

2.2 内存技术与优化

Ch2-2 如何提高内存层次结构的性能

□ 性能指标 > 延迟是缓存关注的问题

带宽是多处理器和I/O关注的问题 > 访问时间 - 读取请求发出到所需字到达之间的时间 > 周期时间 - 对内存的不相关请求之间的最短时间

□ 静态随机存取存储器 (SRAM) 延迟低, 用于缓存
□ 将动态随机存取存储器 (DRAM) 芯片组织成多个存储体以实现高带宽, 用于主存

2

内存技术

OSRAM

- > 保留位所需功耗低
- > 每位需要6个晶体管

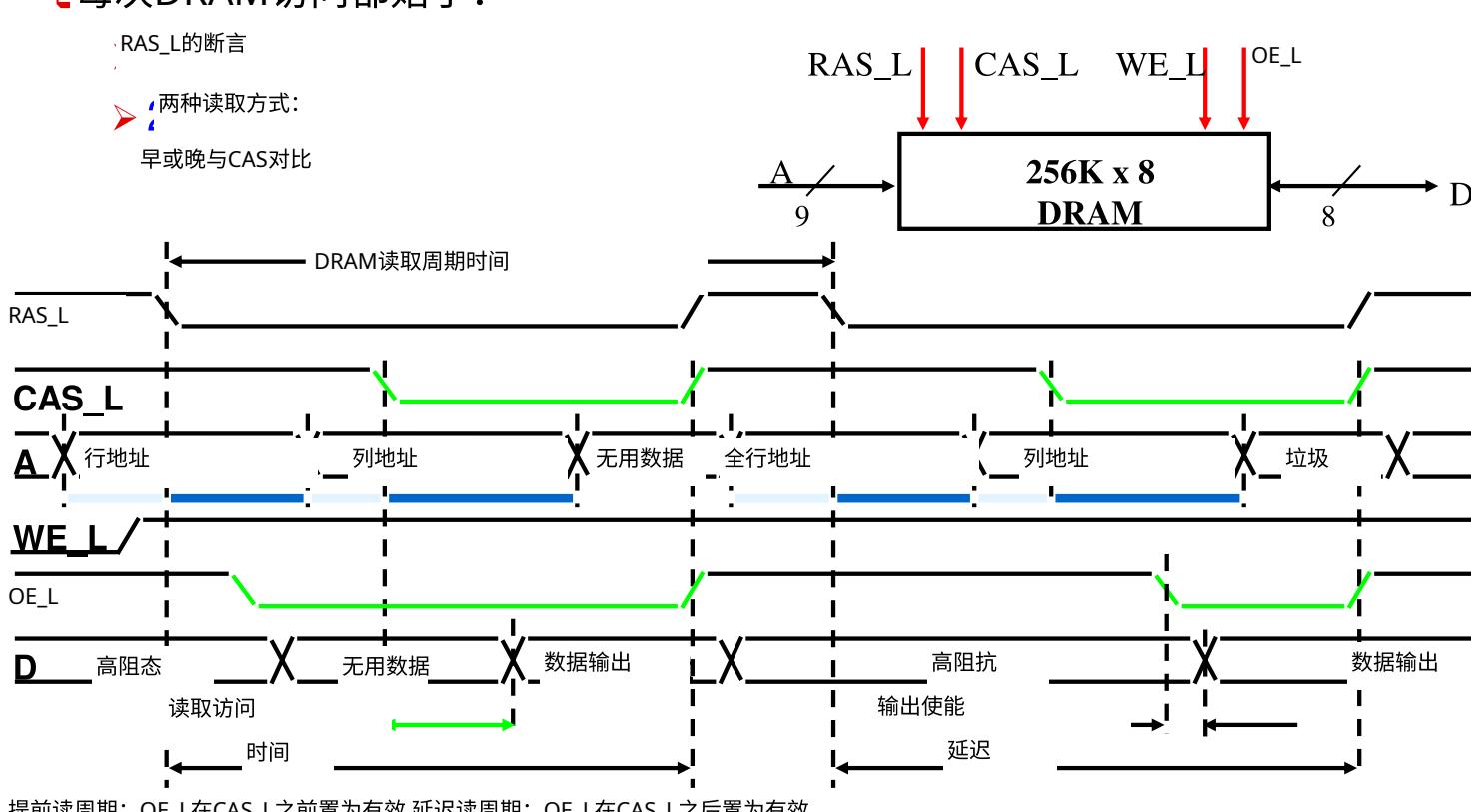
□ 动态随机存取存储器

- > 读取后必须重写
- > 还必须定期刷新
 - 每 ~ 8 ms (大约 5% 的时间)
 - 每行可以同时刷新
- > 一个晶体管/位
- > 地址线采用复用方式:
 - 地址的上半部分: 行访问选通信号 (RAS)
 - 地址的下半部分: 列地址选通 (CAS)

