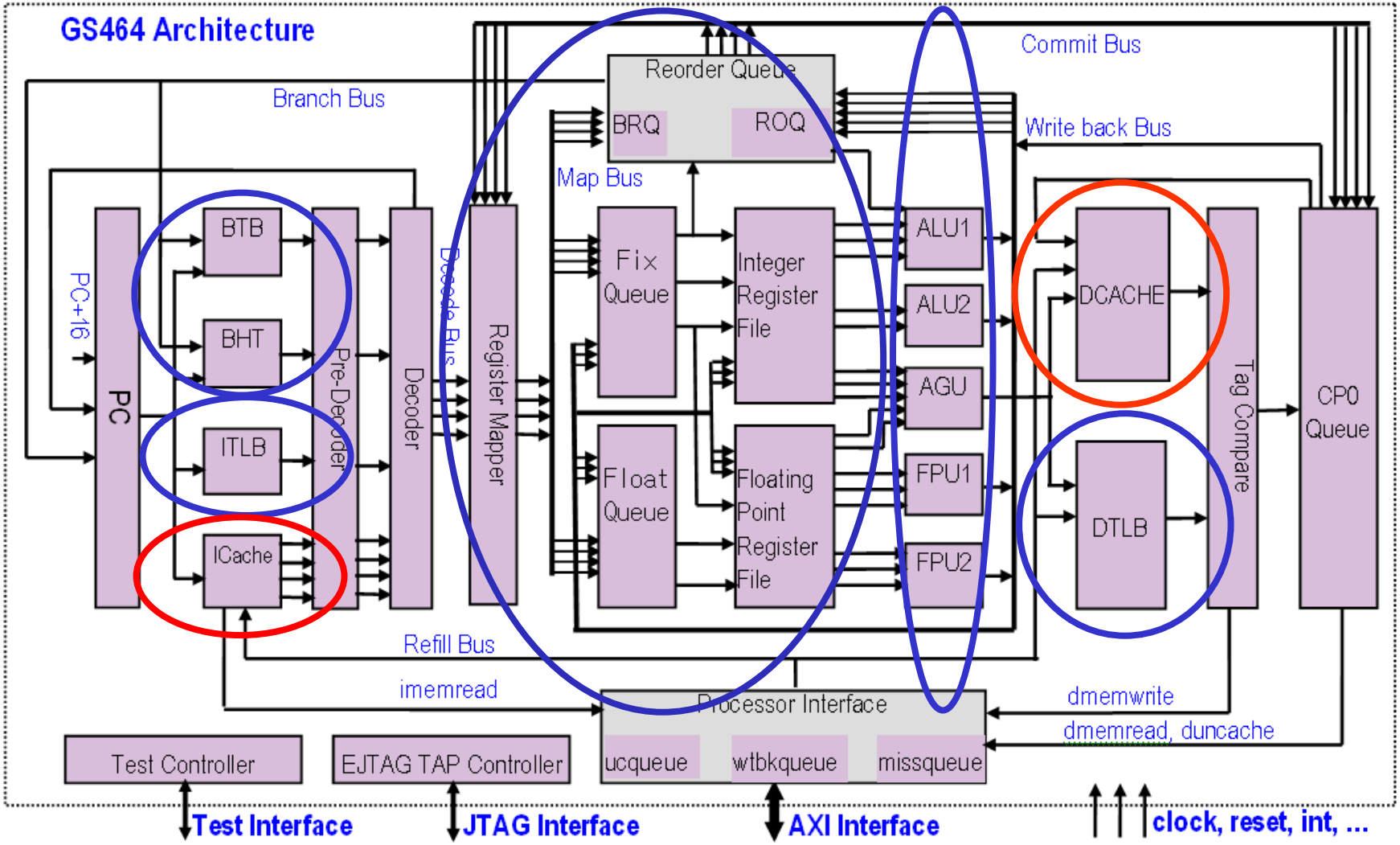
计算机体系结构

胡伟武、汪文祥

龙芯2号处理器核结构图



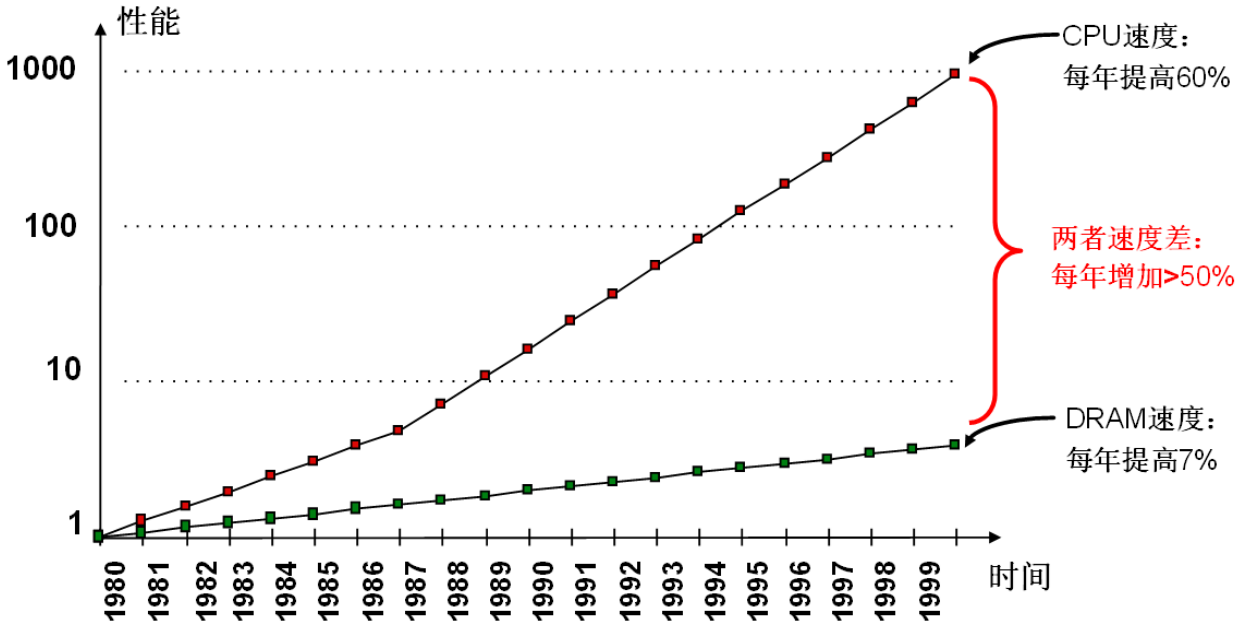
高速缓存（Cache）

- 存储层次的基本概念
- Cache结构
- Cache性能优化
- 常见处理器的存储层次

存储层次的基本概念

CPU与RAM的速度剪刀差

- 摩尔定律
  - **CPU的频率和RAM的容量每18个月翻一番**
  - **但RAM的速度增加缓慢**
- 通过存储层次来弥补差距
  - **寄存器、Cache、存储器、IO**



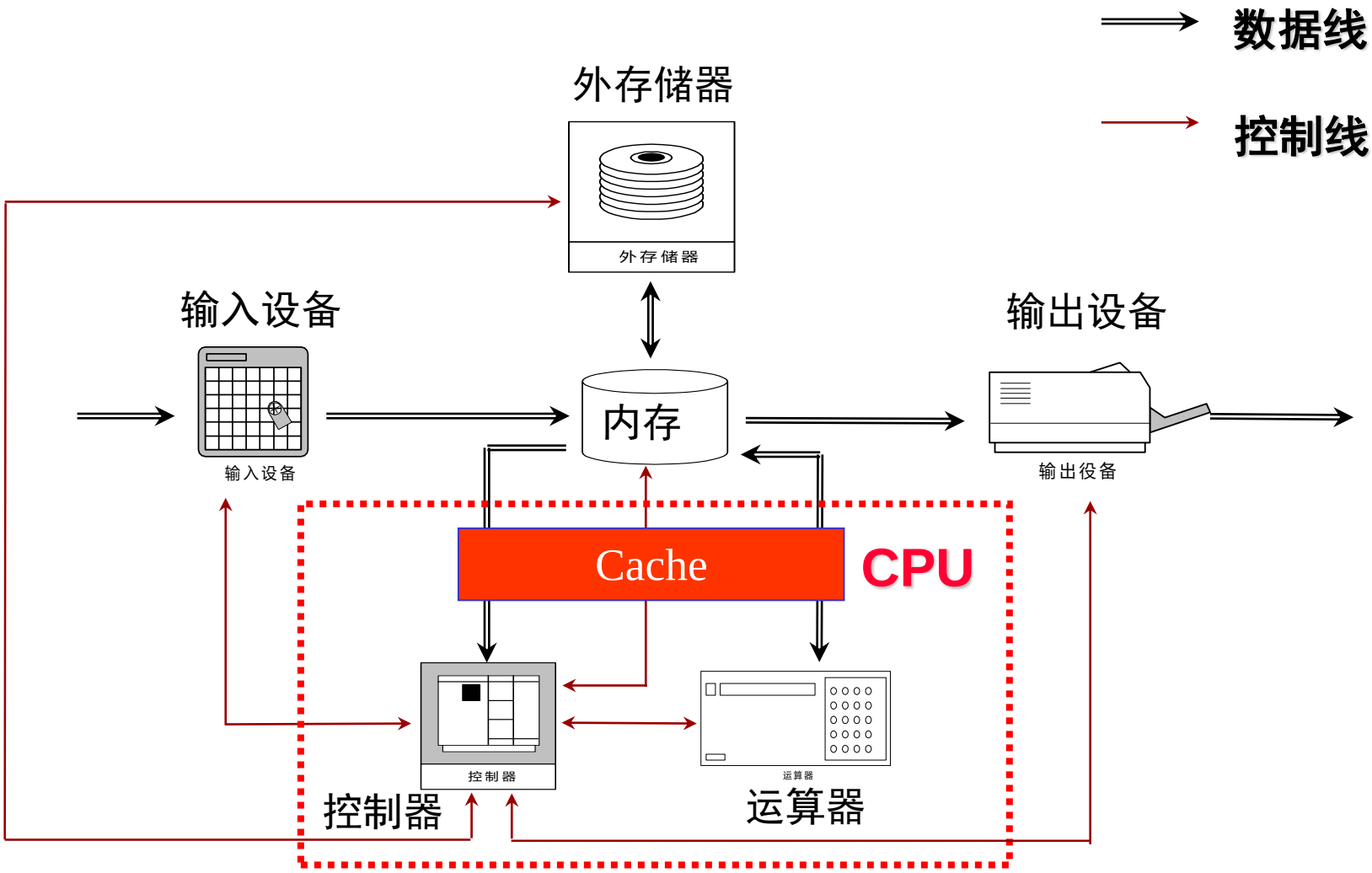
处理器和内存速度剪刀差

- 早期Alpha处理器Cache失效延迟
  - 1st Alpha (7000): 340 ns/5.0 ns = 68 clks x 2 or 136
  - 2nd Alpha (8400): 266 ns/3.3 ns = 80 clks x 4 or 320
  - 3rd Alpha (t.b.d.): 180 ns/1.7 ns =108 clks x 6 or 648
- 当前主流处理器主频2GHz以上
  - IBM Power 6主频6GHz以上
  - 内存延迟50ns左右
  - 访存延迟>100拍
- 多发射加剧了访存瓶颈

