04-穿越功耗墙，我们该从哪些方面提升“性能”？

上一讲，在讲CPU的性能时，我们提到了这样一个公式：

程序的CPU执行时间 = 指令数×CPI×Clock Cycle Time

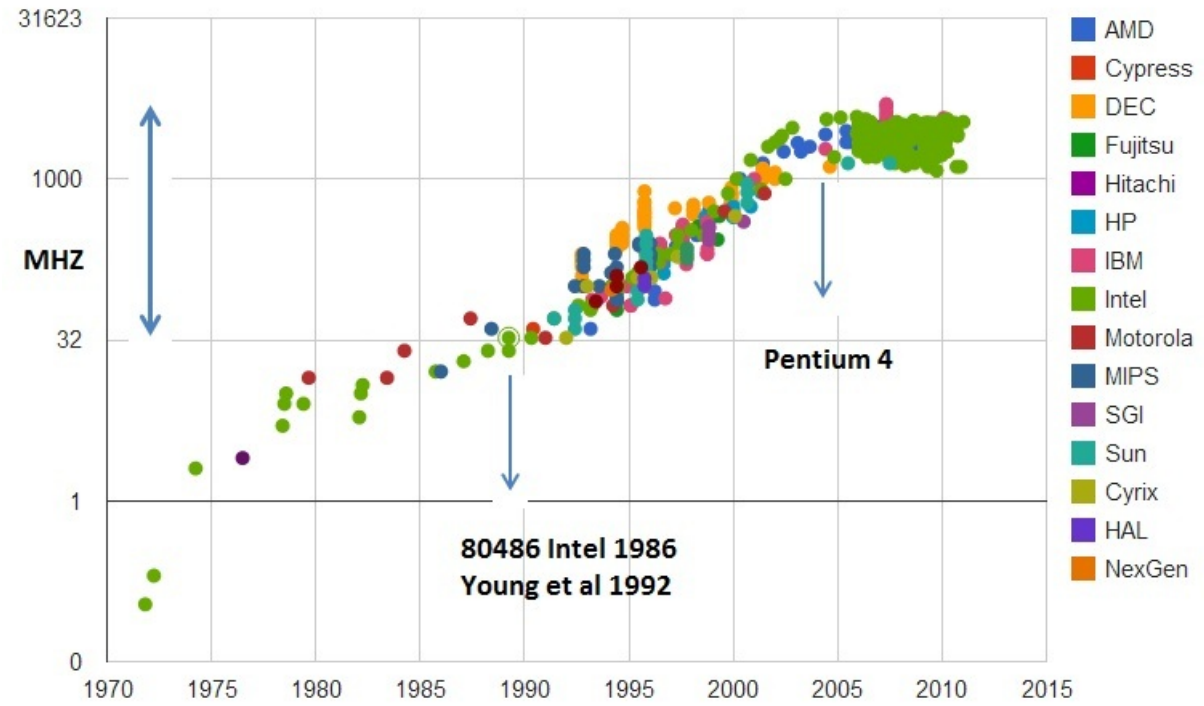
这么来看，如果要提升计算机的性能，我们可以从指令数、CPI以及CPU主频这三个地方入手。要搞定指令数或者CPI，乍一看都不太容易。于是，研发CPU的硬件工程师们，从80年代开始，就挑上了CPU这个“软柿子”。在CPU上多放一点晶体管，不断提升CPU的时钟频率，这样就能让CPU变得更快，程序的执行时间就会缩短。

于是，从1978年Intel发布的8086 CPU开始，计算机的主频从5MHz开始，不断提升。1980年代中期的80386能够跑到40MHz，1989年的486能够跑到100MHz，直到2000年的奔腾4处理器，主频已经到达了1.4GHz。而消费者也在这20年里养成了“看主频”买电脑的习惯。当时已经基本垄断了桌面CPU市场的Intel更是夸下了海口，表示奔腾4所使用的CPU结构可以做到10GHz，颇有一点“大力出奇迹”的意思。

功耗：CPU的“人体极限”