3

动态随机存取存储器 (DRAM) 读取时序

□ 每次DRAM访问都始于:



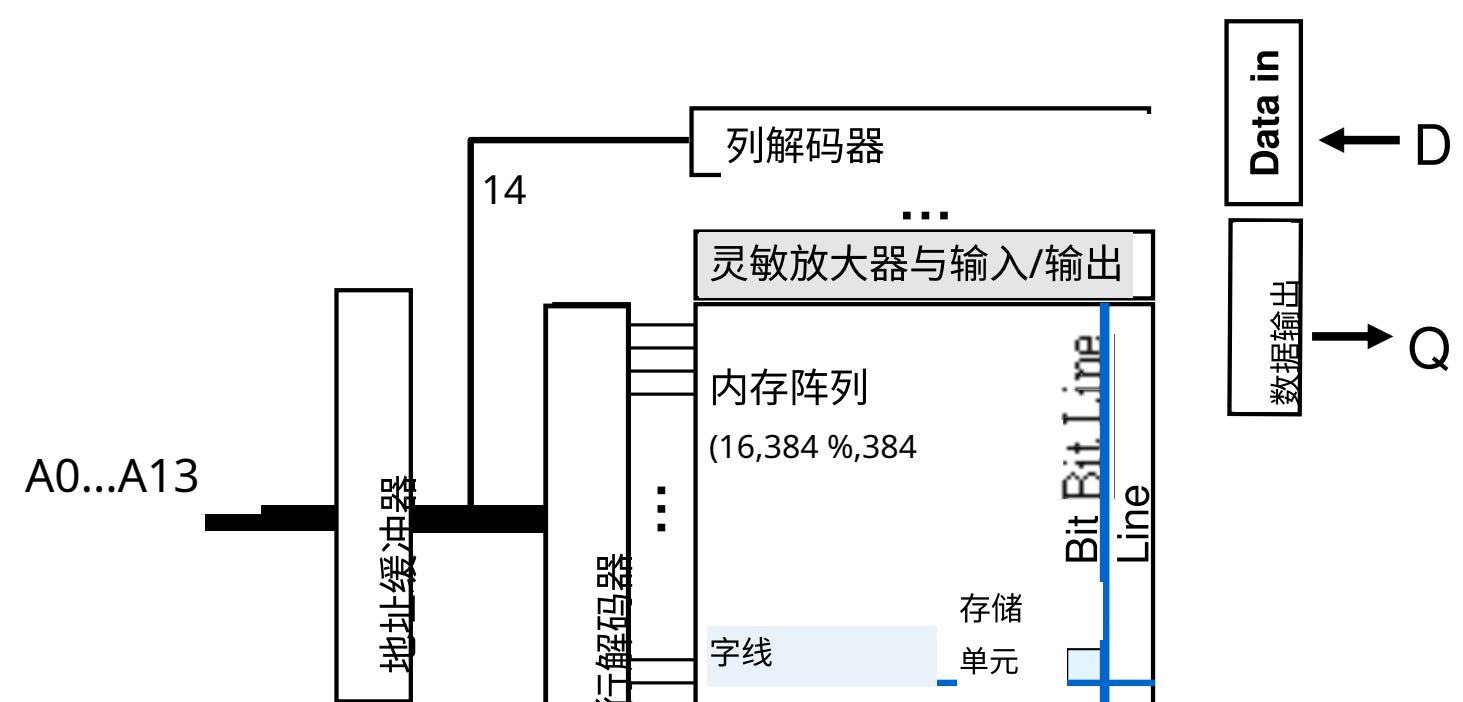
5

各代DRAM的快速和慢速时间。

推出.....的年份	芯片尺寸	行地址选通脉冲 (RAS)		Column 选通脉冲数 (CAS) 据传输时间	周期时间
		最慢 DRAM	最快 DRAM		
1980	64千比特	180 ns	150 ns	75 ns	250 ns
1983	256千比特	150 ns	120 ns	50 ns	220 ns
1986	1兆比特	120 ns	100 ns	25 ns	190 ns
1989	4兆比特	100 ns	80 ns	20 ns	165 ns
1992	16兆比特	80 ns	60 ns	15 ns	120 ns
1996	64兆比特	70 ns	50 ns	12 ns	110 ns
1998	128兆比特	70 ns	50 ns	10 ns	100 ns
2000	256兆比特	65 ns	45 ns	7 ns	90 ns
2002	512兆比特	60 ns	40 ns	5 ns	80 ns

