**深入理解计算机存储器层次结构**

摘要：如果程序员理解了系统是如何将数据在存储器层次结构中上上下下移动的，那么就可以将自己编写的应用程序地数据项存储在层次结构中较高的地方。介绍了不同的存储技术包括SRAM、DRAM和磁盘存储。强调了局部性地重要性和影响局部性地一些条件。简单说明了缓存的概念和高速缓存的一些实现方式

**一、存储技术**

**1. 随机访问存储器**

随机访问存储器分为两类：静态的和动态的。静态RAM(SRAM)比动态RAM(DRAM)更快，但也贵的多。SRAM作为高速缓存存储器，既可以再CPU芯片上，也可以在片下。DRAM用来作为主存以及图形系统的帧缓冲区。典型的，一个桌面系统的SRAM不会超过几兆字节，但是DRAM却又几百或几千兆字节，

**(1) 静态RAM**

SRAM将每个位存储在一个双稳态的(bistable)存储单元里。每个单元是用一个六晶体管电路来实现的。每个电路有这样一个属性，它可以无限期的保持在两个不同的电压配置(configuration)或状态(state)直以。其他任何状态都属不稳定的——从不稳定状态开始，电路会迅速转移到两个稳定状态中的一个。这样一个存储器单元类似于一个倒转的钟摆

**(2) 动态RAM**

DRAM将每个位存储位对一个电容的充电。这个电容非常小，大约只有30毫微微法拉。DRAM可以制造的而非常密集——每个单元由一个电容和一个访问晶体管组成。它和SRAM不同，DRAM存储单元对干扰非常敏感。当电容电压被扰乱之后，它就永远不会恢复了。

**2. 磁盘存储**

磁盘是广为应用的保存大量数据的存储设备，存储数据的数量级可以达到几百到几千千兆字节，而基于RAM的存储器只能有几百或几千兆字节。不过，从磁盘上读信息的时间为毫秒级，比从DRAM读慢了10万倍，比从SRAM读满了100万倍。

**(1) 磁盘构造** 磁盘是由盘片(platter)构成的．每个盘片有两面或者称为表面(suffer)．表面種盖着磁性记录材料．盘片中央有一个可以旋转的主轴(spindle)，它使得盘片以固定的旋转速率(rotational rate)旋转．通常是5400～15000转每分钟（RevolutionPerMinute,RPM)。磁盘常包含一个或多个这样的盘片，并封装在一个密封的容器内。

每个表面是由一组称为磁道(track)的同心组成的．每个磁道被划分为一组扇区(sector)。每个扇区包含相等数量的数据位(通常是512字节)，这些数据编码在区上的磁性材料中．扇区之间由一些间隙（gap）分开，这些间隙中不存储数据位。间隙存储用来标识扇区的格式化位。

磁盘是由一个或多个叠放在一起的盘片组成的，它们被封装在一个巒封的包装里。整个装置通常被称为磁盘驱动器(diskdrive)，我们通常简称为磁盘(disk)。有时，我们会称磁盘为旋栲磁盘(rotating disk)•以使之区别于基于闪存的态硬盘(SSD)，SSD是没有移动部分的。

磁盘造商涌常用术语柱面(cylinder)来描述多个盘片驱动器的构造，这里，柱面是所有盘片表面上到主轴中心的距离相等的磁道的集合·例如，如果一个驱动器有三个盘片和六个面，每个表面上的磁道的编号都是一致的，那么柱面k就是6个磁道k的集合。

**(2) 磁盘容量**

一个磁盘上可以记录的最大位数称为它的最大容量，或者简称为容量，磁盘容量是由一下技术因素决定的。

* 记录密度(recording density)(位/英寸): 磁道一英寸的段中可以放入的尾数。
* 磁道密度(track density)(道/英寸): 从盘片中心出发半径上一英寸的段内可以有的磁道数。
* 面密度(areal density)(位/平方英寸): 记录密度和磁道密度的乘积。

磁盘容量公式：

磁盘容量=字节数/扇区 \* 平均扇区数/磁道 \* 磁道数/磁面 \* 表面数/盘片 \* 盘片数/磁盘

**(3) 磁盘操作**

磁盘用读/写头(read/write head)来读写存储在磁性表面的位，而读写头连接到一个转动臂(actuator arm)一端。通过沿着半径轴前后移动这个转动臂，驱动器可以将读/写头定位到盘面上的任何磁道上。这样的机械运动成为寻道(seek)。一旦读/写头定位到了期望的磁道上，那么当磁道上的每个位通过他的下面是，读/写头可以感知到这个位的值(读该位),也可以修改这个位值(写该位)。有多个盘片的磁盘针对每个盘面都有一个独立的读/写头。该读/写头垂直排列，一致行动，任何时候，所有的读/写头都位于同一个柱面上。

