序的 CPU 执行时间 = CPU 时钟周期数 × 时钟周期时间

我们先来理解一下什么是时钟周期时间。你在买电脑的时候，一定关注过 CPU 的主频。比如我手头的这台电脑就是 Intel Core-i7-7700HQ 2.8GHz，这里的 2.8GHz 就是电脑的主频（Frequency/Clock Rate）。这个 2.8GHz，我们可以先粗浅地认为，CPU 在 1 秒时间内，可以执行的简单指令的数量是 2.8G 条。

如果想要更准确一点描述，这个 2.8GHz 就代表，我们 CPU 的一个“钟表”能够识别出来的最小的时间间隔。就像我们挂在墙上的挂钟，都是“滴答滴答”一秒一秒地走，所以通过墙上的挂钟能够识别出来的最小时间单位就是秒。

而在 CPU 内部，和我们平时戴的电子石英表类似，有一个叫晶体振荡器（Oscillator Crystal）的东西，简称为晶振。我们把晶振当成 CPU 内部的电子表来使用。晶振带来的每

一次“滴答”，就是时钟周期时间。

在我这个 2.8GHz 的 CPU 上，这个时钟周期时间，就是 $1/2.8G$ 。我们的 CPU，是按照这个“时钟”提示的时间来进行自己的操作。主频越高，意味着这个表走得越快，我们的 CPU 也就“被逼”着走得越快。

如果你自己组装过台式机的话，可能听说过“超频”这个概念，这说的其实就相当于把买回来的 CPU 内部的钟给调快了，于是 CPU 的计算跟着这个时钟的节奏，也就自然变快了。当然这个快不是没有代价的，CPU 跑得越快，散热的压力也就越大。就和人一样，超过生理极限，CPU 就会崩溃了。

我们现在回到上面程序 CPU 执行时间的公式。

程序的 CPU 执行时间 = CPU 时钟周期数 × 时钟周期时间

最简单的提升性能方案，自然缩短时钟周期时间，也就是提升主频。换句话说，就是换一块好一点的 CPU。不过，这个是我们这些软件工程师控制不了的事情，所以我们就把目光挪到了乘法的另一个因子——CPU 时钟周期数上。如果能够减少程序需要的 CPU 时钟周期数量，一样能够提升程序性能。

对于 CPU 时钟周期数，我们可以再做一个分解，把它变成“指令数 × **每条指令的平均时钟周期数** (Cycles Per Instruction, 简称 CPI) ”。不同的指令需要的 Cycles 是不同的，加法和乘法都对应着一条 CPU 指令，但是乘法需要的 Cycles 就比加法要多，自然也就慢。在这样拆分了之后，我们的程序的 CPU 执行时间就可以变成这样三个部分的乘积。

程序的 CPU 执行时间 = 指令数 × CPI × Clock Cycle Time

因此，如果我们想要解决性能问题，其实就是要优化这三者。

1. 时钟周期时间，就是计算机主频，这个取决于计算机硬件。我们所熟知的[摩尔定律](#)就一直在不停地提高我们计算机的主频。比如说，我最早使用的 80386 主频只有 33MHz，现在手头的笔记本电脑就有 2.8GHz，在主频层面，就提升了将近 100 倍。
2. 每条指令的平均时钟周期数 CPI，就是一条指令到底需要多少 CPU Cycle。在后面讲解 CPU 结构的时候，我们会看到，现代的 CPU 通过流水线技术 (Pipeline)，让一条指令

需要的 CPU Cycle 尽可能地少。因此，对于 CPI 的优化，也是计算机组成和体系结构中的重要一环。

3. 指令数，代表执行我们的程序到底需要多少条指令、用哪些指令。这个很多时候就把挑战交给了编译器。同样的代码，编译成计算机指令时候，就有各种不同的表示方式。

我们可以把自己想象成一个 CPU，坐在那里写程序。计算机主频就好像是你的打字速度，打字越快，你自然可以多写一点程序。CPI 相当于你在写程序的时候，熟悉各种快捷键，越是打同样的内容，需要敲击键盘的次数就越少。指令数相当于你的程序设计得够合理，同样的程序要写的代码行数就少。如果三者皆能实现，你自然可以很快地写出一个优秀的程序，你的“性能”从外面来看就是好的。

