第06章：存储器层次结构

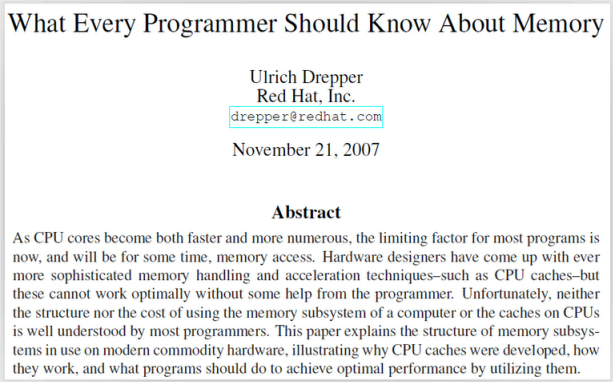
万能的金字塔模型

视频解说

导读

为什么你（程序员）需要理解内存？推荐你认真阅读一下Ulrich Drepper撰写的长达114页的经典论文：What Every Programmer Should Know About Memory，如果你实在没有耐心看完它，或者想了解其中的重点内容，那么也可以通过观看我自己录制的小视频来了解其中的重点内容：

每个程序员都应该知道的内存知识（第1部分：内存基础），每个程序员都应该知道的内存知识（第2部分：CPU缓存），每个程序员都应该知道的内存知识（第4部分：实践部分）



What Every Programmer Should Know About Memory

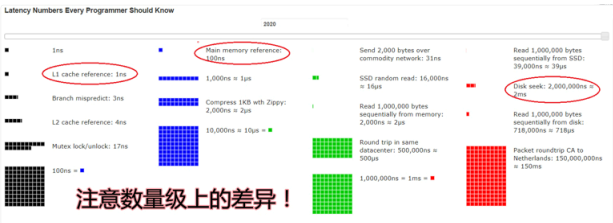
Ulrich Drepper
Red Hat, Inc.
drepper@redhat.com

November 21, 2007

Abstract

As CPU cores become both faster and more numerous, the limiting factor for most programs is now, and will be for some time, memory access. Hardware designers have come up with ever more sophisticated memory handling and acceleration techniques–such as CPU caches–but these cannot work optimally without some help from the programmer. Unfortunately, neither the structure nor the cost of using the memory subsystem of a computer or the caches on CPUs is well understood by most programmers. This paper explains the structure of memory subsystems in use on modern commodity hardware, illustrating why CPU caches were developed, how they work, and what programs should do to achieve optimal performance by utilizing them.

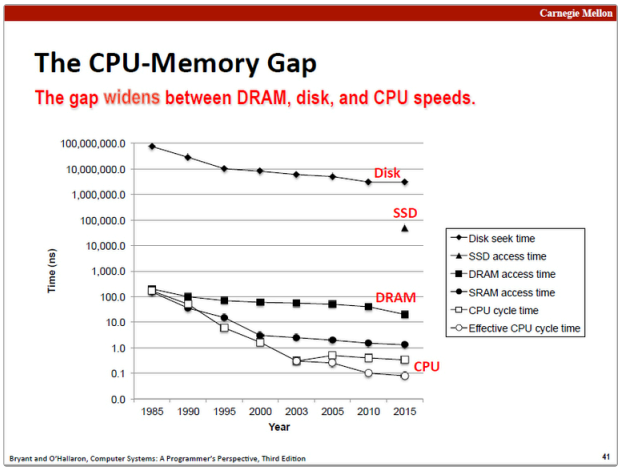
重点提示： Latency Numbers Every Programmer Should Know（数量级上的差异，这是引入缓存的原因）



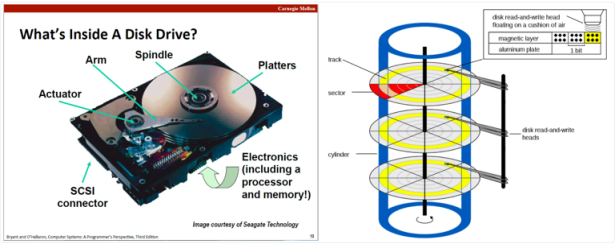
学习方式

CMU教授的视频教程 - Lecture11：内存层次结构 (Memory hierarchy)

