# 二． 优化程序性能

## **优化程序性能**

首先使程序在所有可能的情况下都能正常工作，然后想办法让程序跑的更快

优化程序性能的切入点：

1. 选择一组合适的算法和数据结构。
2. 编写出源代码
3. 多线程并行处理运算

## **优化编译器的能力和局限性**

编译器遵循的一个优化程序的原则是：安全优化.编译器都不会对程序进行各种激进的优化，所以程序员必须以一种简化编译器生成高效代码的任务来编写程序

## **表示程序性能**

从程序员的角度来看，用时钟周期来表示度量标准要比用纳秒或者皮秒来表示有用的多。用时钟周期来表示，度量值表示的是执行了多少条指令，而不是时钟运行的有多快。

## **特定体系结构或应用特性的性能优化，限制因素，消除性能瓶颈**

与机器无关:

1. **消除循环的低效率**：将每次循环中执行多次但计算结果不改变的部分提出循环，这样只需计算一次，而不用循环一次，计算一次。以此提高算法效率。
2. **减少过程调用**: 也就是减少函数方法的调用，因为函数方法的调用会带来相当大的开销。但是这样也会带来缺点，就是破坏程序的模块化，所以需要我们权衡利弊。
3. **消除不必要的存储器引用：** 在循环中不停的对指针所指的变量赋值的时候，我们可以用一个中间变量代替指针，以增加速度。
4. **选择合适的算法和数据结构：** 为遇到的问题选择合适的算法和数据结构，避免使用产生糟糕性能的算法或变成技术

与机器相关：

1. **理解现代处理器**： 现代微处理器了不起的成就就是它们采用复杂而奇异的微处理结构，多条指令可以并行执行，同时又呈现出一种简单的顺序执行指令的表象。
2. **提高并行性**: 循环分割，利用功能单元的流水线化的能力提高代码性能。

# 三． 存储结构及虚拟存储器

## **局部性**

局部性分为时间局部性temporal(means relating to time) locality和空间局部性spatial(空间的) locality:

1. 时间局部性：被引用过一次的**内存位置**可能在短时间内被多次引用；
2. 空间局部性：某个**内存位置**被引用了一次，则**短时间内其附近的内存位置被程序引用。**

局部性在硬件层面、操作系统和应用程序中的应用：

1. 计算机引用小而快的cache保存最近使用的指令和数据项；
2. 操作系统中的虚拟内存系统使用内存作为虚拟地址空间最近被引用块的高速缓存；
3. Web浏览器将最近使用的文档放在本地磁盘上。

**怎样提高时间局部性和空间局部性**

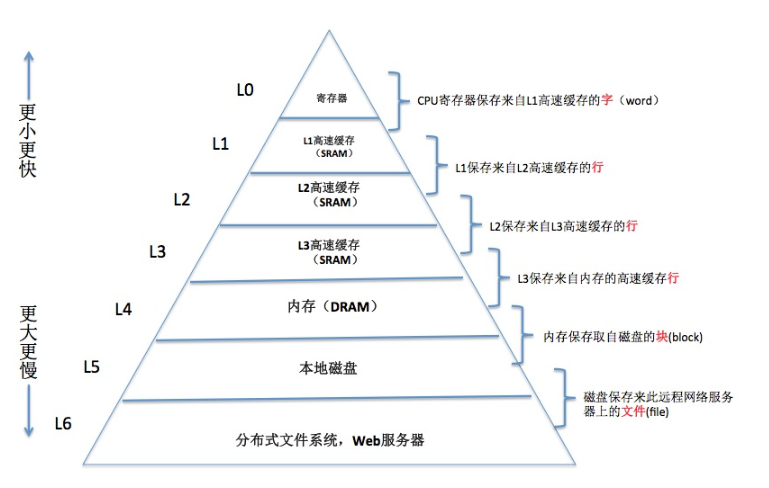
一个连续的向量，每个K个元素进行访问，我们就成为步长为K的引用模式stride-k reference pattern，**步长越长，空间局部性越差，因此我们应该按照数据在内存中的存储位置，按照步长为1的模式读取数据**；