7

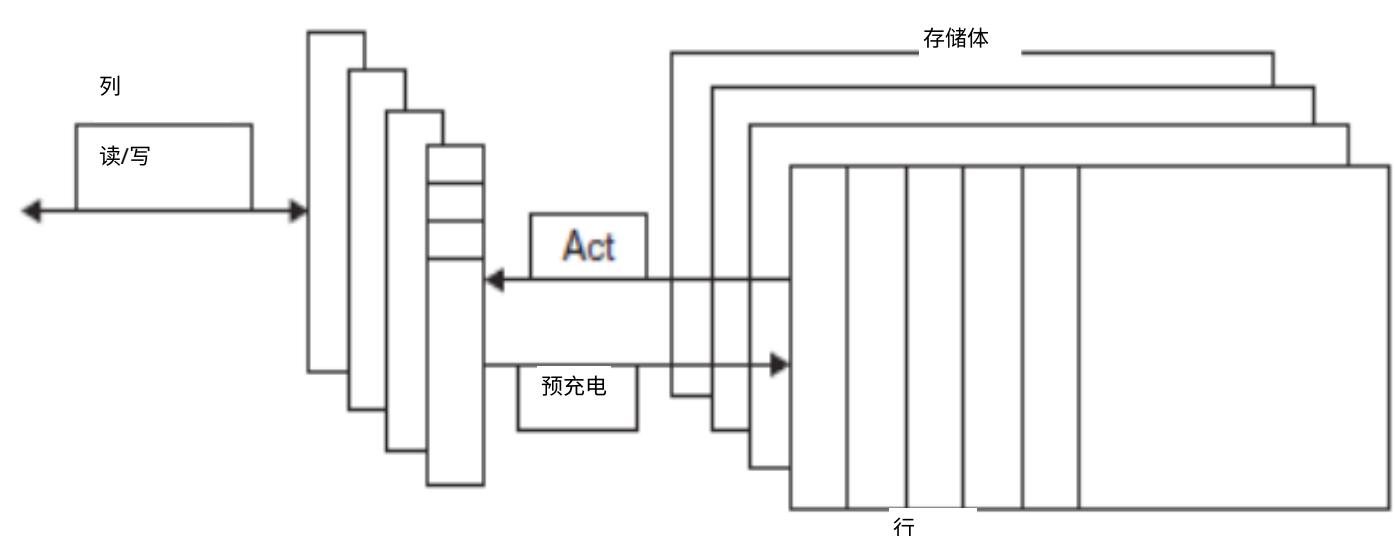
DRAM逻辑组织 (256 Mbit, 原“64 Mbit”有误)



□ 每次行地址选通/列地址选通的位数平方根

4

动态随机存取存储器 (DRAM) 的内部组织



6

内存优化

生产年份	芯片尺寸	动态随机存取存储器类型	最佳情况访问时间 (无需预充电)		需要预充电	
			RAS时间 (纳秒)	CAS时间 (纳秒)	总计 (纳秒)	总计 (纳秒)
2000	256兆位	DDR1	21	21	42	63
2002	512兆位	DDR1	15	15	30	45
2004	1吉比特	DDR2	15	15	30	45
2006	2吉比特	DDR2	10	10	20	30
2010	4吉比特	DDR3	13	13	26	39
2016	8吉比特	DDR4	13	13	26	39

8

内存技术

□ 阿姆达尔:

- > 内存容量应与处理器速度呈线性增长
- > 遗憾的是，内存容量和速度未能跟上处理器的发展

□ 一些优化措施:

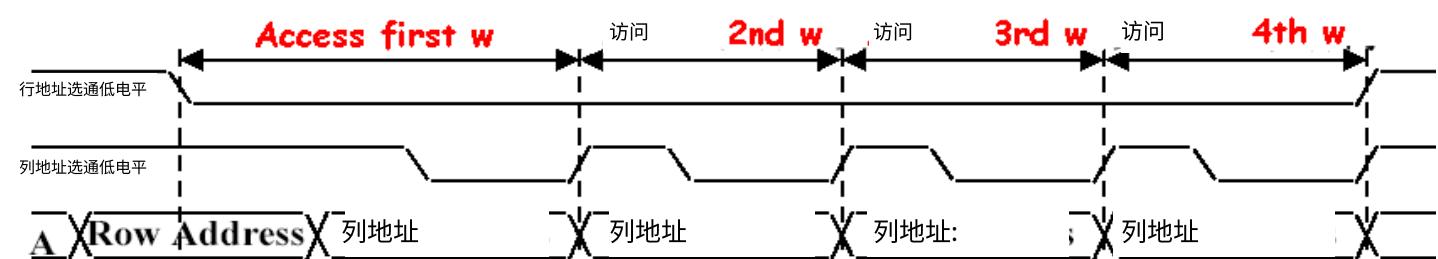
- > 对同一行的多次访问
- P 同步动态随机存取存储器 (SDRAM) 异步访问
握手过程很耗时 - 为DRAM接口添加时钟 - 采用关键字优先的突发模式
 - 更宽的接口一次获取更多字
 - > 双倍数据速率 (DDR) 工作频率提高
 - > 每个DRAM设备上有多个存储体

