Chapter4 存储体系

计算：地址映像，替换算法、命中率

重点：

存储体系性能、程序局部性
虚拟存储器和管理方式
页式虚拟存储器
Cache 替换算法

4.1存储体系概念和并行存储系统

存储器 ≠存储系统

存储器：

种类：主存、Cache、磁盘、光盘etc

材料工艺：ELC MOS SRAM DRAM

访问方式：随机访问、相联访问、块传送

主存：正在运行程序和数据

辅助存储器：等待运行程序和数据

通用寄存器组：最近常用到的数据

存储系统：

定义：两个或两个以上速度容量价格不同的存储器采用一定办法连接成的一个系统

目标：高速、低价、容量大

矛盾：容量大->延迟增加速度降低、贵；速度高：贵

容量：$S_M = W \times L \times m$

W-存储体字长（设计） L-每个存储体字数（工艺） m-存储体个数（设计）

速度：访问时间T_A（CPU等待时间）、存贮周期T_M（一次存取时间）、频宽B_M（传输速率）

解决方法：

1. 改进工艺和技术、降低成本、提高速度
2. 构成并行主存系统（eg：多通道内存）->提升有限、延迟增加、系统效率不高（指令读取不是顺序的）
3. 使用存储器系统 ：主存 辅存->主存速度相对于 CPU 还是太慢
4. 使用存储体系（存储层次）

4.1.2 并行存储系统

特点：一个存储周期内访问多个数据，从而提高主存频宽

类型：

1.单体多字：增加字长（m 倍），一次读出多个字

D：需要位数多的寄存器、多次访问总线

2.多体单字：多个（m 个）存储体构成

A：实际带宽比单体多字高、价格差不多、并行访问

D：访问冲突->模 m 交叉编码（高位交叉(片选）->扩大容量（连续存储、没有并行）；低位交叉->提高速度（连续访问的时候并行读取））

3.多体多字

问题：m变大，总线负载变大、门的级数增加、延迟增加、转移指令使系统效率下降

4.1.3 存储体系的定义与分支

存储体系：对应用程序员透明、对系统程序员：可能透明可能不透明

分支：

虚拟存储系统：解决容量问题（主存-辅存，二级存储层次），主存辅存之间的数据交换由软硬件完成。

Cache：解决速度问题（Cache-主存，二级存储层次）对应用程序猿和系统程序猿都透明，Cache 与主存之间的数据交换由硬件完成。

程序和数据分布在不同层次的存贮器上、如何传送？

逐级传送、预判（预取，程序局部性原理）。

程序局部性=预判的基础：分布不是堆积的，相对聚集成块或页

时间局部性：最近要用的可能就是当前正在使用的-程序循环造成的

空间局部性：最近要用的可能是当前相邻的信息-程序顺序执行造成

结论：

1.M1（离CPU最近的存储器）不必存放整个程序、存放最近使用的块或页即可（时间局部性）

2.调入的时候一起调入数据所在的块或页（空间局部性）

设计原则：

1.一致性原则：同一信息可以处于不同层次的存储器中，但值应保持相同

2.包含原则：高层次的存储器中的信息包含在低层次的存储器中

性能参数：

1.每位价格c

2.命中率H: CPU产生的逻辑地址能在M1中访问到的概率

M1次数R_1, M_2次数R_2，$H=\frac{R_1}{R_1+R_2}$

3.等效访问时间T_A

（1）M1/M2的访问同时启动（M1、M2与CPU都有直接通路）：$T_A=HT_{A1}+(1-H)T_{A2}$

（2）不是同时启动（只能通过M1访问，那么还需要再重新把数据从M2传送到M1当中才能进行访问）：$T_A=HT_{A1}+(1-H)(T_{A1}+T_{A2})=T_{A1}+(1-H)T_{A2}$

访问效率：$e=\frac{T_{A1}}{T_A}$，通常计算用第一种

提高命中率：将M2中相邻的几个单元数据送入M1

预取后的命中率（不命中率降低n倍）：$H'=1-\frac{1-H}{n}=\frac{H+n-1}{n}$

H：原来的命中率　　n：数据块大小*数据重复数用次数（不命中率降低n倍？）

虚拟存储器

定义：大容量存储器逻辑模型：指的是 主存辅存层次

解决容量问题

特点：

1. 对应用程序猿透明、对系统程序猿基本不透明
2. 依赖局部性原理、提高了主存利用率
3. 每个程序有独立的程序空间
4. 提供了主存保护机制

三个地址与空间：

- **程序空间与程序地址：**
  - **程序空间：程序员用来编写程序的地址空间。也称为虚空间、虚拟存储空间、逻辑空间。**
  - **程序地址：程序员编写程序时所使用的地址，也称为虚地址、逻辑地址。**
- **主存空间与主存地址：**
  - **主存空间：主存储器的地址空间。也称为主存地址空间、主存物理空间或实存地址空间。**
  - **主存地址：主存储器的地址。也称为物理地址、实地址 。**
- **辅存空间与辅存地址：**
  - **辅存空间：辅助存储器的地址空间。也称为辅助地址空间、辅助物理空间或实存地址空间。**
  - **辅存地址：辅助存储器的地址。也称为辅存物理地址、辅存实地址。**