摩尔定律使CPU的内容发生了变化

- 冯诺依曼结构的核心思想
  - 存储程序：指令和数据都存放在存储器中
- 计算机的五个组成部分
  - 运算器、控制器、存储器、输入、输出
  - 运算器和控制器合称中央处理器（CPU）
- 为了缓解存储瓶颈，把部分存储器做在片内
  - 现在的CPU芯片：控制器+运算器+部分存储器
  - 片内Cache占了整个芯片的很大一部分面积

计算机硬件系统的组成


数据线
控制线
外存储器
外存储器
输入设备
输入设备
内存
输出设备
输出设备
Cache
CPU
控制器
运算器
控制器
运算器

CPU中RAM的面积和晶体管比例

CPU名称	片内RAM面积	片内RAM晶体管
Alpha 21164	37%	77%
StrongArm SA110	61%	94%
Pentium Pro	64%	88%
龙芯3A	31%	80%

存储层次基本原理

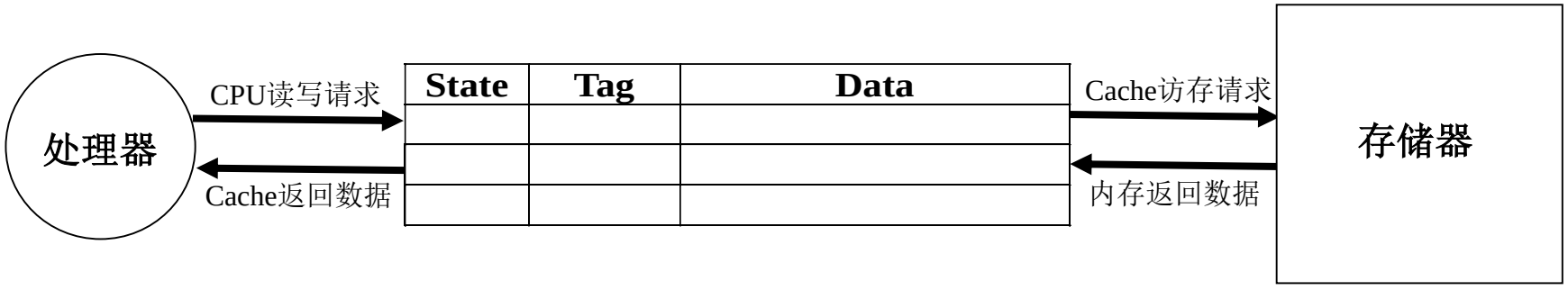
- 程序访问的局部性：时间局部性和空间局部性
  - 新型的应用（如媒体）对传统的局部性提出了挑战
- 越小越简单的硬件越快
  - 越快的硬件越昂贵

存储层次	大小	访问时间	数据调度	调度单位	实现技术
寄存器	<1KB	0.25-0.5ns	编译器	字	CMOS 寄存器堆
CACHE	<16MB	0.5-25ns	硬件	块	CMOS SRAM
存储器	<16GB	80-250ns	操作系统	页	CMOS DRAM
IO 设备	>100GB	5ms	人工	文件	磁盘

Cache结构

Cache的结构

- **Cache的特征**
  - **Cache的内容是主存储器内容的一个子集**
  - **Cache没有程序上的意义，只是为了降低访存延迟**
    - **处理器访问Cache和访问存储器使用相同的地址**
- **Cache的结构特点**
  - **同时存储数据和地址**
    - **通过地址的比较判断相应数据是否在Cache中**
  - **需要考虑所需要的数据不在Cache中的情况**
    - **替换机制，写策略等**

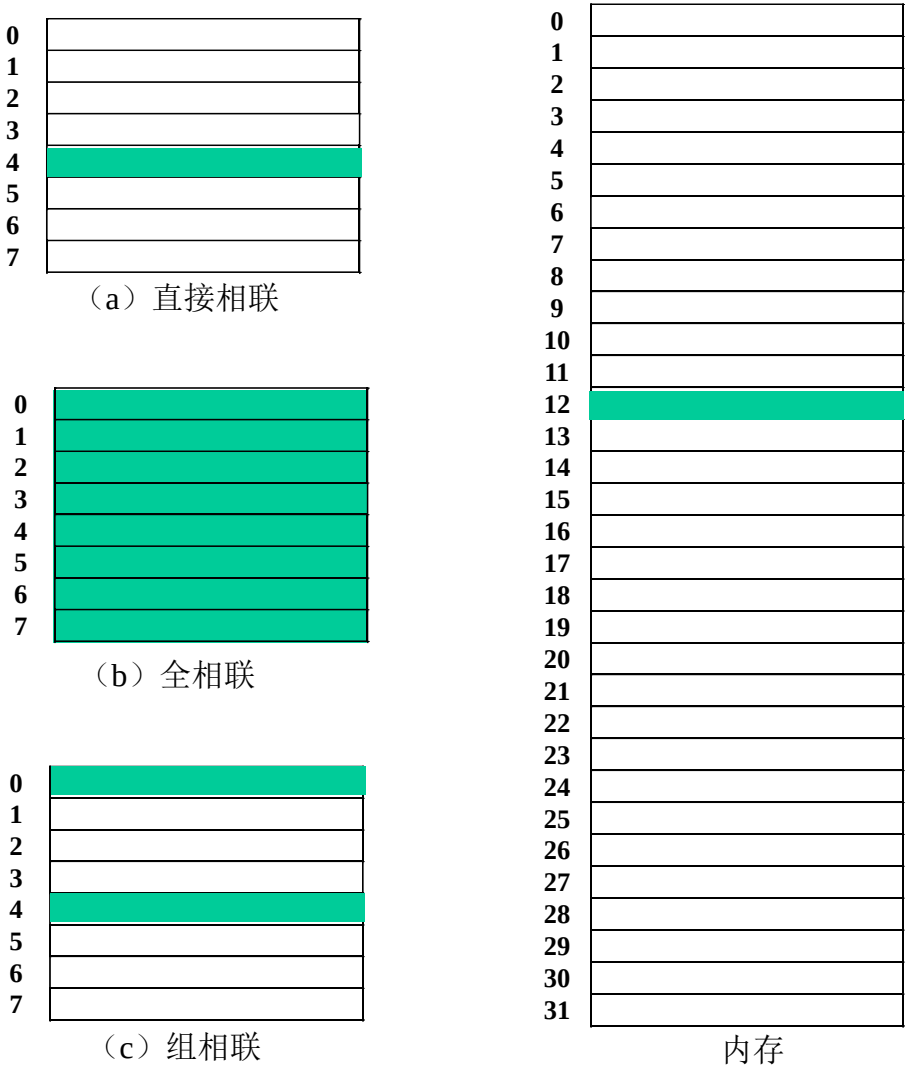


Cache的类型

- **Cache块的位置**
  - **全相联（Fully Associative）**
  - **组相联（ Set Associative）**
  - **直接相联（ Direct Mapped）**
- **Cache失效时的替换机制?**
  - **随机替换, LRU, FIFO**
- **写策略**
  - **写命中策略：写回（Write Back）vs. 写穿透（ Write Through）**
  - **写失效策略：写分配（Write Allocate）vs. 写不分配（Write Non-allocate）**