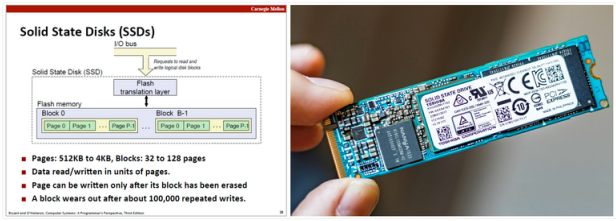
随着工艺的提升，最近几十年CPU的频率不断提升，而受制于制造工艺和成本限制，计算机的内存（主要是DRAM）在访问速度上没有质的突破。因此，CPU的处理速度和内存的访问速度差距越来越大，**如何跨越CPU和内存之间的鸿沟？**聪明的硬件工程师引入了一种性价比极高的的技术，即基于SRAM技术的缓存。



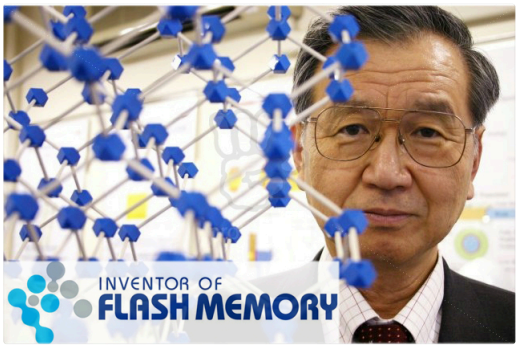
磁盘的访问为什么那么慢？想一想它的物理特性，寻道时间是机械移动的时间，很难再有突破，怎么办？



固态硬盘SSD可以用来部分弥补磁盘和内存之间的速度瓶颈，它利用了Flash闪存技术，今天绝大部分的电子设备，比如笔记本电脑、U盘、手机上都在广泛使用闪存，闪存就是一种特殊的、以块擦写的EEPROM



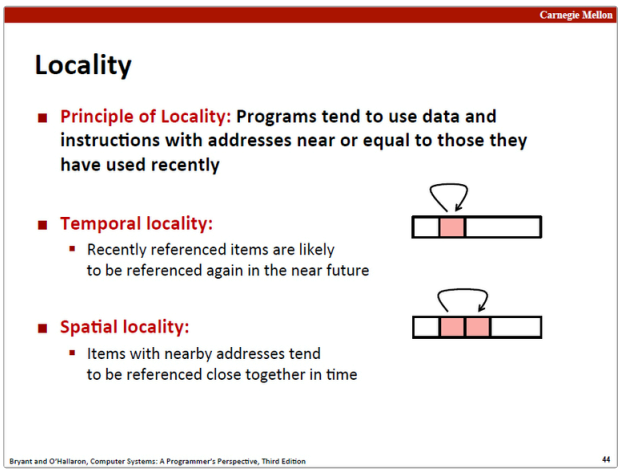
历史故事：闪存是由东芝公司的Fujio Masuoka在1984年首先提出的，他还是日本东北大学的教授，大家可能不知道的是，Intel公司在成立的初期（也就是在转行做处理器之前）是做存储器的，Intel 公司在1988年推出了全球首款256K NOR Flash芯片，不过，很快就被NAND Flash抢了风头... 展望未来，SSD能否完全取代HDD？我想这个事情大概会受到两个因素影响：一个是价格因素（比如3D NAND采用多层堆叠降低颗粒的成本），一个是存储系统具体应用情况，闪存发展趋势还是明朗的，只是完全替代还需要很长时间罢了...