总结延伸

好了，学完这一讲，对“性能”这个名词，你应该有了更清晰的认识。我主要对于“响应时间”这个性能指标进行抽丝剥茧，拆解成了计算机时钟周期、CPI 以及指令数这三个独立的指标的乘积，并且为你指明了优化计算机性能三条康庄大道。也就是，提升计算机主频，优化 CPU 设计使得在单个时钟周期内能够执行更多指令，以及通过编译器来减少需要的指令数。

在后面的几讲里面，我会为你讲解，具体怎么在电路硬件、CPU 设计，乃至指令设计层面，提升计算机的性能。

课后思考

每次有新手机发布的时候，总会有一些对于手机的跑分结果的议论。乃至有“作弊”跑分或者“针对跑分优化”的说法。我们能针对“跑分”作弊么？怎么做到呢？“作弊”出来的分数对于手机性能还有参考意义么？

欢迎留言和我分享你的思考和疑惑，你也可以把今天的内容分享给你的朋友，和他一起学习和进步。

深入浅出计算机组成原理

带你掌握计算机体系全貌

徐文浩 bothub 创始人



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | 给你一张知识地图，计算机组成原理应该这么学

下一篇 04 | 穿越功耗墙，我们该从哪些方面提升“性能”？

精选留言 (65)

写留言



活的潇洒 置顶

2019-04-30

👍 26

运行的代码是：

```
[root@nfs ~]# time seq 1000000 | wc -l  
1000000
```

```
real 0m0.058s...
```

展开 ▾

作者回复: 因为你在一台多核或者多cpu的机器上运行，seq和wc命令会分配到两个cpu上，user和sys是两个cpu时间相加的，而real只是现实时钟里走过的时间，极端情况下user+sys可以到达real的两倍



趁早 置顶

2019-04-29

👍 3

```
time seq 100000 | wc -l  
100000
```

```
real 0m0.006s  
user 0m0.003s...
```

展开 ▾

作者回复: 我知道原因了, 这个的确是因为“并行原因”的运行的。虽然seq和wc这两个命令都是单线程运行的, 但是这两个命令在多核cpu运行的情况下, 会分别分配到两个不同的cpu, 于是user和sys的时间都是两个cpu上运行的时间之和, 就可能超过real的时间。你可以这样来快速验证

运行

```
time seq 100000000 | wc -l &
```

让这个命令多跑一会儿, 并且在后台运行。

然后利用 top 命令看不同进程的cpu占用情况, 你会在top的前几行里看到seq和wc的cpu占用都接近100, 实际是各被分配到了一个不同的cpu执行。

我写文稿测试的时候开了一个1u的最小的虚拟机, 只有一个cpu所以不会遇到这个问题。



changing

2019-04-29

👍 17

运行的代码是 time seq 100000 | wc -l

```
real 0m0.033s
```

```
user 0m0.030s
```

```
sys 0m0.005s
```

为什么user + sys 运行出来会比real time 多呢

展开 ▾

作者回复: changing同学你好, 一般情况下, 如果user+sys比real大, 甚至光光user比real大的情况出现, 都是因为对应的程序被多个进程或者多个线程并行执行了, 也很常见。不过你遇到的这个问题的确有些奇怪, 我要研究一下, 因为linux下的seq和wc这些命令按照我的理解都是单线程运行的。

能告诉我你使用的硬件和操作系统么?