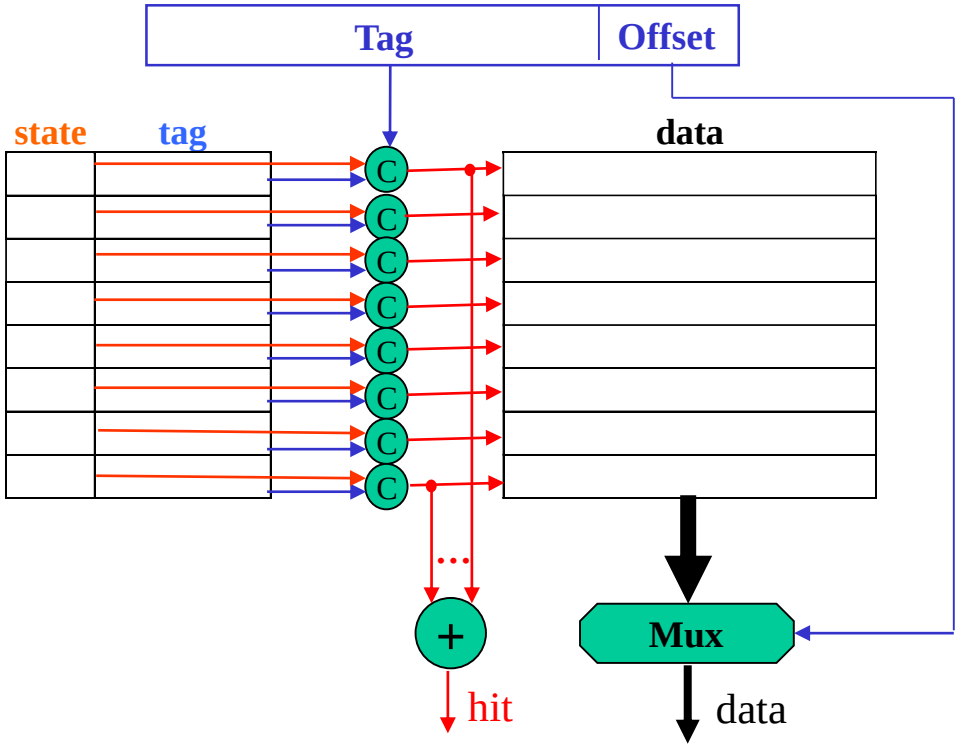
全相联、直接相联、组相联

- 同一单元在不同结构Cache中的位置
  - 直接相联
  - 全相联
  - 组相联



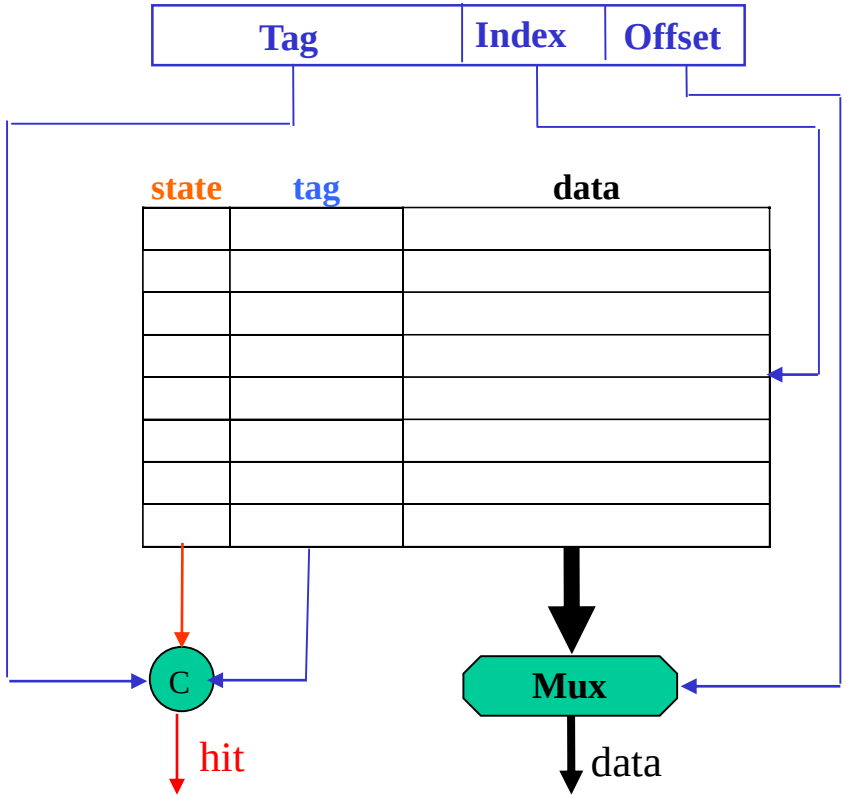
全相联

- 命中率高
- 硬件复杂、延迟大



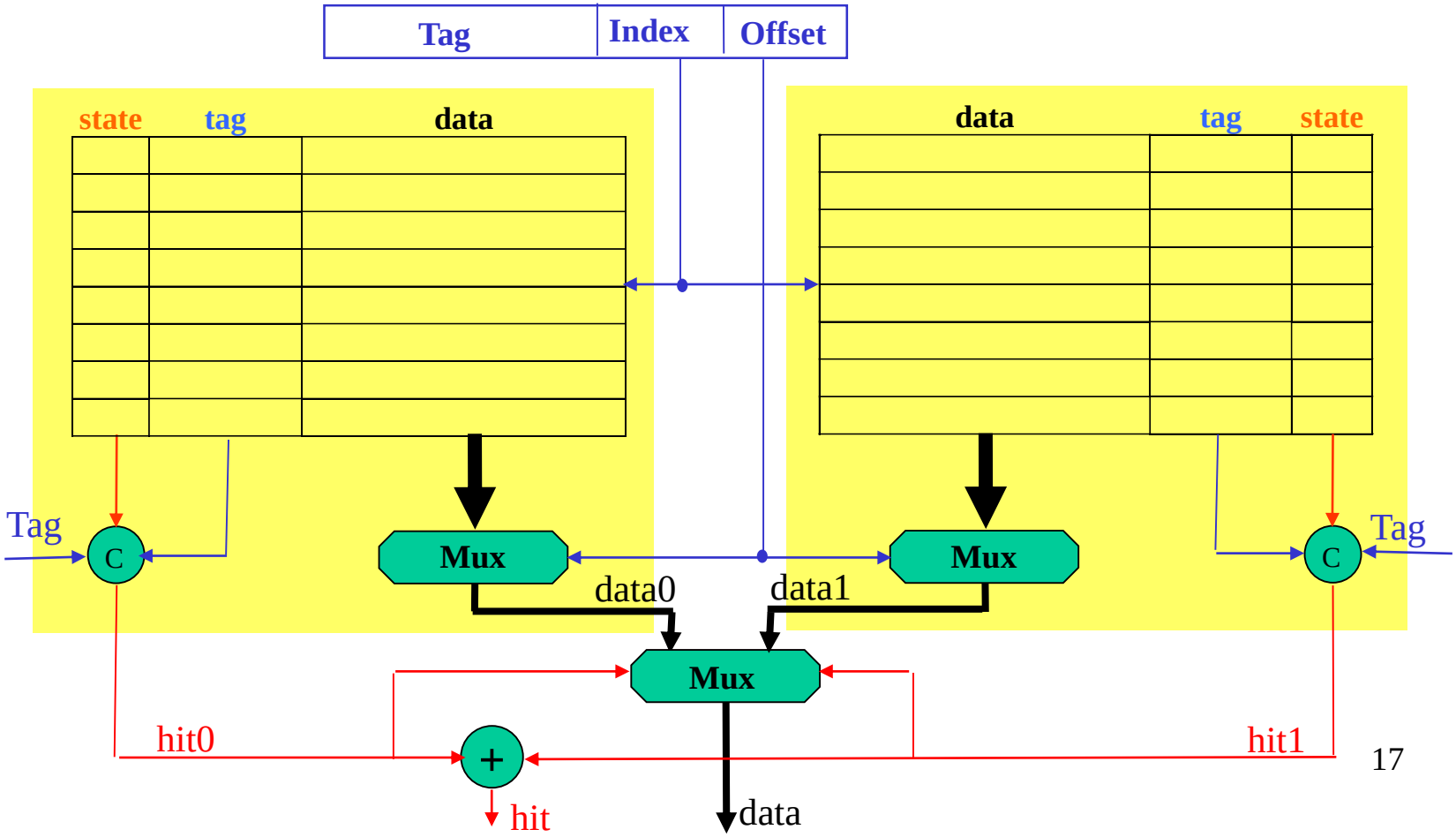
直接相联

- 硬件简单、延迟最小
- 命中率低



组相联

- 介于全相联和直接相联之间



Cache替换算法

- 对直接相联Cache不存在替换算法问题
- 常见的替换算法
  - 随机替换、LRU、FIFO
- 每1000条指令失效次数统计
  - SPEC CPU2000中的gap, gcc, gzip, mcf, perl, applu, art, equake, lucas, swim 10个程序
  - Aplha结构，块大小64B

	2-way			4-way			8-way		
	LRU	Rand	FIFO	LRU	Rand	FIFO	LRU	Rand	FIFO
16KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3

写命中时采取的策略

- 写穿透（Write Through）
  - 写Cache的同时写内存
  - 内存里的数据永远是最新的，Cache替换时直接扔掉
  - Cache块管理简单，只需有效位
- 写回（Write-back）
  - 只写Cache不写内存
  - 替换时要把Cache块写回内存
  - Cache块状态复杂一些，需要有效位和脏位
- 写回/写穿透的使用
  - L1到L2用写穿透的多，L2较快
  - L2到内存用写回的多，内存太慢了
  - 龙芯2号两级都采用写回策略。

写失效时采取的策略

- 写分配（ **Write Allocate** ）
  - 先把失效块读到**Cache**，再在**Cache**中写
  - 一般用在写命中时采用写回策略的**Cache**中
- 写不分配（**Write Non-allocate**）
  - 写**Cache**失效时直接写进内存
  - 一般用在写命中时采用写穿透的**Cache**中

Cache性能优化

Cache性能分析

- **CPU执行时间与访存延迟的关系**
  - **AMAT = Average Memory Access Time**
  - **CPI_{ALUOps} 不包括访存指令**

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

$$AMAT = HitTime + MissRate \times MissPenalty$$
$$= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) +$$
$$(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})$$

Cache性能优化

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

$$AMAT = HitTime + MissRate \times MissPenalty$$

- 降低失效率（**MissRate**）
- 降低失效延迟（**MissPenalty**）
- 降低命中延迟（**HitTime**）
- 提高**Cache**访问并行性