然而，计算机科学界从来不相信“大力出奇迹”。奔腾4的CPU主频从来没有达到过10GHz，最终它的主频上限定格在3.8GHz。这还不是最糟的，更糟糕的事情是，大家发现，奔腾4的主频虽然高，但是它的实际性能却配不上同样的主频。想要用在笔记本上的奔腾4 2.4GHz处理器，其性能只和基于奔腾3架构的奔腾M 1.6GHz处理器差不多。

于是，这一次的“大力出悲剧”，不仅让Intel的对手AMD获得了喘息之机，更是代表着“主频时代”的终结。后面几代Intel CPU主频不但没有上升，反而下降了。到如今，2019年的最高配置Intel i9 CPU，主频也只不过是5GHz而已。相较于1978年到2000年，这20年里300倍的主频提升，从2000年到现在的这19年，CPU的主频大概提高了3倍。



CPU的主频变化，在奔腾4时代进入了瓶颈期，图片来源

奔腾4的主频为什么没能超过3.8GHz的障碍呢？答案就是功耗问题。什么是功耗问题呢？我们先看一个直观的例子。

一个3.8GHz的奔腾4处理器，满载功率是130瓦。这个130瓦是什么概念呢？机场允许带上飞机的充电宝的容量上限是100瓦时。如果我们把这个CPU安在手机里面，不考虑屏幕内存之类的耗电，这个CPU满载运行45分钟，充电宝里面就没电了。而iPhone X使用ARM架构的CPU，功率则只有4.5瓦左右。

我们的CPU，一般都被叫作**超大规模集成电路**（Very-Large-Scale Integration，VLSI）。这些电路，实际上都是一个个晶体管组合而成的。CPU在计算，其实就是让晶体管里面的“开关”不断地去“打开”和“关闭”，来组合完成各种运算和功能。

想要计算得快，一方面，我们要在CPU里，同样的面积里面，多放一些晶体管，也就是**增加密度**；另一方面，我们要让晶体管“打开”和“关闭”得更快一点，也就是**提升主频**。而这两者，都会增加功耗，带来耗电和散热的问题。

这么说可能还是有点抽象，我还是给你举一个例子。你可以把一个计算机CPU想象成一个巨大的工厂，里面有很多工人，相当于CPU上面的晶体管，互相之间协同工作。

为了工作得快一点，我们要在工厂里多塞一点人。你可能会问，为什么不把工厂造得大一点呢？这是因为，人和人之间如果离得远了，互相之间走过去需要花的时间就会变长，这也会导致性下降。这就好像如果CPU的面积大，晶体管之间的距离变大，电信号传输的时间就会变长，运算速度自然就慢了。

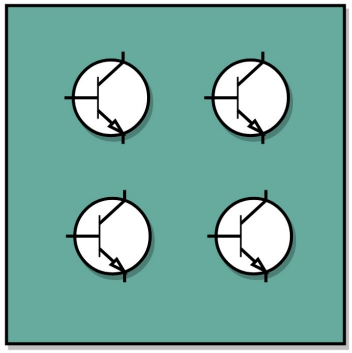
除了多塞一点人，我们还希望每个人的动作都快一点，这样同样的时间里就可以多干一点活儿了。这就相当于提升CPU主频，但是动作快，每个人就要出汗散热。要是太热了，对工厂里面的人来说会中暑生病，对CPU来说就会崩溃出错。

我们会在CPU上面抹硅脂、装风扇，乃至用上水冷或者其他更好的散热设备，就好像在工厂里面装风扇、空调，发冷饮一样。但是同样的空间下，装上风扇空调能够带来的散热效果也是有极限的。

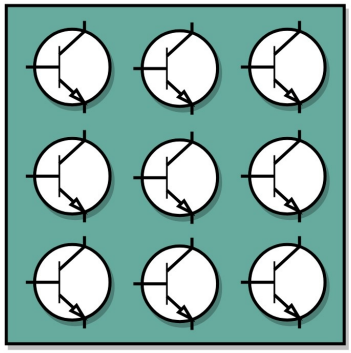
因此，在CPU里面，能够放下的晶体管数量和晶体管的“开关”频率也都是有限的。一个CPU的功率，可以用这样一个公式来表示：

$$\text{功耗} \sim= 1/2 \times \text{负载电容} \times \text{电压的平方} \times \text{开关频率} \times \text{晶体管数量}$$