程序重定位：地址映射（最终定位到在主存中的地址）

1.静态重定位：执行之前变换为物理地址（装入时装入程序完成变换）

2.动态重定位：在执行过程中（操作系统等完成变换）

页面替换算法：

1.随机(RAND)算法
2.先进先出(FIFO)算法
3.近期最少使用(LFU)算法
4.近期最久未使用(LRU)算法(最常用)

3/4 的区别在哪？

LFU 是一段时间内使用次数最少，比如 10 分钟，通常不用
LRU 使用循环堆栈方法就可以计算。

解决 CPU-主存瓶颈的办法：

1. 改进工艺和设计，提高存储器的性能
2. CPU 主存之间加 Cache

Cache 特点：

1. 与 CPU 相同工艺
2. 地址变换、页面调度专门的硬件实现
3. 与 CPU 很近或者放在 CPU 中，减少与 CPU 之间的传输延迟
4. Cache 对于所有应用程序员透明
5. Cache 访问主存的优先级高于其他系统访问主存的优先级
6. Cache 与 CPU 之间有直接通路，主存和 CPU 之间也有直接通路，可以实现读直达和写直达

Cache 与虚拟存储器的区别：

存储层次 / 比较项目	Cache	虚拟存储器
目　的	弥补主存速度的	弥补主存容量的不足
存储管理实现	由专用硬件实现	软件、硬件实现
访问速度的比值(第一级和第二级)	几比一	几百比一
典型的块(页)大小	几十个字节	几百到几千个字节
CPU对第二级的访问方式	可直接访问	均通过第一级
失效时CPU是否切换	不切换	切换到其他进程
透明性	对所有程序员透明	仅对应用程序员透明

Cache 系统需要解决的三个问题：

1. 定位问题
2. 替换问题
3. 数据一致性问题

典型地址映像与变换方法：计算整理

1. 全相联映像与变换
2. 直接映像与变换
3. 组相联映像与变换（组内直接映像，组间全相联）

全相联：有空就坐

优点：块冲突概率低，Cache 空间利用率高。

缺点：所需容量的相联存储器代价高

直接相联：每一个主存块只能对应指定一块 Cache

优点：硬件简单成本低，节省地址变换时间

缺点：块冲突概率高，Cache 空间利用率低，很少使用

Cache 典型替换算法：计算整理

1. 随机算法：随机选择一块替换
2. FIFO：最早装入的块替换
3. LRU 近期最少被 CPU 访问的替换

实现：全部使用硬件，两种方法：堆栈法和比较对法（触发器）

Cache 一致性算法：

1. 写直达法：CPU 写操作的时候利用直接通路，同时写入 Cache 和主存
2. 写回法：只有当替换的时候才把修改过的 Cache 写回主存

	写直达法	写回法
可靠性	好于写回法	块替换前仍存在一致性问题
与主存的通信量		少于写直达法达10多倍
控制复杂性	比写回法简单	
硬件实现代价		比写直达法低得多

写调块：写不命中的时候，是否将主存块调入 Cache

1. 不按写分配：不命中直接写主存，不调块
2. 按写分配：写不命中的时候，写入主存并把该单元所在的块调入 Cache

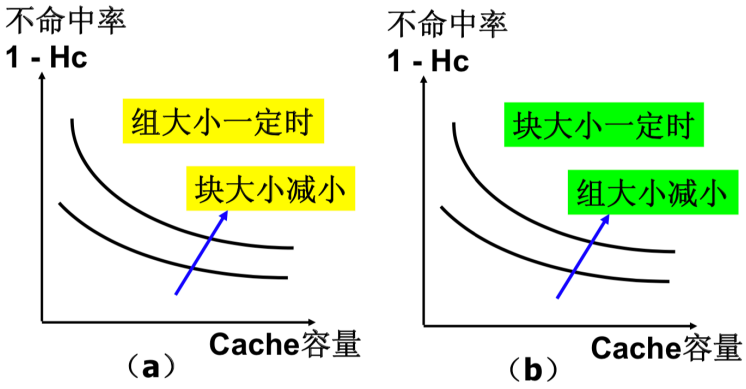
通常写直达法是不按写分配，写回法采用按写分配。

Cache 预取算法：

1. 按需取：不命中把一个块取到 Cache
2. 恒预取：无论是否命中都把下一块取到 Cache
3. 不命中预取：不命中，把本块和下一块一起取到 Cache 中

任务切换对失效率影响解决：

1. 增大 Cache
2. 修改调度算法
3. 设置多 Cache
4. 一些长的向量字符运算不经过 Cache 直接进行



Cache 性能计算：计算整理

Cache 基本优化方法：

1. 降低失效率：
   a. 增加 Cache 块大小
   b. 增加 Cache 容量
   c. 提高相联度
2. 减少失效开销：
   a. 多级 Cache
   b. 使读失效优先于写失效
3. 缩短命中时间
4. 降低失效率：编译优化
5. 并行降低失效代价或失效率：硬件预取，编译器控制预取