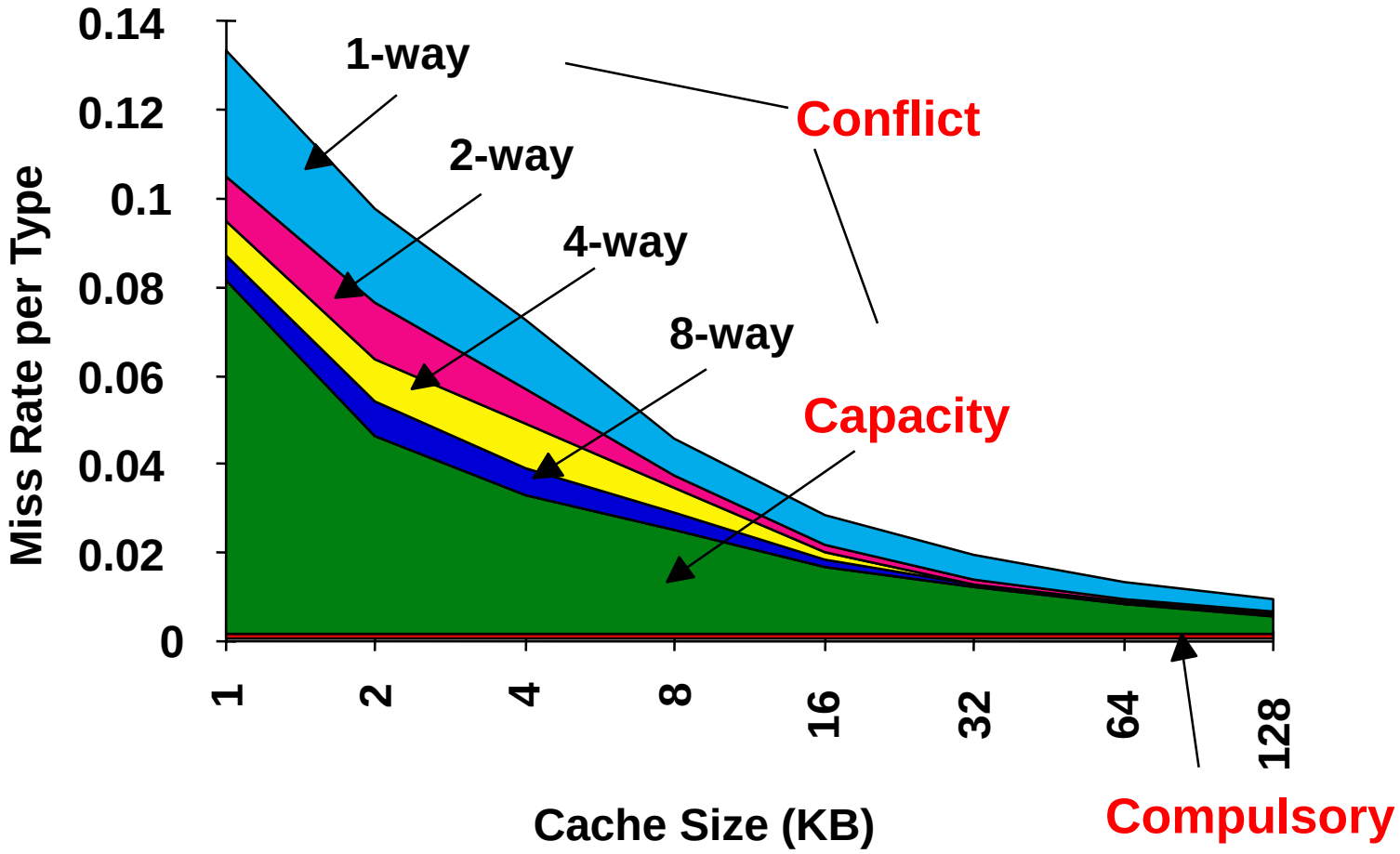
降低失效率

- 增加块大小
- 增加Cache容量
- 增加相联数目
- 路预测（Way Prediction）
- 软件优化

引起Cache失效的因素（3C/4C）

- 冷失效（Cold Miss或Compulsory Miss）
  - CPU第一次访问Cache块时Cache中还没有该Cache块引起的失效
  - 冷失效是不可避免的，即使Cache容量再大也会有
- 容量失效（Capacity Miss）
  - 程序执行过程中，有限的Cache容量导致Cache放不下时替换出部分Cache块，被替换的Cache块再被访问时引起失效
  - 一定容量下全相联Cache中的失效
- 冲突失效（Conflict Miss）
  - 直接相联或组相联Cache中，不同Cache块由于index相同引起冲突
  - 在全相联Cache不存在
- 一致性失效（Coherence Miss）
  - 由于维护Cache一致性引起的失效

3C失效率分析（SPEC92）

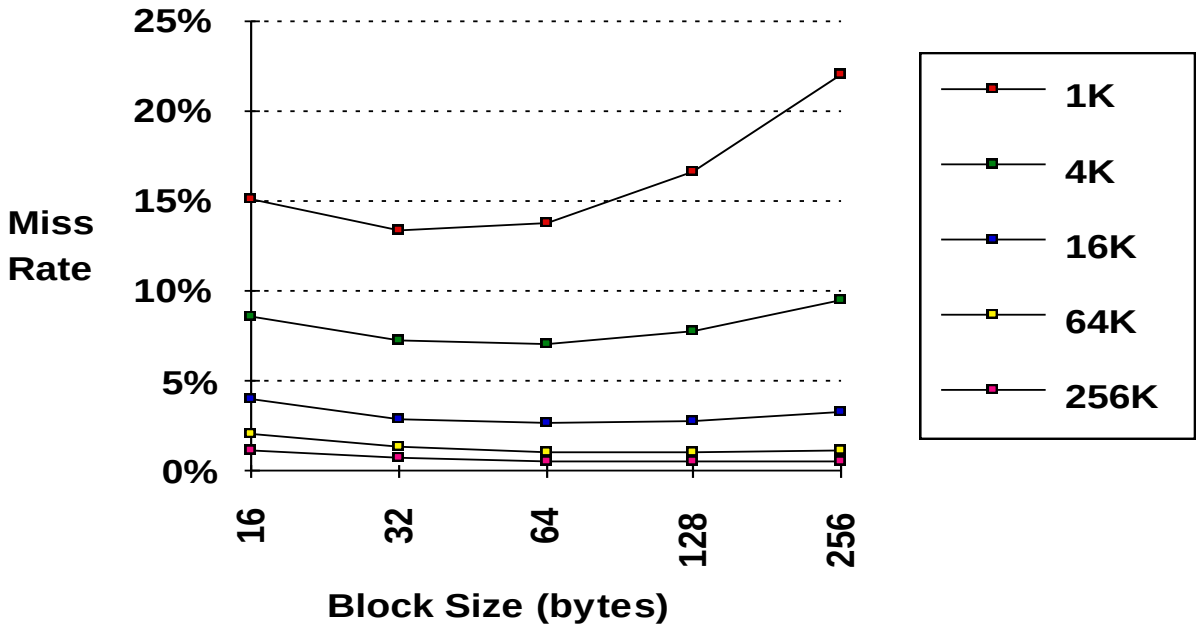


大小（KB）	相联度	失效率	冷失效	容量失效	冲突失效	冷失效百分比	容量失效百分比	冲突失效百分比
4	1-way	0.098	0.0001	0.070	0.027	0.1%	72%	28%
4	2-way	0.076	0.0001	0.070	0.005	0.1%	93%	7%
4	4-way	0.071	0.0001	0.070	0.001	0.1%	99%	1%
4	8-way	0.071	0.0001	0.070	0.000	0.1%	100%	0%
8	1-way	0.068	0.0001	0.044	0.024	0.1%	65%	35%
8	2-way	0.049	0.0001	0.044	0.005	0.1%	90%	10%
8	4-way	0.044	0.0001	0.044	0.000	0.1%	99%	1%
8	8-way	0.044	0.0001	0.044	0.000	0.1%	100%	0%
16	1-way	0.049	0.0001	0.040	0.009	0.1%	82%	17%
16	2-way	0.041	0.0001	0.040	0.001	0.2%	98%	2%
16	4-way	0.041	0.0001	0.040	0.000	0.2%	99%	0%
16	8-way	0.041	0.0001	0.040	0.000	0.2%	100%	0%
32	1-way	0.042	0.0001	0.037	0.005	0.2%	89%	11%
32	2-way	0.038	0.0001	0.037	0.000	0.2%	99%	0%
32	4-way	0.037	0.0001	0.037	0.000	0.2%	100%	0%
32	8-way	0.037	0.0001	0.037	0.000	0.2%	100%	0%
64	1-way	0.037	0.0001	0.028	0.008	0.2%	77%	23%
64	2-way	0.031	0.0001	0.028	0.003	0.2%	91%	9%
64	4-way	0.030	0.0001	0.028	0.001	0.2%	95%	4%
64	8-way	0.029	0.0001	0.028	0.001	0.2%	97%	2%
128	1-way	0.021	0.0001	0.019	0.002	0.3%	91%	8%
128	2-way	0.019	0.0001	0.019	0.000	0.3%	100%	0%
128	4-way	0.019	0.0001	0.019	0.000	0.3%	100%	0%
128	8-way	0.019	0.0001	0.019	0.000	0.3%	100%	0%
256	1-way	0.013	0.0001	0.012	0.001	0.5%	94%	6%
256	2-way	0.012	0.0001	0.012	0.000	0.5%	99%	0%
256	4-way	0.012	0.0001	0.012	0.000	0.5%	99%	0%
256	8-way	0.012	0.0001	0.012	0.000	0.5%	99%	0%

SPEC CPU2000, Alpha结构

通过增加块大小降低失效率

- 利用空间局部性
  - 降低冷失效，增加冲突失效以及容量失效
- **SPEC92，DECstation 5000**
  - 小容量Cache块较小



通过增加Cache大小提高命中率

- 一级Cache访问直接决定时钟周期
  - 尤其是在深亚微米的情况下，连线延迟很大
  - PIII一级Cache为16KB，PIV一级Cache为8KB。
- 增加片内Cache大小增加芯片面积
  - 有的处理器片内Cache面积占整个芯片面积的80%以上
  - 现代处理器二级或三级Cache大小已经达到几MB甚至几十MB。
- 现代通用处理器的一级Cache大小
  - HP PA8700：一级Cache为1MB+1.5MB，没有二级Cache
  - 其他RISC处理器（ Alpha, Power, MIPS, Ultra SPARC）

    32/64KB+32/64KB
  - PIV 12Kop Trace cache+8KB数据Cache（PIII: 16KB+16KB）
  - 反映出设计人员的不同取舍

通过增加相联度降低失效率

- 增加相联度
  - 2:1规则：大小为N的直接相联的Cache命中率与大小为N/2的二路组相联的 Cache命中率相当
  - 八路组相联的效果已经与全相联的失效率相差很小
- 增加相联度会增加时钟周期延迟
  - 例如，直接相连时钟周期为1ns，2路组相联为1.36ns，4路组相联为1.44ns，8路组相联为1.52ns，失效延迟为25时钟周期
  - Cache访问可能是整个处理器的关键路径