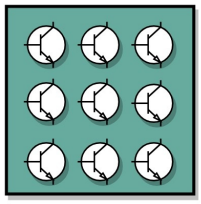
那么，为了要提升性能，我们需要不断地增加晶体管数量。同样的面积下，我们想要多放一点晶体管，就要把晶体管造得小一点。这个就是平时我们所说的提升“制程”。从28nm到7nm，相当于晶体管本身变成了原来的1/4大小。这个就相当于我们在工厂里，同样的活儿，我们要找瘦小一点的工人，这样一个工厂里面就可以多一些人。我们还要提升主频，让开关的频率变快，也就是要找手脚更快的工人。



芯片内部的晶体管



更多数量的晶体管会带来能耗和散热的挑战



制程的提升解决了能耗和散热的问题，还带来了更小的芯片

但是，功耗增加太多，就会导致CPU散热跟不上，这时，我们就需要降低电压。这里有一点非常关键，在整个功耗的公式里面，功耗和电压的平方是成正比的。这意味着电压下降到原来的1/5，整个的功耗会变成原来的1/25。

事实上，从5MHz主频的8086到5GHz主频的Intel i9，CPU的电压已经从5V左右下降到了1V左右。这也是为什么我们CPU的主频提升了1000倍，但是功耗只增长了40倍。比如说，我写这篇文章用的是Surface Go，在这样的轻薄笔记本上，微软就是选择了把电压下降到0.25V的低电压CPU，使得笔记本能有更长的续航时间。

并行优化，理解阿姆达尔定律

虽然制程的优化和电压的下降，在过去的20年里，让我们的CPU性能有所提升。但是从上世纪九十年代到本世纪初，软件工程师们所用的“面向摩尔定律编程”的套路越来越用不下去了。“写程序不考虑性能，等明年CPU性能提升一倍，到时候性能自然就不成问题了”，这种想法已经不可行了。

于是，从奔腾4开始，Intel意识到通过提升主频比较“难”去实现性能提升，边开始推出Core Duo这样的多核CPU，通过提升“吞吐率”而不是“响应时间”，来达到目的。