**3. 固态硬盘**

固态硬盘(Solid State Disk SSD)是一种基于闪存的存储技术，在某些情况下是传统旋转磁盘的极有吸引力的替代品。SSD封装插到I/O总线上标准硬盘插槽(通常是USB或SATA)中，行为就和其他硬盘一样处理来自CPU的读写逻辑磁盘块的请求。一个SSD封装由一个或多个闪存芯片和闪存翻译层(flash translation layer)组成，闪存芯片替代传统的旋转磁盘中的机器驱动器，而闪存翻译层是一个硬件/固件设备，扮演与磁盘控制器相同的角色，将对那个逻辑块的请求翻译成对底层物理设备的访问。

**二、局部性**

一个编写良好的计算机程序常常具有良好的局部性(locality)。也就是，他们倾向于引用临近于其他最近引用过的数据项的数据项，或者最近引用过的数据项本身。这种倾向性，被称为局部性原理(principle of locality),是一个持久的概念，对硬件和软件系统的设计和性能都有极大的影响。

局部性通常有两种不同的形式:时间局部性(temporal locality)和空间局部性(spatial locality)。在一个具有良好的时间局部性的程序中，被引用过一次的内存位置很可能在不远的将来再被多次引用。再一个具有良好空间局部性的程序中，如果一个内存位置被引用了一次，那么程序很可能再不远的将来引用附近的一个内存位置。

**1. 对数据引用的局部性**

下面这个程序变量sum再每次循环迭代中被引用一次，因此，对于sum来说，有好的时间局部性。另一方面，因为sum是标量，对于sum来说，没有空间局部性。

int sumvec(int v[N]){  
int i,sum=0;  
for(i=0;i<N;++i){  
sum += v[i];  
}  
return sum;  
}

上面程序顺序访问每一个向量元素的函数，具有步长位1的引用模式(stride-1 reference pattern)(相对于元素的大小)。在一个连续向量中，每隔k个元素进行访问，就成为步长为k的引用模式。步长位1的引用模式是程序中空间局部性的常见和重要来源。一般而言，随步长增加，空间局部性下降。

对于引用多维数组的程序来说，步长也是一个很重要的问题。sumarrayrow双重嵌套循环按照行优先顺序(row-major order)读数组的元素。他具有良好的空间局部性。因为他按照数组被存储的行优先顺序来访问这个数组，其结果是得到一个很好的步长位1的引用模式。

int sumarrayrows(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;  
}

一些小改动会对他的局部性产生很大影响,sumarraycols函数按列顺序来扫描数组。因为再c语言中数组是按行来存放的，结果就得到了步长位N的引用模式。

int sumarraycols(int a[M][N]){  
int i,j,sum=0;  
​  
for(j=0;i<N;++J){  
for(i=0;j<M;++i){  
sum+=a[i][j];  
}  
}  
​  
return sum;  
}

**2. 取指令的局部性**

因为程序指令是存放在内存中的，CPU必须取出(读出)这些指令，所以我们也能够评价一个程序关于取指令的局部性。代码区别于程序数据的一个重要属性是在运行时他是不能被修改的。当程序正在执行时，CPU只从内存中读出他的指令。CPU很少会重写或修改这些指令

**3. 局部性小结**

* 重复引用相同变量的程序具有良好的时间局性
* 对于具有步长为k的引用模式的程序，步长越小，空间局部性越好。具有步长为1的引用模式的程序有很好的空间局部性。在内存中以大步长跳来跳去的程序空间局部性会很差。
* 对于取指令来说，循环有好的时间和空间局部性。循环体越小，循环迭代次数越少，局部性越好。

**三、存储器层次结构**

一般而言，存储器的层次结构，从高层往低层走，存储设备变得更慢，更便宜和更大。在最高层是少量的快速CPU寄存器，CPU可以在一个时钟周期内访问他们。接下来是一个或多个小型到中型的基于SRAM的高速缓存存储器，可以在几个CPU时钟周期内访问他们。然后是一个大的基于DRAM的主存，可以在几十到几百个时钟周期内访问他们。接下来是慢速但是容量很大的本地磁盘。最后，有些系统甚至包括了一层附加的远程服务器上的磁盘，要通过网络来访问他们。

**1. 存储器层次结构中的缓存**

一般而言，高速缓存(cache)是一个小而快速的存储设备，他作为存储在更大、也更慢的设备中的数据对象的缓冲区域。使用高速缓存的过程称为缓存(caching)。

存储器层次结构的中心思想史，对于每个k，位于k层的更快更小的存储设备作为位于k+1层的更大更慢的存储设备的缓存。换句话说层次结构中的每一层都缓存来自较低一层的数据对象。例如，本地磁盘作为通过网络从远程磁盘取出的文件的缓存，主存作为本地磁盘上数据的缓存，依次类推，知道最小的缓存——CPU寄存器组。

**(1) 缓存命中**

当程序需要第k+1层的某个数据对象，他首先在当前存储在第k层的一个块中查找d。如果d刚好缓存在第k层，那么就是我们说的缓存命中(cache hit)。该程序直接从第k层读取d，根据存储器层次结构的小女孩子，这要比第k+1层读取d更快。