Cache容量（KB）	相联度			
	1	2	4	8
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25
128	1.52	1.84	1.92	2.00
256	1.32	1.66	1.74	1.82

通过“路猜测”和“伪相联”提高命中率

- 结合直接相联访问时间短和组相联命中率高的优点
- 在多路相联的结构中，每次Cache访问只检查一路，如果不命中再检查其他路
  - 两种命中：hit和pseudohit
  - 直接判断第0路，或进行Way-Prediction：Alpha 21264 ICache路猜测命中率85%，hit时间为1拍，pseudohit时间3拍
  - 不用并行访问每一路，可以大幅度降低功耗
  - 在片外或L2以下cache中用得较多，如 MIPS R10000、UltraSPARC的L2

通过软件优化降低失效率

- **McFarling [1989]在8KB直接相联的Cache上通过软件优化降低失效率75%**
- **常见软件优化技术**
  - **数组合并（Merging Arrays）**
  - **循环交换（Loop Interchange）**
  - **循环合并（Loop Fusion）**
  - **数组分块（Array Blocking）**

数组合并

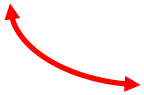
- 通过数组合并降低数组val和key的冲突，增加空间局部性

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of stuctures */
struct merge {
   int val;
   int key;
};
struct merge merged_array[SIZE];
```

循环交换

- 通过循环交换提高空间局部性，把非连续访问变换成连续访问



```
/* Before */
for (k = 0; k < 100; k = k+1)
   for (j = 0; j < 100; j = j+1)
    for (i = 0; i < 5000; i = i+1)
            x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
   for (i = 0; i < 5000; i = i+1)
    for (j = 0; j < 100; j = j+1)
            x[i][j] = 2 * x[i][j];
```

循环合并

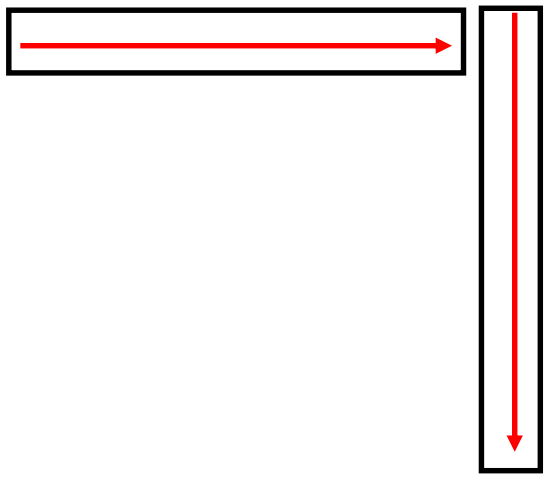
- 通过循环合并提高空间局部性，数组a & c的失效次数从2次降低到1次

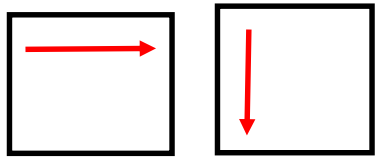
```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
    a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
    d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   {       a[i][j] = 1/b[i][j] * c[i][j];
    d[i][j] = a[i][j] + c[i][j];}
```

数组分块

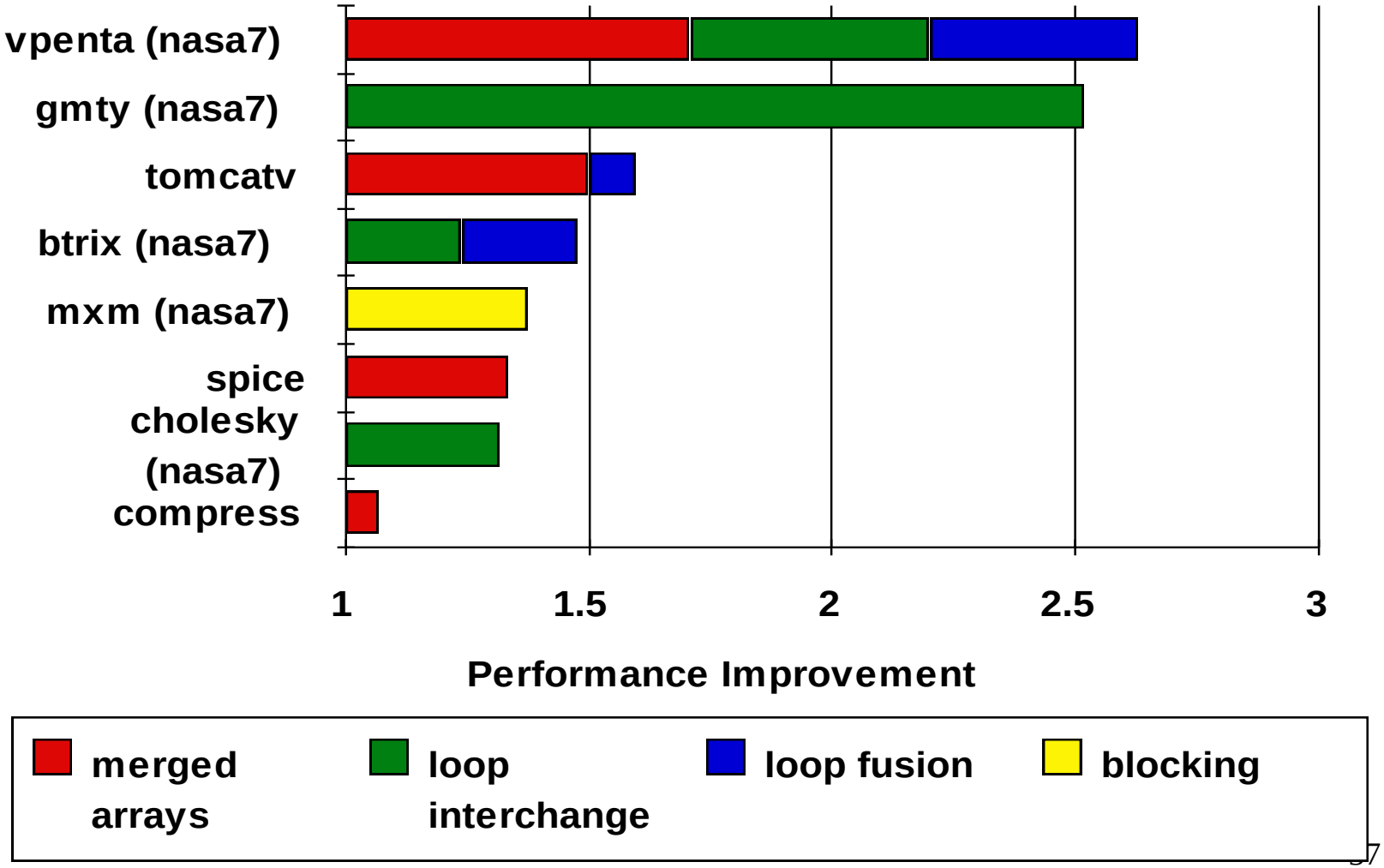
- 分块前，y和z失效N^3次，x失效N^2次，总失效次数从$2N^3 + N^2$
- 分块后，由于可以在cache中放下BxB的小矩阵，y和z失效1次可以用B次，因此失效次数从$2N^3 + N^2$ 降为 $2N^3/B + N^2$

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
    {r = 0;
     for (k = 0; k < N; k = k+1){
            r = r + y[i][k]*z[k][j];};
     x[i][j] = r;
    };
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
    for (j = jj; j < min(jj+B-1,N); j = j+1)
    {r = 0;
     for (k = kk; k < min(kk+B-1,N); k = k+1) {
            r = r + y[i][k]*z[k][j];};
     x[i][j] = x[i][j] + r;
    };
```





软件优化的效果


vpenta (nasa7)
gmty (nasa7)
tomcatv
btrix (nasa7)
mxm (nasa7)
spice
cholesky (nasa7)
compress
1
1.5
2
2.5
3
Performance Improvement
merged arrays
loop interchange
loop fusion
blocking

降低MissPenalty

- 关键字优先
- 读失效优先
- **Victim Cache**
- 写合并
- 多级**Cache**

通过关键字优先降低失效延迟

- 在Cache访问失效时，优先访问读访问需要的字
  - 例如，Cache块大小为64字节，分为8个8字节的双字，如果取数指令访问其中的第6个双字引起Cache失效，就按6、7、0、1、2、3、4、5的次序，而不是按0、1、2、3、4、5、6、7的次序访问内存
  - 关键字优先的实现只需要对访问地址进行简单变换。
  - 在失效的数据从内存取回之后，往Cache送的同时直接送回到寄存器也可以降低失效延迟。

通过读优先降低失效延迟

- 读Cache失效对指令流水线效率的影响比写失效大
  - 在处理器中一般都有写缓存（Write Buffer），写指令只要把要写的数据写到写缓存就可以提交，再由写缓存写到Cache或内存
  - 取数在数据读回来回前取数指令不能提交，与取数指令存在数据相关的后续指令要等待取数操作读回来的数据才能执行，可见读Cache失效容易堵塞指令流水线
- 在进行Cache失效处理时，优先处理读失效以减小堵塞
  - 读失效时要注意与Write Buffer的RAW相关，不能等写缓存为空，要动态检查写缓存
  - 读失效可能需要替换dirty块，不要把替换块写回到内存再读，可以把替换块写到写缓存，然后先读，后把写缓存的内容写回内存