1st 提高动态随机存取存储器 (DRAM) 性能快速页模式动态随机存取存储器 (FPM)

□ 允许在无需额外行访问时间的情况下对行缓冲区 (页) 进行重复访问的定时信号

C] 这种缓冲区是自然存在的，因为每个阵列每次访问都会缓冲1024到2048位。

□ 页：同一行上的所有位（空间局部性）> 无需等待字线充电
> 用新的列地址切换列地址选通 (CAS)



10

3rd 提升DRAM性能DDR - 双倍数据速率

□ 在DRAM时钟信号的上升沿和下降沿，DRAM提高带宽的创新方法是传输数据，> 从而使峰值数据速率翻倍。

标准	时钟频率 (兆赫兹)	每秒兆次传输	DRAM名称	兆字节/秒 /DIMM	DIMM名称
2.5V	DDR	133	DDR266	2128	PC2100
	DDR	150	DDR300	2400	PC2400
	DDR	200	DDR400	3200	PC3200
1.8v	DDR2	266	DDR2-533	4264	PC4300
	DDR2	333	DDR2-667	5336	PC5300
	DDR2	400	DDR2-800	6400	PC6400
1.5v	DDR3	533	DDR3-1066	8528	PC8500
	DDR3	666	DDR3-1333	10,664	PC10700
	DDR3	800	DDR3-1600	12,800	PC12800

12

2nd 提高DRAM性能 同步动态随机存取存储器

□ 传统DRAM与内存控制器采用异步接口，因此每次传输都需要额外开销来与控制器同步。

□ 解决方案是向DRAM接口添加时钟信号，这样重复传输就不会产生该开销。

- > 数据以突发方式输出，每个元素都由时钟控制



11

DDR - 双倍数据速率

□ DDR

> DDR2

- 更低功耗 (2.5 V → 1.8 V)
- 更高时钟频率 (266 MHz, 333 MHz, 400 MHz) > DDR3 - 1.5 V # 800 MHz > DDR4 - 1 - 1.2 V - 1333 MHz

□ GDDR5是基于DDR3的图形内存

4rd 提升动态随机存取存储器 (DRAM) 性能

新型动态随机存取存储器接口：兰巴斯动态随机存取存储器 (RDRAM)

□ 一种同步动态随机存取存储器，由兰巴斯公司设计。□ 每个芯片都有交错式内存和高速接口。□ 基于协议的随机存取存储器，采用窄 (16位) 总线> 高时钟频率 (400兆赫兹)，但延迟较长 > 流水线操作

□ 多个数组，数据在时钟的两个边沿传输

第一代 Rambus 接口舍弃了行地址选通 (RAS) /列地址选通 (CAS)，取而代之另一种总线，该总线允许在发送地址和返回数据之间通过总线进行其他访问。它通常被称为 RDRAM。

□ 第二代 Rambus 接口包括独立的行命令总线和列命令总线，而非传统的复用方式；并且芯片上有更复杂的控制器。由于数据总线、行总线和列总线相互独立，因此可以同时执行三个事务。称为直接 RDRAM 或 DRDRAM