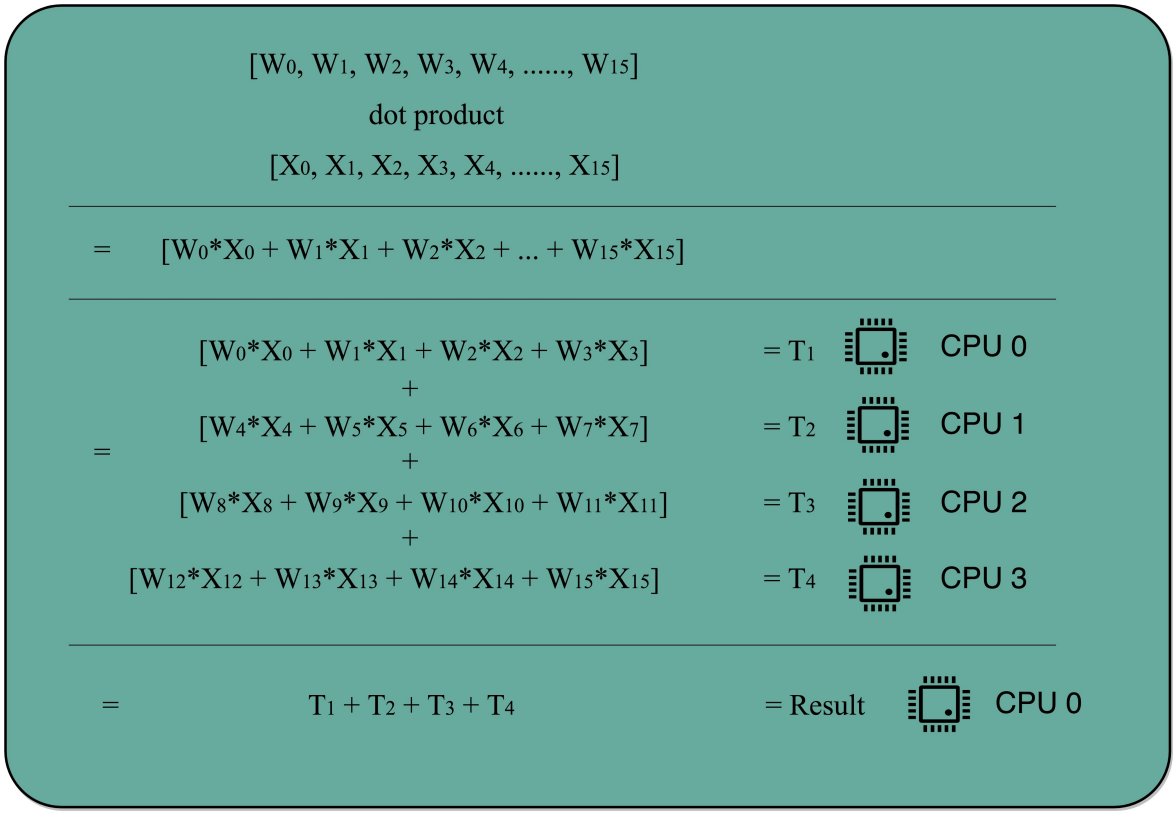
提升响应时间，就好比提升你用的交通工具的速度，比如原本你是开汽车，现在变成了火车乃至飞机。本来开车从上海到北京要20个小时，换成飞机就只要2个小时了，但是，在此之上，再想要提升速度就不太容易了。我们的CPU在奔腾4的年代，就好比已经到了飞机这个速度极限。

那你可能要问了，接下来该怎么办呢？相比于给飞机提速，工程师们又想到了新的办法，可以一次同时开2架、4架乃至8架飞机，这就好像我们现在用的2核、4核，乃至8核的CPU。

虽然从上海到北京的时间没有变，但是一次飞8架飞机能够运的东西自然就变多了，也就是所谓的“吞吐率”变大了。所以，不管你有没有需要，现在CPU的性能就是提升了2倍乃至8倍、16倍。这也是一个最常见的提升性能的方式，**通过并行提高性能**。

这个思想在很多地方都可以使用。举个例子，我们做机器学习程序的时候，需要计算向量的点积，比如向量 $W = [W_0, W_1, W_2, ⋯, W_{15}]$和向量 $X = [X_0, X_1, X_2, ⋯, X_{15}]$，$W·X = W_0 * X_0 + W_1 * X_1 +$
$W_2 * X_2 + ⋯ + W_{15} * X_{15}$。这些式子由16个乘法和1个连加组成。如果你自己一个人用笔来算的

话，需要一步一步算16次乘法和15次加法。如果这个时候我们把这个人物分配给4个人，同时去算$W_0 ~ W_3$, $W_4 ~ W_7$, $W_8 ~ W_{11}$, $W_{12} ~ W_{15}$这样四个部分的结果，再由一个人进行汇总，需要的时间就会缩短。