取指令的局部性/编写高速缓存友好的代码：

指令不同于数据，是不能被修改的，cache friendly的代码编写的基本方法：

1. **核心函数(常用函数)的循环应该运行的块，针对其进行优化，其他可以忽略**；
2. 尽量减少核心循环内部的缓存不命中数量；
3. 对局部变量的反复引用是好的，因为他们缓存在寄存器中（时间局部性）；
4. 一旦存储器读取了一个对象，就尽可能多的引用他。

## **存储器层次结构**



## **计算机高速缓存器原理**

高速缓存器是介于中央处理器和主存储器之间的高速小容量存储器

实现原理: 把CPU最近最可能用到的少量信息（数据或指令）从主存复制到CACHE中，当CPU下次再用到这些信息时，它就不必访问慢速的主存，而直接从快速的CACHE中得到，从而提高了速度。

## **高速缓存对性能的影响**

高速缓冲存储器的性能常用命中率来衡量. 影响命中率的因素是高速存储器的容量、存储单元组的大小、组数多少、地址联想比较方法、替换算法、写操作处理方法和程序特性等。

CACHE 的容量与命中率的关系: 虽然容量大一些好，但CACHE 容量达到一定大小之后，再增加其容量对命中率的提高并不明显。

Cache Line Size ( CACHE每次与内存交换信息的单位量)与命中率的关系：   
每次交换信息的单位量适中，不是以一个字为单位，而是以几个字（称为CACHE行容量，通常为4～32个字节）在主存与CACHE之间实现信息传送。

多级的CACHE结构与命中率的关系：

CACHE的不同映像方式与命中率的关系：   
全相联映像方式不适用   
直接映像方式命中率低   
多路组相联方式性能/价格比更好   
直接映像方式中CACHE容量为8K字，被分成1024组，每组8个字，同时，主存也分成8个字的组，1024组构成一页。主存的0组只能映射到CACHE的0组，主存的1组只能映射到CACHE的1组，依次类推。

## **地址空间**

物理地址空间：**程序中访问的内存地址都是实际的物理内存地址。缺点: 1.** 进程地址空间不隔离 2. 内存使用效率低; 3. 程序运行的地址不确定

虚拟地址空间: **程序中访问的内存地址不再是实际的物理内存地址，而是一个虚拟地址，然后由操作系统将这个虚拟地址映射到适当的物理内存地址上**

映射方式: 分页

主存中的每个字节都有一个虚拟地址空间的虚拟地址，和一个物理地址空间的物理地址。

## **虚拟存储器**

虚拟存储器是指具有请求调入功能和置换功能，能从逻辑上对[内存](https://wiki.mbalib.com/wiki/%E5%86%85%E5%AD%98)容量加以扩充的一种存储器系统。虚拟内存的管理

虚拟存储器具有虚拟性、离散性、多次性及强对换性等特征，其中最重要的特征是虚拟性。

虚拟存储器的实现都是建立在离散分配存储管理方式的基础上，目前的实现方法主要有以下两种。(1)请求分页系统 (2)请求分段系统

**Cache vs. 虚拟存储器**

相同： 1. 出发点相同，都是为了提高存储系统的性价比而构造的分层存储系统；2. 原理相同，都是利用程序运行的局部性原理把最近常用的信息调入高速存储

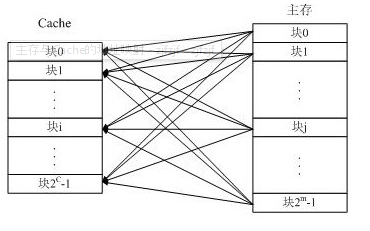
不同: 1. 侧重点不同，catch解决主存与CPU速度差异；虚拟存储器主要解决存储容量问题； 2. 数据通路不同，cache可以直接访问CPU/主存，虚拟存储器只能通过主存调页才能访问CPU; 3. 透明性不同，cache透明由硬件完成，虚存不透明，由操作系统和硬件完成； 4. 未命中损失不同， 主存未命中时系统损失远大于cache未命中