13

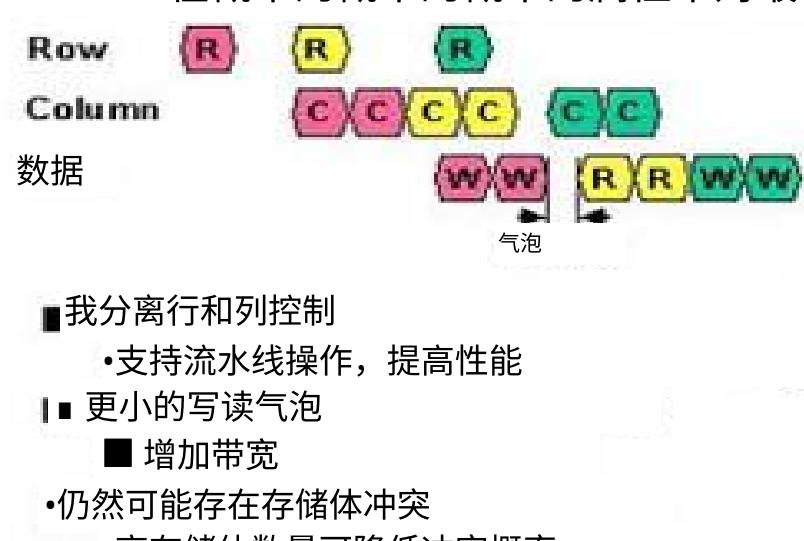
14

RDRAM时序

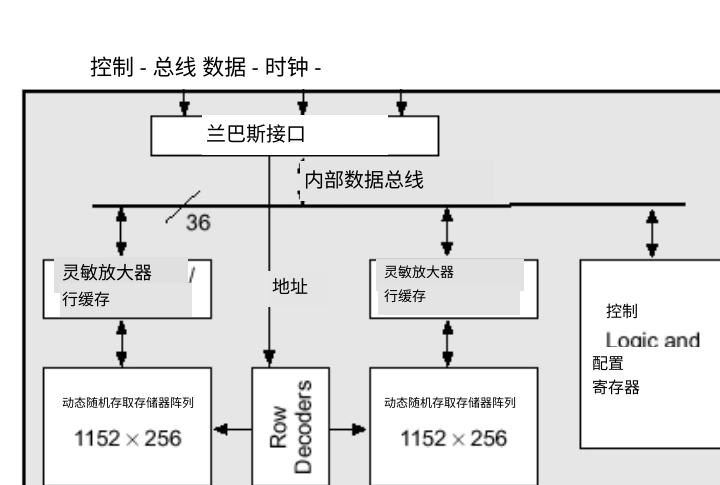
Rambus (RDRAM)

直接RDRAM协议 (32字节传输)

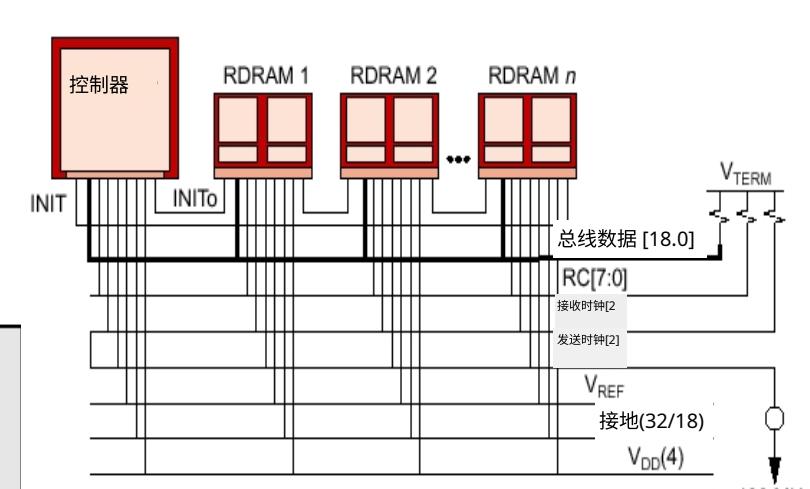
在概率的概率的属性中对最终可开发性的表征



15



兰巴斯银行



16

比较Rambus和DDR SDRAM

由于大多数计算机使用双列直插式内存模块 (DIMM) 封装的内存，其宽度通常至少为64位，因此与仅比较DRAM芯片时的预期相比，DIMM内存带宽更接近Rambus所提供的带宽。