但是，并不是所有问题，都可以通过并行提高性能来解决。如果想要使用这种思想，需要满足这样几个条件。

第一，需要进行的计算，本身可以分解成几个可以并行的任务。好比上面的乘法和加法计算，几个人可以同时进行，不会影响最后的结果。

第二，需要能够分解好问题，并确保几个人的结果能够汇总到一起。

第三，在“汇总”这个阶段，是没有办法并行进行的，还是得顺序执行，一步一步来。

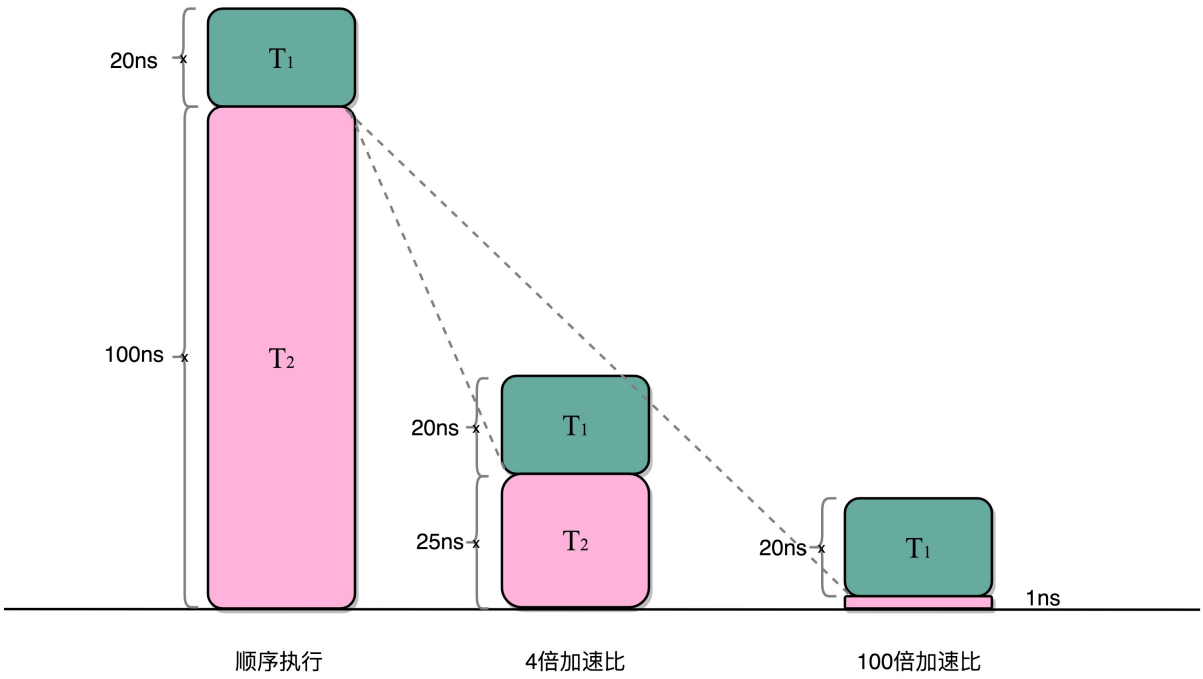
这就引出了我们在进行性能优化中，常常用到的一个经验定律，**阿姆达尔定律**（Amdahl’s Law）。这个定律说的就是，对于一个程序进行优化之后，处理器并行运算之后效率提升的情况。具体可以用这样一个公式来表示：

优化后的执行时间 = 受优化影响的执行时间/加速倍数+不受影响的执行时间

在刚刚的向量点积例子里，4个人同时计算向量的一小段点积，就是通过并行提高了这部分的计算性能。但是，这4个人的计算结果，最终还是要在一个人那里进行汇总相加。这部分汇总相加的时间，是不能通过并行来优化的，也就是上面的公式里面**不受影响的执行时间**这一部分。

比如上面的各个向量的一小段的点积，需要100ns，加法需要20ns，总共需要120ns。这里通过并行4个CPU有了4倍的加速度。那么最终优化后，就有了100/4+20=45ns。即使我们增加更多的并行度来提供加速倍

数，比如有100个CPU，整个时间也需要100/100+20=21ns。



总结延伸

我们可以看到，无论是简单地通过提升主频，还是增加更多的CPU核心数量，通过并行来提升性能，都会遇到相应的瓶颈。仅仅简单地通过“堆硬件”的方式，在今天已经不能很好地满足我们对于程序性能的期望了。于是，工程师们需要从其他方面开始下功夫了。

在“摩尔定律”和“并行计算”之外，在整个计算机组成层面，还有这样几个原则性的性能提升方法。

1.**加速大概率事件**。最典型的就是，过去几年流行的深度学习，整个计算过程中，99%都是向量和矩阵计算，于是，工程师们通过用GPU替代CPU，大幅度提升了深度学习的模型训练过程。本来一个CPU需要跑几小时甚至几天的程序，GPU只需要几分钟就好了。Google更是不满足于GPU的性能，进一步地推出了TPU。后面的文章，我也会为你讲解GPU和TPU的基本构造和原理。