通过写合并降低失效延迟

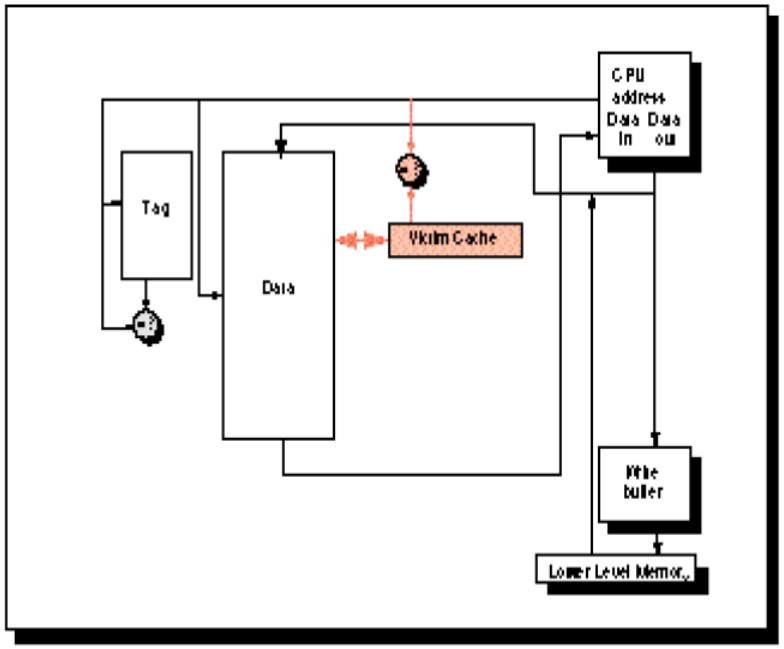
- 写缓存（Write Buffer）的作用
  - 处理器写到WB就算完成写操作，由WB写到下一级存储器
  - 注意一致性问题：处理器后续的读操作、外设的DMA操作等
- 通过把写缓存中对同一Cache块的写操作进行合并来提高写缓存使用率，减少处理器等待
  - 注意IO操作不能合并
  - Store Fill Buffer：连续的写操作拼满一个Cache块，写失效时不用到下一级存储器取数

块地址	v		v		v		v	
100	1	word0						
108	1	word1						
116	1	word2						
124	1	word3						

块地址	v		v		v		v	
100	1	word0	1	word1	1	word2	1	word3

通过Victim Cache降低失效延迟

- 在直接相联的Cache中避免冲突失效
- 增加缓存（Victim Cache）保存从Cache中替换出来的数据
  - Jouppi [1990]：4项victim cache可消除直接相联4KB Cache中20%-95%的冲突访问
- 在Alpha, HP等处理器中应用



通过二级Cache降低失效延迟

- 通过L2降低失效延迟

$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$$

$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$$

$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

- L2的失效率
  - 局部失效率（Local miss rate）— L2失效次数除以L2访问次数
  - 全局失效率（Global miss rate） — L2失效次数除以所有访存次数（$\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$）
  - 局部失效率较高（10%-20%），全局失效率很低（<1%）

降低HitTime

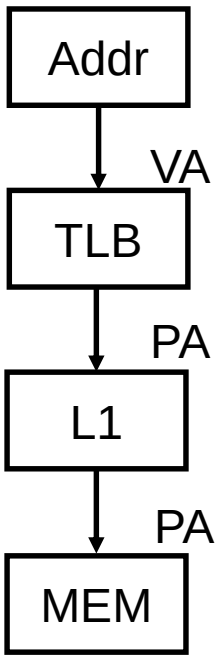
- **简化Cache设计**
- **并行访问Cache与TLB**
- **增加Cache访问流水级**

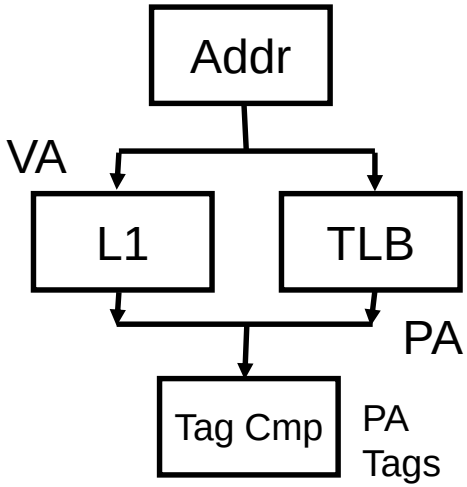
简化Cache设计

- 访问Cache常常是整个CPU的时钟关键路径
  - Cache越小，延迟越小
  - 直接相联的Cache延迟小
- 简化一级Cache设计需要统筹考虑
  - PIV数据Cache从PIII的16KB降低为8KB，而且只有定点可以访问，从而达到高主频
  - 但PIV的二级Cache只有6拍的访问延迟

避免地址转换延迟：虚地址Cache

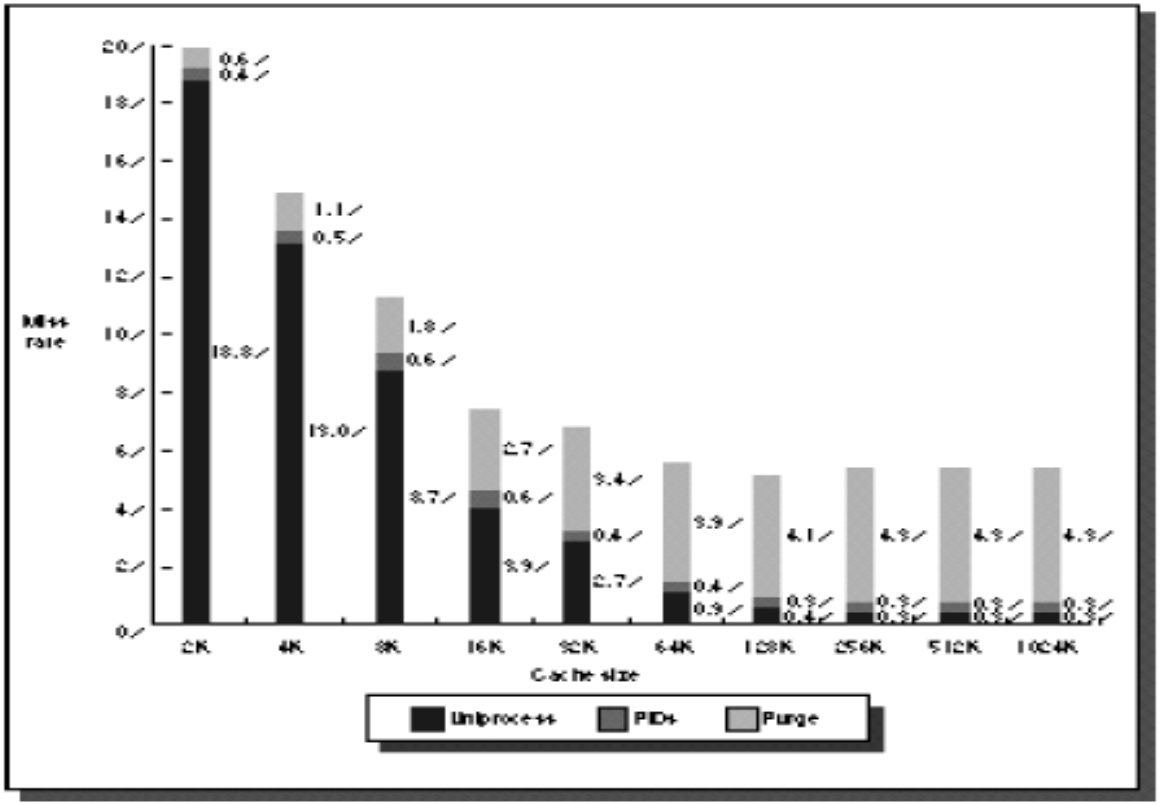
- 通过虚地址直接访问Cache减少虚实地址转换时间
  - 区分进程问题：不同进程的虚地址空间是一样的，需要在进程切换时候刷Cache以维护一致性，增加了冷失效
  - 别名（aliases）问题：操作系统有时候（如为了进程间共享）需要不同的虚地址对应同一物理地址





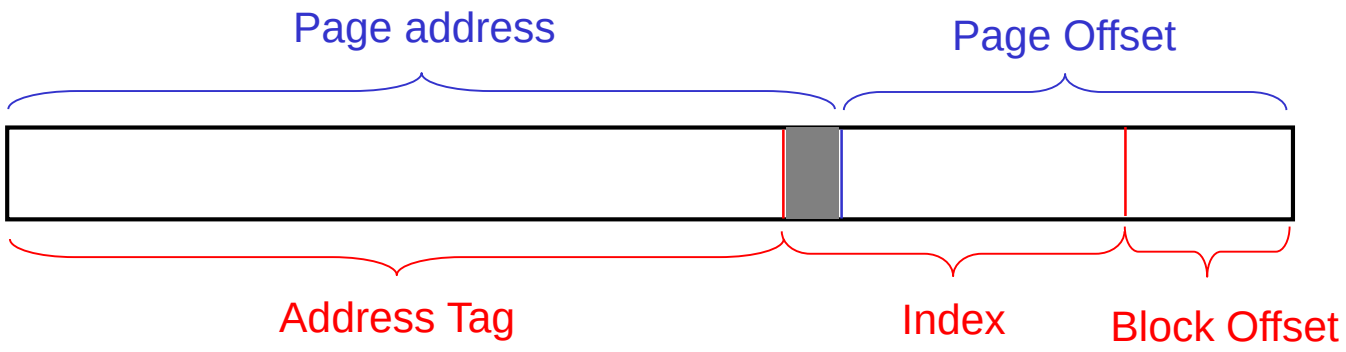
虚地址Cache中多进程对Cache命中率的影响

- 进程切换问题可以通过在TLB中增加进程号的方式来解决
  - 单进程、多进程切换时刷Cache、多进程用ID号来区分虚地址
  - Y轴表示Cache失效率；X轴表示Cache大小（2 KB - 1024 KB）



虚Index实Tag技术

- 在用**Index**从**cache**中读**Tag**的同时，进行虚实地址转换，可以使用物理地址做**tag**
- 如何保证虚实地址**Index**位的一致？
  - 增加页大小、增加相联度
  - 软件保证：页着色（**Page coloring**）



增加Cache访问流水级

- **Cache访问时间是处理器主频的决定因素之一**
  - **其他因素包括加法及bypass时间等**
- **把cache访问分成多拍以提高主频**
  - **MIPS：1拍**
  - **Alpha：2拍**
  - **增加流水节拍增加了load-use延迟**