□ 请注意，缓存性能部分取决于访问第一个字节的延迟，部分取决于传输块中其余字节的带宽。

> 尽管这些创新有助于解决后一种情况，但对延迟问题并无帮助。

阿姆达尔定律提醒我们，在加速问题的一部分而忽略另一部分时存在局限性。

17

总结

□ 内存组织

- > 更宽的内存
- > 简单交错式内存
- > 独立内存体
- > 避免内存库冲突

□ 内存芯片

- > 快速页面模式动态随机存取存储器
- > 同步动态随机存取存储器
- > 双倍数据速率

> 总线式动态随机存取存储器

> 总线式动态随机存取存储器

内存优化

□ 降低SDRAM的功耗：

> 更低的电压

> 低功耗模式（忽略时钟，继续刷新）

□ 图形内存：

> 与DDR3相比，每个DRAM可实现2 - 5倍的带宽 -

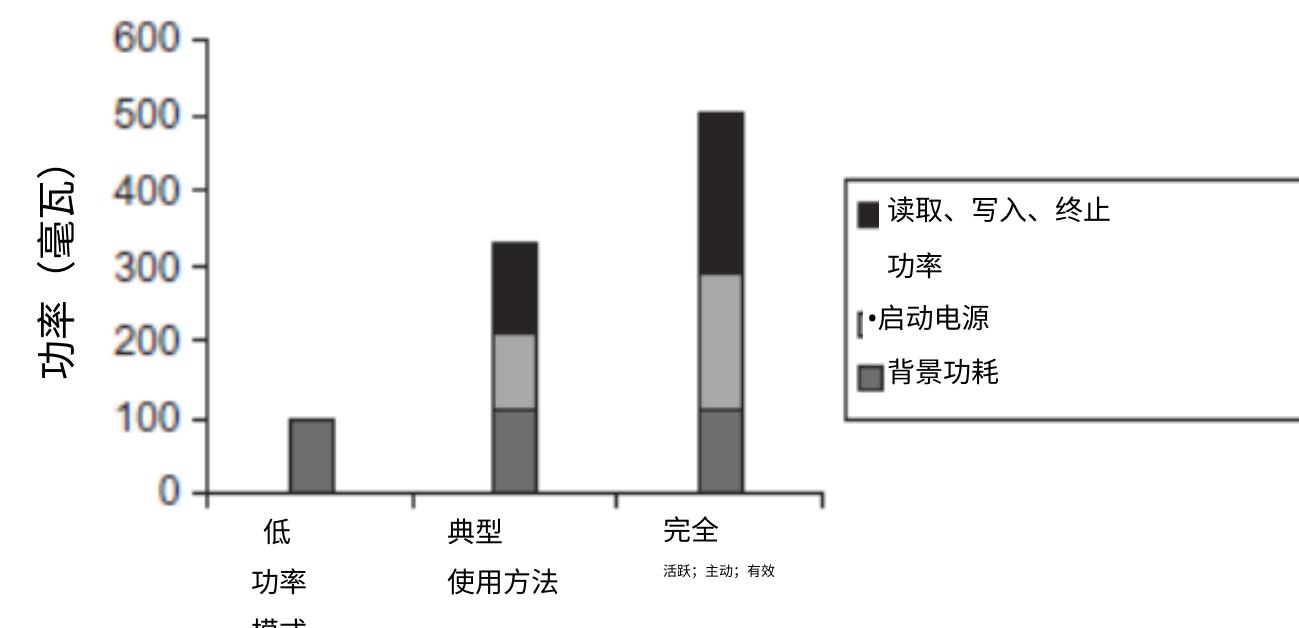
更宽的接口（32位对16位）

• 更高的时钟频率

> 这是可能的，因为它们是通过焊接连接的，而不是采用插槽式DIMM模块

18

内存功耗

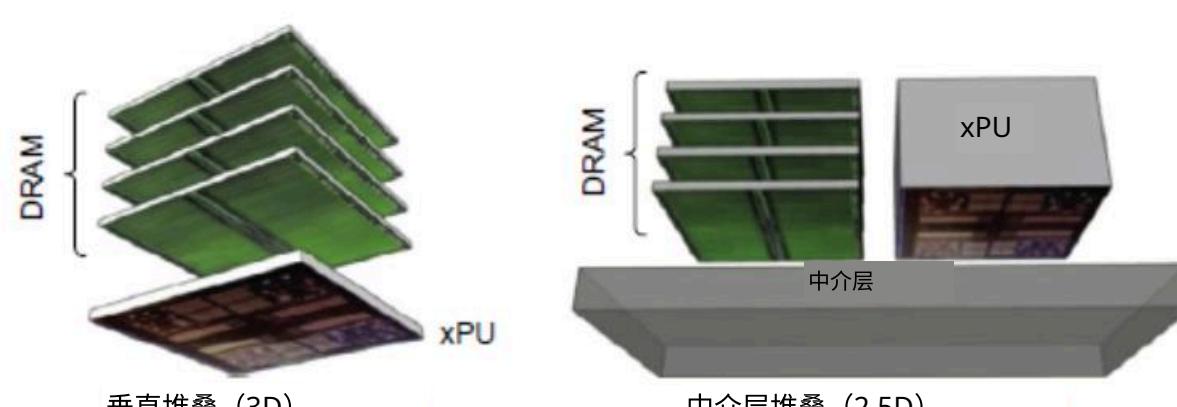


19

堆叠/嵌入式动态随机存取存储器

□ 与处理器封装在同一封装中的堆叠式动态随机存取存储器

> 高带宽内存 (HBM)



21

闪存 (NAND Flash Memory)

□ 在覆盖写入前必须（按块）擦除

□ 非易失性，可实现零功耗

□ 写入周期次数有限（约100,000次）

□ 每吉字节2美元，相比之下，同步动态随机存取存储器 (SDRAM) 每吉字节20 - 40美元，磁盘每吉字节0.09美元

□ 相变/忆阻器内存 写入性能可能提升10倍，读取性能可能提升2倍

20

闪存

□ 电可擦可编程只读存储器类型

□ 类型：NAND（密度更高）和NOR（速度更快）

□ NAND闪存：

> 读取是顺序进行的，读取整页（0.5到4千字节）

> 25用于第一个字节，40MiB/s用于后续字节

> SDRAM：第一个字节40纳秒，后续字节传输速度为4.8GB/秒

> 2KiB传输：与SDRAM相比，75uS，150X更慢

> 比磁盘快300到500X倍

22

内存可靠性

□ 内存易受宇宙射线影响

□ 软错误：动态错误

> 通过纠错码 (ECC) 检测并修复

□ 硬错误：永久性错误

> 使用备用行替换有缺陷的行

□ 芯片杀手：一种类似RAID的错误恢复技术

23

24

如何提高缓存性能?