2.**通过流水线提高性能**。现代的工厂里的生产线叫“流水线”。我们可以把装配iPhone这样的任务拆分成一个个细分的任务，让每个人都只需要处理一道工序，最大化整个工厂的生产效率。类似的，我们的CPU其实就是一个“运算工厂”。我们把CPU指令执行的过程进行拆分，细化运行，也是现代CPU在主频没有办法提升那么多的情况下，性能仍然可以得到提升的重要原因之一。我们在后面也会讲到，现代CPU里是如何通过流水线来提升性能的，以及反面的，过长的流水线会带来什么新的功耗和效率上的负面影响。

3.**通过预测提高性能**。通过预先猜测下一步该干什么，而不是等上一步运行的结果，提前进行运算，也是让程序跑得更快一点的办法。典型的例子就是在一个循环访问数组的时候，凭经验，你也会猜到下一步我们会访问数组的下一项。后面要讲的“分支和冒险”、“局部性原理”这些CPU和存储系统设计方法，其实都是在利用我们对于未来的“预测”，提前进行相应的操作，来提升我们的程序性能。

好了，到这里，我们讲完了计算机组成原理这门课的“前情提要”。一方面，整个组成乃至体系结构，都是基于冯·诺依曼架构组成的软硬件一体的解决方案。另一方面，你需要明白的就是，这里面的方方面面的设

计和考虑，除了体系结构层面的抽象和通用性之外，核心需要考虑的是“性能”问题。

接下来，我们就要开始深入组成原理，从一个程序的运行讲起，开始我们的“机器指令”之旅。

补充阅读

如果你学有余力，关于本节内容，推荐你阅读下面两本书的对应章节，深入研读。

1.《计算机组成与设计：软/硬件接口》（第5版）的1.7和1.10节，也简单介绍了功耗墙和阿姆达尔定律，你可以拿来细细阅读。

2.如果你想对阿姆达尔定律有个更细致的了解，《深入理解计算机系统》（第3版）的1.9节不容错过。

课后思考

我在这一讲里面，介绍了三种常见的性能提升思路，分别是，加速大概率事件、通过流水线提高性能和通过预测提高性能。请你想一下，除了在硬件和指令集的设计层面之外，你在软件开发层面，有用到过类似的思路来解决性能问题吗？

欢迎你在留言区写下你曾遇到的问题，和大家一起分享、探讨。你也可以把今天的文章分享给你朋友，和他一起学习和进步。

精选留言：

- pyhhou 2019-05-01 05:41:10
  对于思考题：
  * 加速大概率事件
  通常我们使用 big-O 去表示一个算法的好坏，我们优化一个算法也是基于 big-O，但是 big-O 其实是一个近似值，就好比一个算法时间复杂度是 O(n^2) + O(n)，这里的 O(n^2) 是占大比重的，特别是当 n 很大的时候，通常我们会忽略掉 O(n)，着手优化 O(n^2) 的部分

* 通过流水线提高性能
能够想到的是任务分解，把一个大的任务分解成好多个小任务，一般来说，分的越细，小任务就会越简单，整个框架、思路也会变得更加清晰

* 通过预测提高性能
常常在计算近似值的时候，例如计算圆周率，我们可以根据条件预设立一个精确率，高过这个精确率就会停止计算，防止无穷无尽的一直计算下去；另外就是深度优先搜索算法里面的 “剪枝策略”，防止没有必要的分支搜索，这会大幅度提升算法效率 [5赞]

作者回复2019-05-02 02:30:51
👍算法的例子举得很好，剪枝策略的例子也很好。

不过流水线和圆周率的例子不太好，可以再想想。

- 活的潇洒 2019-05-01 18:25:41
通读三遍全文，花了3个多小时做了笔记链接如下：
https://www.cnblogs.com/luoahong/p/10800379.html [2赞]