KR®

2019-04-30

👍 10

又重刷了一遍前四讲, 徐老师讲得又清惜又易懂, 老师备课花了不少心血吧...
现在等待更新的心情就像追了一部超高分剧等更一样!!辛苦徐老师备课喇^^

作者回复: 谢谢支持😊



易儿易

2019-05-01

👍 8

老师, 针对“主频越高, 意味着这个表走得越快, 我们的 CPU 也就“被逼”着走得越快。”这句话我有一点儿疑惑:
时钟周期时间为 $1/2.8G$ 秒, 代表CPU最细粒度时间, 即一次晶振的时间
这个周期时间和指令执行的耗时有直接关系吗? 我说的直接关系指的是比如“一次晶振时间可以固定完成n个CPU (最简单的) 指令”这种, 如果有关系的话, 那可以很明确的得...
展开▼

作者回复: 易儿易同学你好, 这个问题提的得非常好, 你学得和思考得都很仔细深入。1的理解更准确一点, 我们为了理解简单可以暂且认为就是晶振在触发一条一条电路变化指令, 就好像你拨算牌盘的节奏一样。算盘拨得快, 珠算就算得快。结果就是一条简单的指令需要的事件就和一个时钟周期一样, 实际这个问题要比这样一句话复杂很多。一方面, 其实时钟周期应该是放下最复杂的一条指令的时间长度。我们是通过流水线来提升cpi的。我会在讲解cpu的部分更深入讲解始终信号和计数器, 让大家能够理解cpu到底是怎么回事儿。



Only now

2019-04-30

👍 6

猜测, 跑分程序载入后, 停止操作系统的线程调度或者给最高优先级和响应中断, 全力跑跑分。暂时提高时钟频率, 停止温度检测和低级中断, 这样CPU就全力在跑测试程序了吧。

没做过弊, 猜测

展开▼

作者回复: 🐞监测到跑分程序在运行进行超频或者过热也不降频是一种常见的作弊手段



imicode

2019-05-14

👍 4

1. 打卡总结:

性能的CPU有两个重要的指标，响应时间和吞吐率。在这两个重要指标下，要提升性能，核心是优化CPU的执行时间，而CPU执行时间公式如下：

程序的 CPU 执行时间 = 指令数×CPI×Clock Cycle Time...

展开 ▾

作者回复: 📬



大雄逸豪

2019-05-13

👍 3

搞明白这个事实就好了，一个程序对应多条语句，一条编程语句可能对应多条指令，一条CPU指令可能需要多个CPU周期才能完成。

作者回复: 📬



humor

2019-04-30

👍 3

对于文中的CPU钟表时间间隔和时钟周期还是没有理解很清楚，时间间隔和时钟周期是互为倒数的关系吗？就是CPU主频是一个单位时间，而时钟周期就是这个单位时间被分成主频(2.8G)等份的一份吗？

展开 ▾

作者回复: humor同学你好，如果我没理解错你的意思的话，你的理解是对的。CPU主频是一个频率（frequency），频率的单位叫做赫兹（Hz）。意思是一秒内这个事情可以发生多少次。主频2.8GHz就代表一秒内晶振振动了2.8G次，这里的G其实就是10亿次，也就是28亿次。那么我们的时钟周期时间就是1/28亿秒。



sunbiaozj

2019-04-29

👍 3

文章思路非常清晰，很棒

展开 ▾



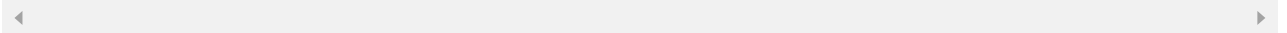
Guarantee

2019-05-17

👍 2

老师，单个CPU的主频是有上限的，所以出现了多核CPU进行计算，为了提高更多的计算，是不是就要运用分布式计算这个技术。

作者回复: 是的



霹雳

2019-05-02

👍 2

用户态运行和系统内核运行这两个什么区别呢

展开 ▾

作者回复: 霹雳同学你好，

关于用户态运行和系统内核运行，如果要深入了解的话，可以去看看刘超老师的《趣谈Linux操作系统》。

如果简单讲一下的话，就是我们的程序实际在操作系统里面是运行在“保护模式”下的，很多指令我们的应用程序并没有权限去操作执行，需要切换到内核态，由操作系统去执行，比如说操作硬件的时候。



爷爷刘大

2019-04-30

👍 2

changing 同学的问题在stackoverflow 上的解释是这样的：

The rule of thumb is:

real < user: The process is CPU bound and takes advantage of parallel execution on multiple cores/CPU's....