提高Cache访问并行性

- 多访存部件
- 非阻塞Cache
- 硬件预取
- 软件预取

多访存部件

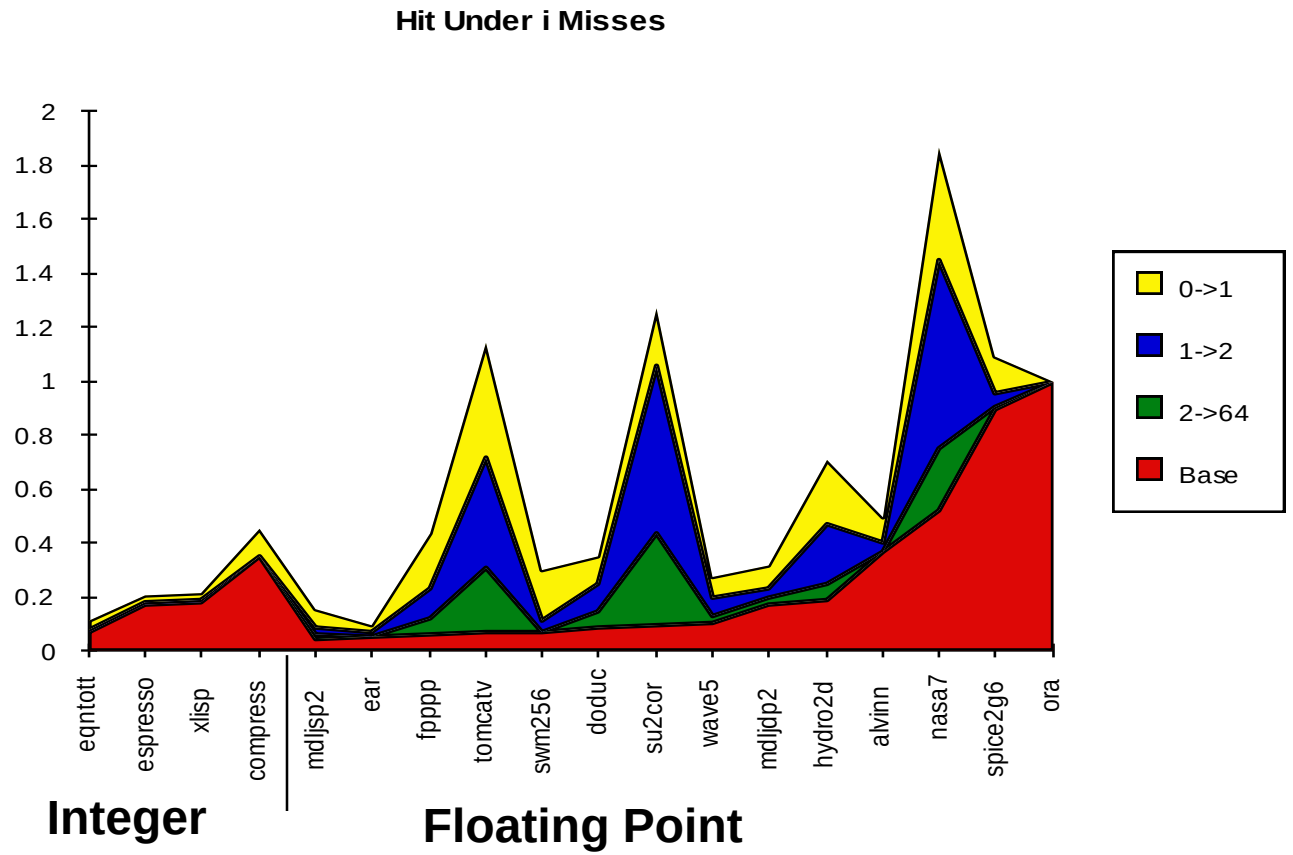
- 现代高性能处理器大都采用两个或更多访存部件
  - 如**Intel**的**IvyBridge**有两个读部件、一个写部件
  - **AMD**的**Bulldozer**有两个读/写部件
- 多个访存部件提高了访存的吞吐率
  - 降低了平均**Hittime**

非阻塞Caches

- 非阻塞（**Non-blocking/Lockup-free**）**Cache**在访问失效时允许后续的访问继续进行
  - **hit under miss**、**hit under multiple miss**、**miss under miss**
  - 前面的失效访问不影响后续访问，需要支持多个**outstanding**访问，需要类似与保留栈的访存队列机制
  - 在乱序执行的**CPU**中使用，显著增加**Cache**控制器复杂度
- 多个层次的非阻塞访问
  - **L1**、**L2**、内存控制器
  - 龙芯**2**号支持**24**个**L1**非阻塞访问，**8**个**L2**非阻塞访问

SPEC程序的非阻塞访存效果

- **8 KB直接相联数据Cache，块大小32字节，失效延迟16拍**
  - **浮点程序AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26**
  - **定点程序AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19**



通过硬件预取降低失效率

- 指令预取
  - **Alpha 21064 在Cache失效时取连续两块，多取的Cache块存放在流缓存（stream buffer）中，下一次Cache失效时先检查指令是否在流缓存中**
- 数据预取
  - **Jouppi [1990]：对4KB的Cache，1项流缓存可以命中25%的Cache失效访问， 4项流缓存可以命中43%的Cache失效访问**
  - **Palacharla & Kessler [1994]：指令和数据Cache各为四路组相联的64KB，对科学计算程序，8项流缓存可以命中 50%-70%的Cache失效访问**
- 现代处理器都实现复杂预取技术
  - 预取对访存带宽提出了更高的要求

通过软件预取降低失效率

- 软件预取都是数据预取
  - 预取到寄存器: (HP PA-RISC loads)
  - 预取到Cache: (MIPS IV, PowerPC, SPARC v. 9)
  - 预取指令不发生例外，预取到Cache的指令可以不等待数据返回
- 预取指令开销：占用指令槽
  - 多发射结构对预取指令占用指令槽不怎么敏感
- 龙芯2号对软件预取的支持
  - 目标寄存器为0号寄存器的取数指令不发生例外，不阻塞流水线

Cache优化小结

优化技术	失效延迟	命 中 率	命中时间	硬件复杂度
多级cache	+			2
关键字优先	+			2
读失效优先	+			1
合并写缓存	+			1
Victim cache	+	+		2
增加块大小	–	+		0
增加cache大小		+	–	1
增加相联度		+	–	1
伪相联、路猜测		+		2
编译优化		+		0
非阻塞cache	+			3
硬件预取	+	+		2i,3d
软件预取	+	+		3
小而简单的cache		–	+	0
Cache和TLB访问并行			+	2
流水cache访问			+	1

常见处理器的存储层次

MIPS R10000存储层次

- **L1**
  - **Dcache：32KB，2路，2拍延迟**
  - **Icache：32KB，2路**
- **L2**
  - **片外，2路伪相联（way prediction）**
  - **512KB-16MB**
- **1个load/store部件**
  - **16项address queue**
- **特点：predecode icache，L2伪相联**

HP PA-8x00存储层次

- **L1:**
  - **0.5~1.5MB**
  - **8500之前为片外（直接映射），之后为片内（4路组相联）**
- **L2**
  - **一直到8700，无**
- **2个load/store部件**
  - **28项address reorder队列**
- **特点：巨大的一级cache**

UltraSparc III存储层次

- **L1**
  - **Dcache：64KB，4路，2拍延迟**
  - **Icache：32KB，4路**
  - **Prefetch：2KB，4路，3拍延迟**
  - **Write cache：2KB，4路，1拍延迟**
- **L2**
  - **1/4/8MB，直接映射，12拍延迟**
  - **片上tag，片外data**
- **1个load/store部件，片上内存控制器**
- **特点：Sum-Addressed-Memory，片上L2 tag和内存控制器**

Alpha 21264存储层次

- **L1**
  - **Dcache：64KB，2路，3拍延迟，倍频访问（double-pumped）**
  - **Icache：64KB，2路，行预测和路预测**
- **L2**
  - **片外，1－16MB，最小12拍延迟**
- **2 load/store unit**
  - **32 load queue**
  - **32 store queue**
- **特点：icache行/路预测，dcache倍频访问**

Power4存储层次

- **L1**
  - **Dcache：32KB，2路，2读1写端口，4拍load延迟**
  - **Icache：64KB，直接映射**
- **L2**
  - **片上，1.5MB，8路组相联**
- **L3**
  - **32MB 8路组相联，片上tag，片外data**
- **2个load/store unit**
  - **32 load queue，32 store queue**
- **特点：片上cache多，数据通路宽，Cache块大（128-512B），并行处理能力强，SMP优化**

Pentium4存储层次

- **L1**
  - **Dcache：8KB，4路，2拍延迟**
  - **Icache：12K trace cache**
- **L2**
  - **256KB，8路，7拍load延迟**
  - **浮点直接访问2级Cache**
- **每拍可以处理一个读一个写**
  - **48项load queue，24项store queue**
- **特点：频率高，延迟较小，trace cache**

Itanium2存储层次

- **L1**
  - **Dcache：16KB，4路，两读两写端口，1拍load延迟**
  - **Icache：16KB，4路，1拍延迟**
- **L2**
  - **256KB，8路，4端口，最小5拍延迟**
- **L3**
  - **片上，3MB，12路，最小12拍延迟**
- **24项store queue**
- **特点：integer load only L1 dcache, pre-validated cache tag, advanced load**