作者回复2019-05-02 01:19:10
👍感谢分享给大家

- 沃野阡陌 2019-05-02 23:04:12
老师，什么是缓存？需要用程序去操作吗？和内存又有什么关系？

- Juexe 2019-05-02 21:22:36
1. 加速大概率事件
可能如 Redis 缓存、CDN 内容分发网络、游戏开发中常用的对象池等

2. 通过流水线提高性能
可能如多线程开发、分布式系统、DDOS攻击等

3.通过预测提高性能
浏览器的一个功能：下一页自动预加载；
Web 开发中用到的一个 InstantClick.js 能够预加载 hover 的链接。

不过「加速大概率事件」和「通过预测提高性能」好像有些重合，分得不是很清楚？

- KR® 2019-05-02 07:41:06
提问, 这里说的预测是硬件cpu层面的预测吗？硬件是固定的，通过什么方式可以预测各种不同软件的下一步呢

- KR® 2019-05-02 07:36:33
对于我这种小白来说，能啃完这些知识点要感谢初中物理老师为我打下的物理基础 哈哈,

徐老师的讲解也太清晰了吧!!!
能看懂跟得上节奏的感觉真好~

还要感谢高阶的同学们，我没有开发经验，看文章时遇到一些专业名词会一脸懵, 好在高阶的同学会在答疑区提问互动, 看你们的提问和回答我都会有收获!

- 活的潇洒 2019-05-02 07:24:43
  对于思考题:

  通过预测提高性能
  3、浏览器缓存
  4、redis缓存
  通过流水线提高
  1、最前段使用类似于F5的硬件设备设备
  2、两台负载nginx负载均衡
  5、微服务springcloud

- 活的潇洒 2019-05-01 18:41:05
  对于思考题:
  1、最前段使用类似于F5的硬件设备设备
  2、两台负载nginx负载均衡
  3、浏览器缓存
  4、消息队列
  5、微服务springcloud

  作者回复2019-05-02 02:36:46
  能具体讲一下你觉得这些问题和哪个性能优化策略对上了么？

- 魏宇靖 2019-05-01 16:57:57
  我对大概率事件的理解是大规模（一系列）即将需要处理的事件，每个个体的概率不小，而且量极大，所以文中说把这些专门交给GPU(TPU)处理可以提高性能
  不知道自己有没有理解偏

  作者回复2019-05-02 04:46:06
  魏宇靖同学你好，这里的大概率事件，就是指实际程序运行频繁发生的事件。比如机器学习里面大量要做矩阵向量运算，所以我们优化矩阵向量运算就能大幅度提高性能。而对于矩阵向量运算，gpu比cpu快很多，所以在这个场景下用gpu运算就比用cpu运算整体性能提升很多

- 易儿易 2019-05-01 15:59:20
  同样主频、核心情况下，低压cpu与标压cpu性能有区别吗？通过公式来看的话应该没有区别，但是经常听到有人讲低压cpu性能打不过标压，对吗？是什么原因呢？

  作者回复2019-05-02 00:56:01
  性能的差异是因为主频就有差异，同样代号的intel cpu，低压的通常主频只有标压的2/3，比如i5-4200m的主频是2.5GHz到3.1GHz。而低压版本的i5-4200u就只有1.6GHz到2.5GHz。它们只是代号相同，主频并不一样

- Geek_fredW 2019-05-01 15:54:55
  我也不明白“加速大概率事件”在文中具体含义。加速可以粗略意识到含义。为什么要提大概率？还是缓存命中？

  作者回复2019-05-02 01:51:24
  Geek_fredW同学你好，因为如果加速的是一个小概率事件，那么对于整体的性能提升就很有限。
  缓存就是一个典型的情况，如果缓存的数据是很少被访问的，加速的就变成了一个小概率事件，那么缓存就并不能提升太多性能也就失去意义了

- 明月 2019-05-01 15:22:21
  一个问题：面积更小使得各个晶体管的距离更短，会加速响应时间吗？我印象中是光速的

  作者回复2019-05-02 01:18:23
  会的，光速也不过就是3*10的八次方，意味着一纳秒也只能走30厘米的距离，所以后面我们还会看到cpu的高速缓存也不能做太大，也是受到光速的限制。