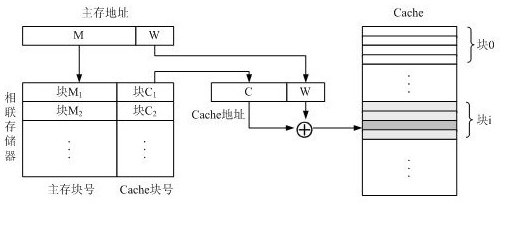
## **翻译和映射**

高速缓存（Cache）与主存地址映射

1. 全相连映射

全相联映射是指主存中任一块都可以映射到Cache中任一块的方式。



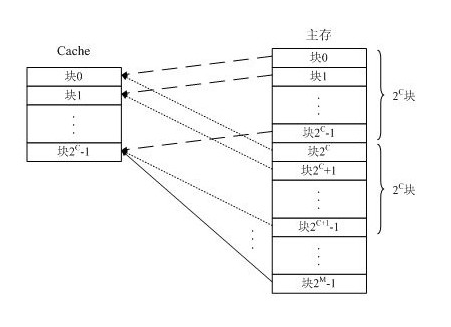


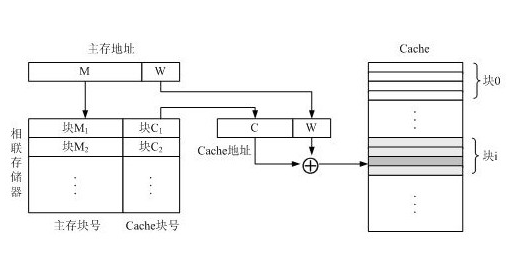
全相联映射方式的优点是Cache的空间利用率高，但缺点是相联存储器庞大，比较电路复杂，因此只适合于小容量的Cache之用。

1. 直接映射方式

直接相联映射方式是指主存的某块j只能映射到满足如下特定关系的Cache块i中

i＝j mod 2C



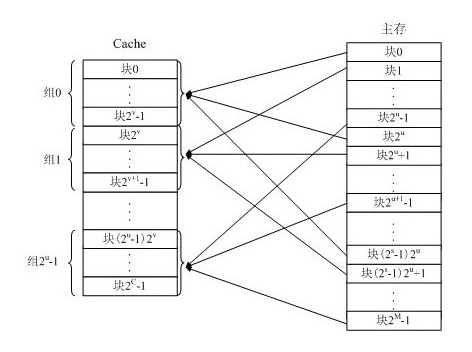


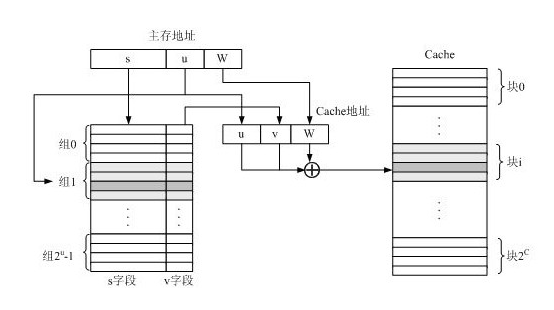
直接相联映射方式的优点 是比较电路最简单，但缺点是Cache块冲突率较高，从而降低了Cache的利用率。由于主存的每一块只能映射到Cache的一个特定块上，当主存的某块 需调入Cache时，如果对应的Cache特定块已被占用，而Cache中的其它块即使空闲，主存的块也只能通过替换的方式调入特定块的位置，不能放置到 其它块的位置上。

1. 组组相连映射方式

将Cache分成2u组，每组包含2v块。主存的块与Cache的组之间采用直接相联映射，而与组内的各块则采用全相联映射。也就是说，主存的某块只能映射到Cache的特定组中的任意一块。主存的某块j与Cache的组k之间满足如下关系：

k＝j mod 2u





## **TLB**

TLB(Translation Lookaside Buffer)转换检测缓冲区（页表缓冲）（虚拟地址到物理地址的转换表）是一个内存管理单元,用于改进虚拟地址到物理地址转换速度的缓存.

## **动态存储器分配和垃圾收集**

需要额外的虚拟存储器时，使用一种动态存储器分配器（dynamic memory allocator）。一个动态存储器分配器维护着一个进程的虚拟存储器区域，称为堆（heap）

分配器将堆视为一组不同大小的块（block）的集合来维护。每个块就是一个连续的虚拟存储器组块（chunk），要么是已分配的，要么是未分配的。

1）显式分配器（explicit allocator）：如通过malloc,free或C++中通过new,delete来分配和释放一个块。

2）隐式分配器（implicit allocator）：也叫做垃圾收集器（garbage collector）。自动释放未使用的已分配的块的过程叫做垃圾回收（garbage collection）。