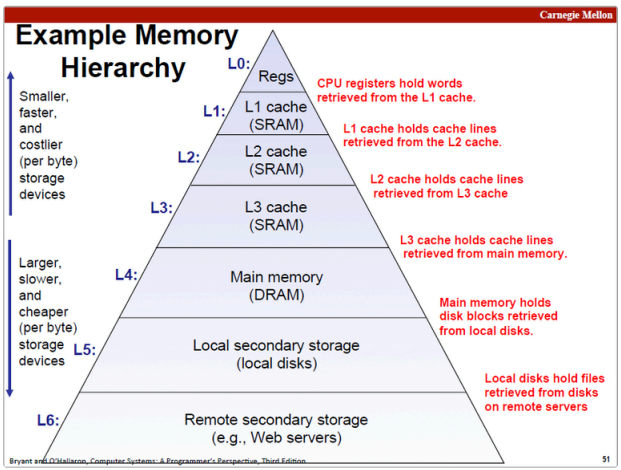
为什么缓存会有用呢? 多亏了我们还有局部性原理！！！

时间局部性（*Temporal*）：最近被访问过的（指令或数据）可能会再次被访问（比如循环，局部变量）

空间局部性（*Spatial*）：被访问的存储单元附近的内容可能很快也会被访问（比如数组）

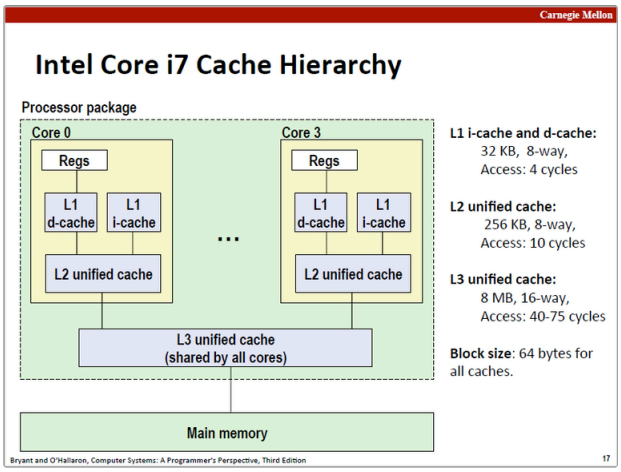


随着数据越来越大，增加一级缓存大小的性价比已经很低了，因此，在一级缓存(L1 Cache)和内存之间又增加一层访问速度和成本介于两者之间的二级缓存(L2 Cache)，后来又增加了三级缓存(L3 Cache)，甚至是四级缓存(L4 Cache)，最后形成了金字塔形的**存储器层次结构**（缓存命中率已经达到惊人的95%以上！！！）

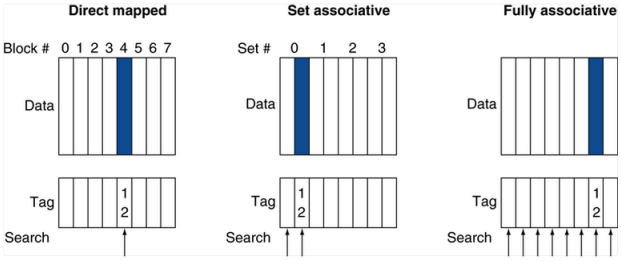


CMU教授的视频教程 - Lecture12：缓存 (Cache Memories)

我们来看一个真实的Intel Core i7 CPU的缓存结构示意图



Cache 是如何存放数据的? 三种形态：Direct Mapped，Fully Associative,，n-way associative



Cache 不命中的场景有哪些? 三种场景：1. Compulsory miss（强制性不命中），例如首次访问，空的缓存必定会造成不命中，2. Capacity miss（容量原因的不命中），即缓存行的太小，不足以存储内存的内容 3. Conflict miss（冲突原因不命中）

由于会发生冲突，某些时候就需要把一部分缓存行驱逐出去缓存，那么到底选哪个倒霉蛋呢？最常见的Cache替换算法有：随机（Random），先进先出（FIFO），最近最少使用（LRU）

提示：需要考虑缓存行的大小，现代CPU的缓存行一般是64个字节（也有使用128个字节的），在Cache的总容量固定且使用组关联的的情况下，缓存行越大，那么能够存放的组就越少，这里需要做一个权衡。

Cache 的写策略有哪些?