- 须臾即 2019-05-01 14:33:33
  有两个问题没想明白：
  1.增加晶体管怎么提高运算速度？
  提高主频好理解，计算的频繁一些，增加晶体管是干了什么，增加计算单元么，或者说是增加流水线控制单元。
  2.cpu的电压是受了什么因素限制的？
  既然电压低功耗低，那么各厂商应该都想把电压做的越低越好，现实是不容易办到，是哪些因素限制的？

  作者回复2019-05-02 01:56:57
  须臾即他9同学你好
  增加晶体管可以增加硬件能够支持的指令数量，增加数字通路的位数，以及利用好电路天然的并行性，从硬件层面更快地实现特定的指令，所以增加晶体管也是常见的提升cpu性能的一种手段。
  电压的问题在于两个，一个是电压太低就会导致电路无法联通，因为不管用什么作为电路材料，都是有电阻的，所以没有办法无限制降低电压，另外一个是对于工艺的要求也变高了，成本也更贵啊。

- 鸟人 2019-05-01 14:18:18
  通过预测提高性能 我记得有个CPU漏洞就是因为可预测导致数据泄露，现在修复了，然后性能下降 是否意味着以后CPU不会采用预测了呢？

  作者回复2019-05-02 00:48:04
  你说的应该是之前的meltdown和spectre的漏洞，那个漏洞很有意思，可以认为是利用了流水线，预测，以及高速缓存的组合带来的一个问题。
  我不是cpu设计的专家，不过我认为这个并不会让大家放弃预测，而且这个的主要危害其实是在多租户的虚拟机这个层面，完全只考虑物理机的话这个漏洞的触发条件还是很难满足的

- Sentry 2019-05-01 12:46:59
  写程序的时候，可以考虑通过使用缓存，内存的局部性原理等提升程序运行时的性能。

  作者回复2019-05-02 01:58:08
  👍缓存是加速大概率事件的典型案例

- Geek 2019-05-01 12:20:35
  对于思考题，软件层面的话，比如前端页面的首屏渲染，由于浏览器按顺序解析DOM，解析到script标签的时候会加载完再它再继续解析，所以一般把它们放到body底部，这部分对应流水线思想？然后如果script使用了defer，继续Dom解析，，。额，好像对不上“预测”，只能说是不阻塞。

  作者回复2019-05-02 02:15:58
  这两个例子都不太对，再想想

- Geek 2019-05-01 11:59:58
  今天的例子太贴切，非常容易理解，感谢徐老师。我有个问题，如果说单个程序的运行时不可拆分的，如果CPU其他参数一样，那是不是四核和八核处理速度是一样的？提高性能的预测手段，是不是相当于异步呢？

  作者回复2019-05-02 01:39:30

Geek同学你好，是的，这个情况下无论你的cpu有多少核，实际只有一个核在计算。
预测和异步还是不太一样，它更多地是因为现代cpu里的流水线的存在而需要的一种加速性能的办法，我们在后面讲解cpu的分支预测的时候会仔细讲一下这个是为什么

- bo 2019-05-01 11:32:28
  请问谁能解释一下cpu和gpu内部结构的区别吗？

  作者回复2019-05-02 02:16:57
  bo同学你好，在讲解处理器的部分，我会专门有一讲来讲解gpu的，要坚持到底啊

- Leon🐟 2019-05-01 10:00:21
  进程绑定cpu，利用cpu本地缓存，磁盘数据缓存到内存，文件系统利用cache提高读写速度，网络处理让网卡分担一部分cpu的工作，降低对cpu的负载

  作者回复2019-05-02 02:32:09
  👍很多例子能够对得上，要是更具体讲一下是和哪个性能优化的策略对上就更好了

- Linuxer 2019-05-01 09:21:36
  子任务并发算是流水线的应用，磁盘预读算是预测

  作者回复2019-05-02 02:31:14
  👍两个例子都不错