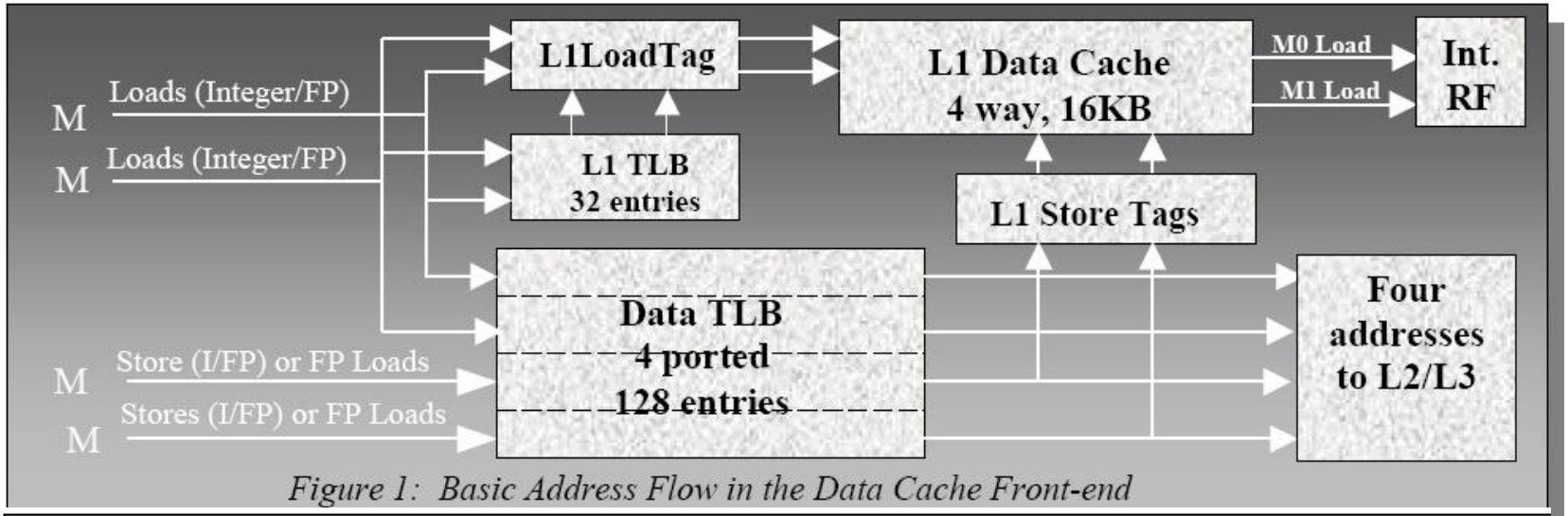


Figure 1: Basic Address Flow in the Data Cache Front-end

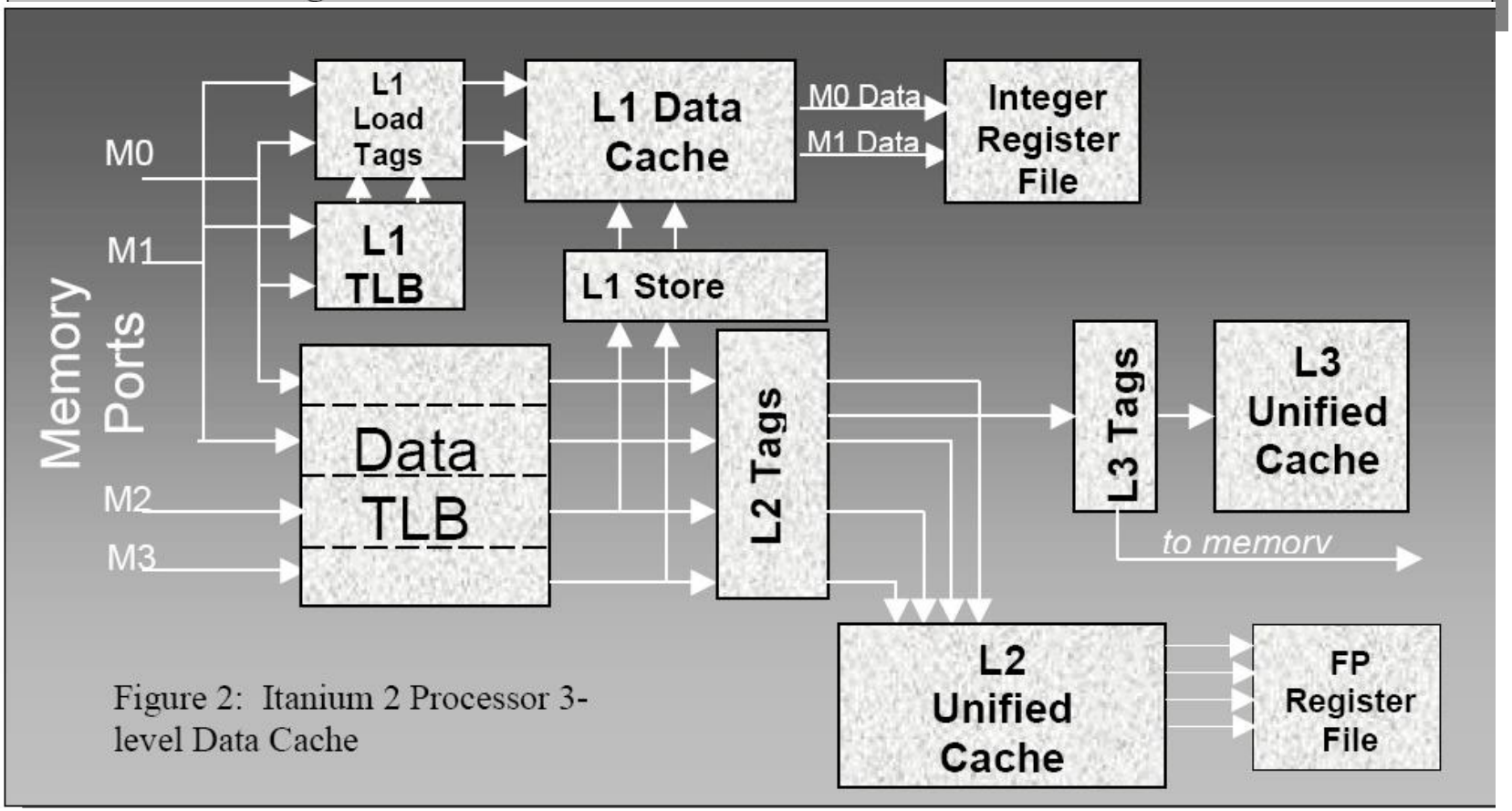


Figure 2: Itanium 2 Processor 3-level Data Cache

Athlon64存储层次

- **L1**
  - **Dcache：64KB，2路，3拍延迟**
  - **Icache：64KB，2路，3拍延迟**
- **L2**
  - **片上256KB/512KB/1MB，16路**
- **2个load/store部件**
- **特色：HT接口，片上DDR内存控制器**

常见处理器的数据通路（最近）

	Intel Sandybridge	AMD BullDozer	IBM Power7	GS464E
前端				
取指宽度	16 字节/时钟周期	32 字节/时钟周期	8 指令/时钟周期	8 指令/时钟周期
一级指令缓存	32KB，8 路，64B/行	64KB，2路，64B/行	32KB，4 路，128B/行	64KB，4 路，64B/行
指令TLB	128 项，4 路，L1 ITLB	72 项，全相联，L1 ITLB 512 项，4 路，L2 ITLB	64 项，2 路，L1 ITLB	64 项，全相联，L1 ITLB
分支预测	BTB（8K-16K?项） 间接目标队列（? 项） RAS（? 项） 循环检测	512 项，4 路 L1 BTB 5120 项，5 路 L2 BTB 512 项 间接目标队列 24 项 RAS 循环检测	8K 项本地 BHT 队列， 16K 项全局BHT 队列， 8K 项 全局 sel 队列 128 项间接目标队列 16 项 RAS	8K项本地 BHT 队列 8K项全局BHT 队列 8K项 全局 sel 队列 128项间接目标队列 16项 RAS 循环检测
乱序执行				
Reorder缓存	168 项	128 项	120 项	128 项
发射队列	54-项 统一	60-项 浮点(共享) 40-项 定点，访存	48-项 标准	16-项 浮点； 16-项 定点； 32-项 访存
寄存器重命名	160 定点； 144 浮点	96 定点； 160 浮点(双核共享)	80 定点,浮点； 56 CR； 40 XER， 24 Link&Count	128 定点； 128 浮点/向量； 16 Acc； 32 DSPCtrl； 32 FCR
执行单元				
执行单元个数	ALU/LEA/Shift/128位 MUL/128位 Shift/256位FMUL/256位Blend + ALU/LEA/Shift/128位 ALU /128bit Shuffle/256位 FADD + ALU/Shift/Branch/ 128位 ALU/128bit Shuffle/256位 Shuffle/256位 Blend	ALU/IMUL/Branch + ALU/IDIV/Count + 128位FMAC/128位 IMAC + 128位FMAC/128位 XBAR + 128位 MMX + 128位 MMX/128位 FSTO	2 定点 + 2 浮点/向量 + 1 转移 + 1 CR	2 定点/转移/DSP + 2 浮点/向量
向量部件宽度	256位	128位	128位	256位

常见处理器的数据通路（最近）

	Intel Sandybridge	AMD BullDozer	IBM Power7	GS464E
访存执行单元	2 取+ 1 存	2 取/存	2 取/存	2 取/存
Load/Store队列	64-项 Load 队列, 36-项 Store队列	40-项 Load 队列, 24-项 Store 队列	32-项 Load队列, 32-项 Store 队列	64-项 取/存 队列
Load/Store 宽度	128 位	128 位	256 位 load, 128 位 Store	256 位
TLB	100项全相联，L1 DTLB, 512项4路，L2 TLB	32项全相联，L1 DTLB, 1024项8路，L2 TLB	64项全相联，L1 DTLB, 512项4路，L2 TLB	256-项全相联，TLB 每项两页
一级数据缓存	32KB，8 路，64B/行	16KB，4路，64B/行	32KB，8 路，128B/行	64KB，4 路，64B/行
二级缓存	每核256KB，8路，64B/行	双核共享2MB，16 路	每核256KB，8路，28B/行	每核256KB，16路,64B/行
三级缓存	8个核20 MB	4个核8 MB	8个核32 MB	4个核4 MB～8MB
一级数据缓存同时处理缺失请求数	10	?	8	Unified 16
二级缓存同时处理缺失请求数	16	23	24	
一级 Load-to-use 延迟	定点4时钟周期, 浮点/向量5时钟周期	4时钟周期	定点2时钟周期, 浮点/向量3 时钟周期	定点3时钟周期, 浮点/向量5 时钟周期
二级 Load-to-use 延迟	12 时钟周期	18-20 时钟周期	8	17 时钟周期

作 业