Cache命中时的写策略 ① 写穿透（Write Through）：数据同时写入Cache和主存 ② 写返回（Write Back）：数据只写入Cache（标记为dirty），仅当该数据块被替换时才将数据写回主存

Cache不命中时的写策略 ① 写不分配（Write Non-Allocate）：直接将数据写入主存 ② 写分配（Write Allocate）：将该数据所在的块读入Cache后，再将数据写入Cache

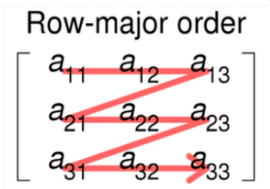
典型情况：① Write--through + No--write--allocate ② Write--back + Write--allocate

重点示例

行序和列序访问对性能产生的影响（理解缓存在其中的作用：Row-major and column-major order ）

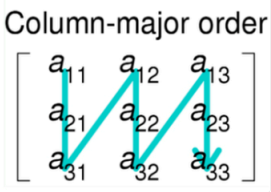
```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

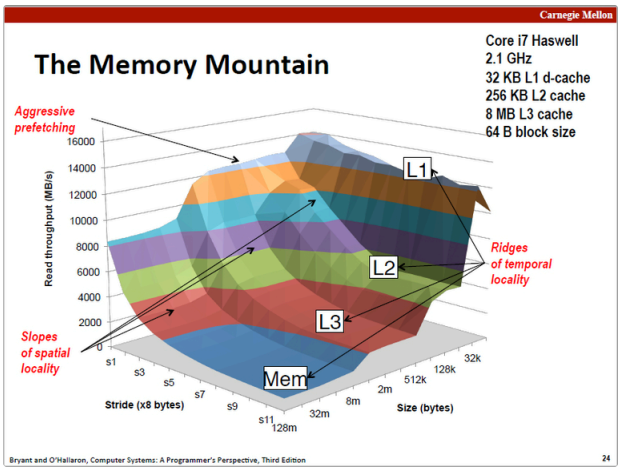



```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```



存储器山（Memory Mountain）（不同的CPU有不同的存储器山）



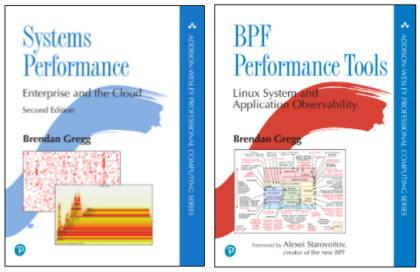
Prefetching：除了缓存，现代处理器还会执行硬件/软件预取（Prefetching），所谓的软件预取，就是在程序中插入一些提示，期待编译器在编译时会插入Prefetch相应的指令，这里以 Intel 为例，Intel 的 SSE SIMD 指令集提供了 Prefetch 相关的指令，示例如下所示，另外还可以参考 GNU官方对于Prefetch的说明

```
#include <mmintrinsics.h>
void _mm_prefetch(char * p , int i ); // 其中 p 是数据所在的内存地址，i 是要载入哪一个层级的Cache
```

总的来说，CPU Cache对于程序员是透明的，所有的操作和策略都在CPU内部完成。但是，了解和理解CPU Cache的设计、工作原理有利于我们更好的利用CPU Cache，写出更多对CPU Cache友好的程序！

延伸阅读

- 苏黎世联邦理工：计算机体系结构(2020年最新版) - 授课教授Onur Mutlu，里面讲了很多内存相关议题
- Linux性能工具(含内存工具)，推荐你看大牛Brendan Gregg的网站：http://www.brendangregg.com/



Brendan Gregg出版的图书

Previous
第05章：优化程序性能

Next
第07章：链接

Last updated 3 years ago