$$\text{平均访存时间} = \text{命中时间} + \text{失效率} \times \text{失效开销}$$

□减少命中时间(4)

> 小型简单的一级缓存、路预测 > 避免地址

转换、踪迹缓存

□增加带宽(3)

> 流水线缓存、多体缓存、非阻塞缓存

□降低缺失代价(5)

> 多级缓存，读缺失优先于写操作

> 关键字优先、合并写缓冲区和牺牲缓存

□降低缺失率(4)

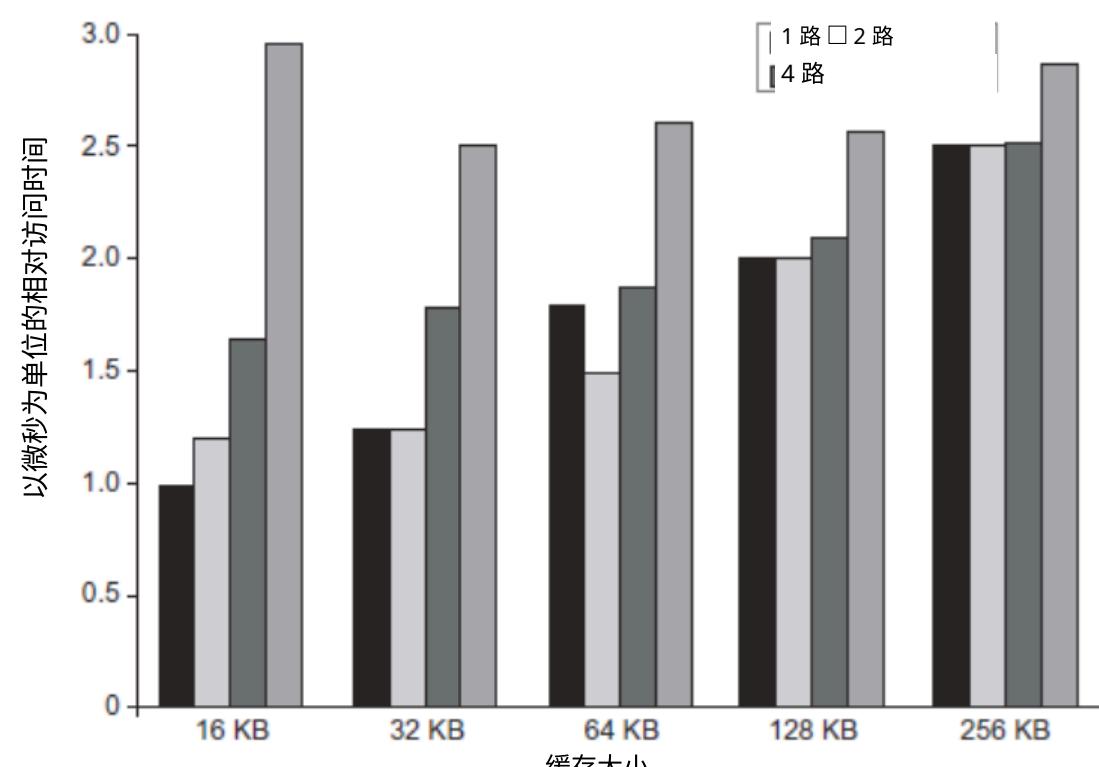
> 更大的块大小、更大的缓存容量、更高的关联性

> 编译器优化

□通过并行化降低缺失代价或缺失率 (1) > 硬件或编译器预取

25

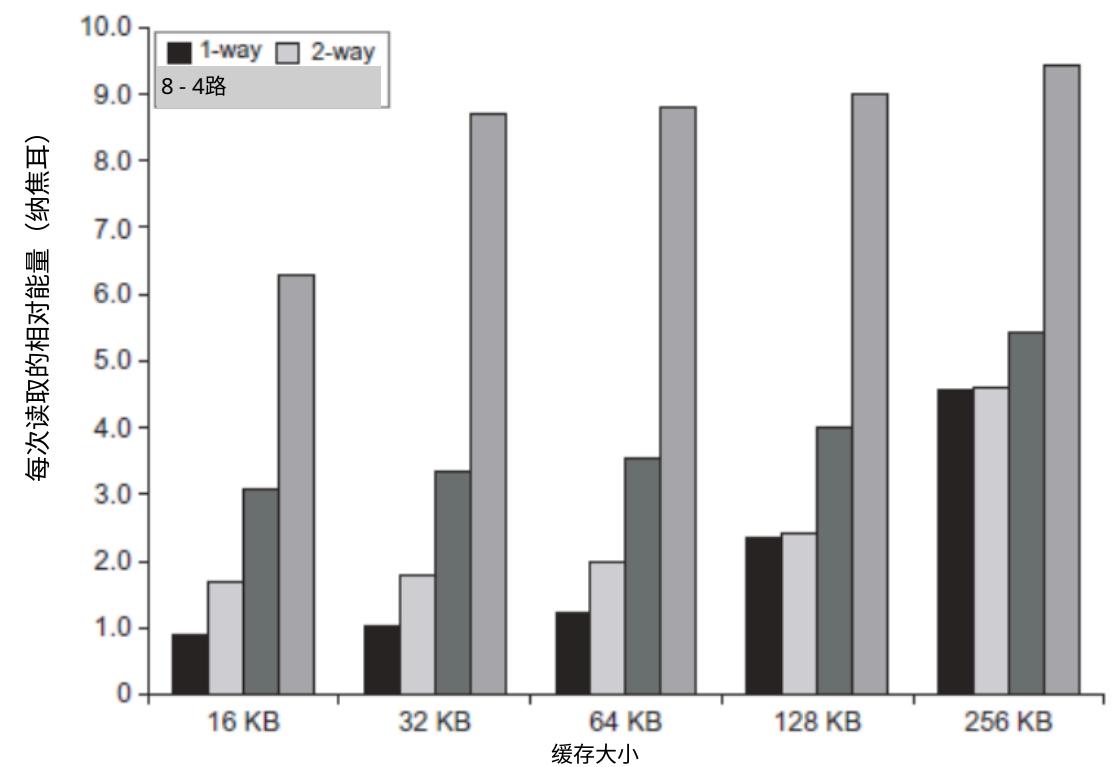
L1 大小和关联性



访问时间与大小和关联性的关系

26

L1 大小和关联性



每次读取的能量与大小和关联性的关系

27

2nd 命中时间缩减技术：路预测

□为了提高命中时间，预测预设置多路复用器的路 > 错误预测会导致更长的命中时间

> 预测准确率

= 90% 对于两路

• 对于四路为50%

• 指令缓存 (I-cache) 比数据缓存 (D-cache) 具有更高的准确性