**(2)缓存不命中**

如果第k层中没有缓存数据对象d，那么就是我们所说的缓存不命中(cache miss)。当发生缓存不命中时，第k层的缓存自从第k+1层缓存中取出包含d的那个块，如果第k层的缓存已经满了，可能就会覆盖现存的一个块。

覆盖一个现存的块的过程被称为替换(replacing)或驱逐(evicting)这个块。被驱逐的这个块有时也成为牺牲块(victim block)。决定该替换哪个块是由缓存的替换策略(replacement policy)来控制的。例如随即替换策略、LRU策略。

**(3)缓存不命中的种类**

如果一个缓存是空的，那么对任何对象的访问都不会命中。一个空的缓存有时被称为冷缓存(cold cache),此类命中被称为强制性不命中(compulsory miss)或者冷不命中(cold miss)。冷不命中很重要，因为他们通常是短暂的时间，不会在反复访问存储器使得缓存暖身(warm up)之后出现的稳定状态中出现。

只要发生了不命中，第k层的缓存就必须执行某个放置策略(placement policy),确定把他从第k+1层中取出的块放在哪里，最灵活的替换策略实训与来自第k+ 1层的任何块，放在k层的任何块中。

**四、高速缓存存储器**

**1. 通用的高速缓存存储器组织结构**

考虑一个计算机系统，每个存储器的地址由m为，形成M=2^m个不同的地址。一个机器的高速缓存被组织成一个有S=2^s个高速缓存组(cache set)的数组。每个组包含E个告诉缓存行(cache line).每个是由一个B=2^b字节的数据块(block)组成的，一个有效位(valid bit)指明这个行是否包含有意义的信息，还有t=m-(b+s)个标记位(tag bit)，他们唯一地标识存储在这个高速缓存行中的块。

| **参数** | **表述** |
| --- | --- |
| S=2^s | 组数 |
| E | 每个组的行数 |
| B=2^b | 块大小 |
| m=log2(M) | (主存)物理地址位数 |

**2. 直接映射高速缓存**

根据每个组的高速缓存行数E，高速缓存被分为不同的类。每个组只有一行(E=1)的高速缓存成为直接映射告诉缓存(direct-mapped cache)。直接映射高速缓存是最容易实现和理解的。

**组选择**

高速缓存从w的地址中间抽取出s个组索引位。这些为被解释成为一个对对应于一个组号的无符号整数。

**行匹配**

在上一步中已选择了某个组i，接下来的异步就是要确定是否有字w的一个副本存储在组i包含的一个告诉缓存中，在直接映射高速缓存中这很容易而且很快，因为每个组只有一行，当且仅当设置可有效位，而且高速缓存行中的标记与w的地址中的标想匹配时，这一行包含w的一个服副本。

**字抽取**

一旦命中，我们知道w就在这个块中的某个地方，最后异步确定所需要的字在块中是从哪里开始的。

**不命中时的行替换**

如果缓存不命中，那么太需要从存储器层次结构中的下一层取出被请求的块，然后将新的块存储在组索引位指示的组中的一个高速缓存行中。一般而言，如果族中都是有效高速缓存行了，那么就必须要驱逐出一个现存的行。

**3. 组相联高速缓存**

组相联高速缓存每个组都保存一个或多个高速缓存行。

**组选择**

和直接映射高速缓存的组选择一样，组索引位标识组。

**行匹配和子选择**

他和行匹配比直接映射高速缓存更复杂，因为他必须检查多个行的标记位和有效位，以确定所请求的字是否在集合中。我们可以把组相联高速缓存中的每个组都看成一个小的相联存储器(key,value)，key是标和有效位，而value就是块的内容。这里一个重要思想就是组中的任何一行都可以包含任何映射到这个组的内存块，所以高速缓存必须搜索中的每一行，寻找一个有效的行，其标记与地址中的标记相匹配，如果高速缓存找到了这一行就命中，块偏移从这个块中选择一个字，和前面一样。

**行替换**

最简单的替换策略是随机选择要替换的行，其他更复杂的策略利用了局部性原理，以使在比较近的将来引用被替换的行的概率最小。例如最不常使用和最近最少使用策略。

**4. 全相联高速缓存**

**组选择**

全相联高速缓存中的组选择非常简单，因为只有一个组，地址中没有组索引位，地址只被划分成了一个标记和块偏移。

**行匹配和字选择**

全相联高速缓存中的行匹配和子选择与组相联高速缓存中的是一样的，他们之间的区别是要是规模大小问题。因为高速缓存电路必须并行地搜索许多相匹配地标记，构造一个又大又快相联高速缓存很困难，而且昂贵。因此，全相联高速缓存只适合做小的高速缓存，例如虚拟内存系统中的翻译备用缓冲。

参考资料

[1]兰德尔E.布莱恩特 大卫R.奥哈拉伦. 深入理解计算机系统[M] 机械工业出版社 399-435