展开 ▾

作者回复: 嗯, 但是核心问题是按理一般seq和wc都不是并行运行的, 我需要看一下是为什么



潜默闻雨

2019-04-29

👍 2

徐老师, 程序的cpu执行时间是不是由很多cpu时间片组成, 而cpu并不知道自己在执行哪个程序的指令, 只是按时间片去按顺序执行指令, 不知道这样理解对不对? 非科班的转行人士, 正在努力补基础😁。。。。

展开 ∨

作者回复: 潜默闻雨同学你好, 这个理解没错。到了cpu层面只有一条条机器码的指令, 它并不关心这个指令具体是从哪个程序里来的。



Geek_63ad8...

2019-05-15

👍 1

老师我尝试了用自己的话理解一下您讲授的内容:

度量一个程序运行的时间 T 需要知道该程序有几条指令(n), 每一个指令平均需要几个基本操作才能执行完毕(k), cpu执行一个基本操作的耗时(t), 从而 $T = n * k * t$, t 作为SE一般是无法提升的, 除非改进硬件, 所以缩短运行时间可能主要还是从 n 、 k 入手。不知道这样理解是否正确? ...

展开 ∨

作者回复: 👍 你的理解是正确的



秦晋

2019-05-12

👍 1

看到cpu指令这一块, 我想起了精简指令和复杂指令, 执行同样的任务, 精简指令需要的条数少, 复杂指令需要的多, 是不是说同样的任务, 放在同样频率的精简指令cpu和复杂指令cpu上执行, 精简指令cpu的执行效率高?

展开 ∨

作者回复: 秦晋同学, 某种程度上来说, 你理解反了。精简指令集意味着cpu从硬件或者电路层面支持的指令数比较少。这个意味着很多复杂的操作需要执行更多的指令而不是更少的。

执行效率这个问题更复杂一些，精简指令也许更容易提高频率或者利用流水线等等，只能说具体问题具体分析。现在更多地是出于一个“混合”的状态

Ant

2019-05-06

👍 1

时钟周期是啥意思

展开 ▾

作者回复: Ant同学你好，时钟周期，是CPU内部通过一个反馈电路形成的一个“晶振”的产生反复的0/1电路信号的一个频率，在17讲讲解CPU的时候，我们还会深入讲解一下时钟周期是怎么回事儿。



长满鱼的树

2019-05-03

👍 1

思路很清晰，谢谢老师

展开 ▾



txhh

2019-04-30

👍 1

```
time seq 100000 | wc -l
100000
```

```
real 0m0.003s
user 0m0.003s...
```

展开 ▾

作者回复: 我知道原因了，这个的确是因为“并行原因”的运行的。虽然seq和wc这两个命令都是单线程运行的，但是这两个命令在多核cpu运行的情况下，会分别分配到两个不同的cpu，于是user和sys的时间都是两个cpu上运行的时间之和，就可能超过real的时间。你可以这样来快速验证

运行

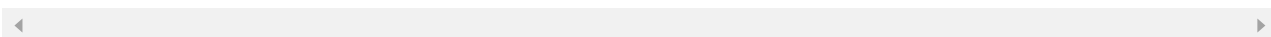
```
time seq 100000000 | wc -l &
```

让这个命令多跑一会儿，并且在后台运行。

然后利用 top 命令看不同进程的cpu占用情况，你会在top的前几行里看到seq和wc的cpu占用都

接近100，实际是各被分配到了一个不同的cpu执行。

我写文稿测试的时候开了一个1u的最小的虚拟机，只有一个cpu所以不会遇到这个问题。



脊椎疼

2019-04-30

👍 1

同学们问题好多啊，老师压力山大啊！

展开 ▾

作者回复: 欢迎大家踊跃提问参与讨论！