> 90年代中期首次在MIPS R10000上使用

> 用于ARM Cortex - A8

□也扩展到预测块

> “路选择”

> 增加了错误预测的代价

28

2nd 命中时间减少技术：路预测

□路预测 (奔腾4)

缓存中保留额外的位，以预测下一次缓存访问的组内的路或块。

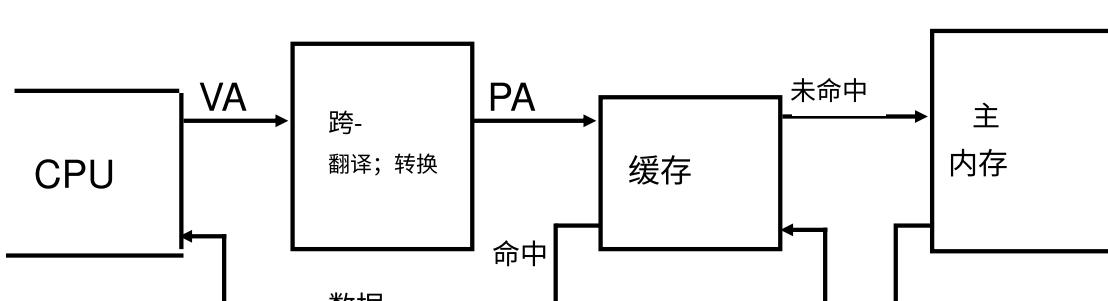
> 如果预测器预测正确，指令缓存延迟为1个时钟周期。

> 如果预测错误，它会尝试另一个块，更改路预测器，并且会额外增加1个时钟周期的延迟。

> 使用SPEC95进行的模拟表明，集预测准确率超过 85%，因此路径预测在超过85%的指令提取中节省了流水线阶段。

29

3rd 命中时间缩减技术： 在缓存索引期间避免地址转换



□页表是内存中的大型数据结构

每次加载、存储或取指令都需要两次内存访问！！！

□虚拟寻址缓存？

> 同义词问题

是否缓存地址转换结果？

30

转换后备缓冲器

一种加快地址转换的方法是使用一个最近使用过的页表条目的特殊缓存——它有很多名称，但最常用的是转换后备缓冲器 (TLB)

虚拟地址	物理地址	脏位	参考	有效	访问

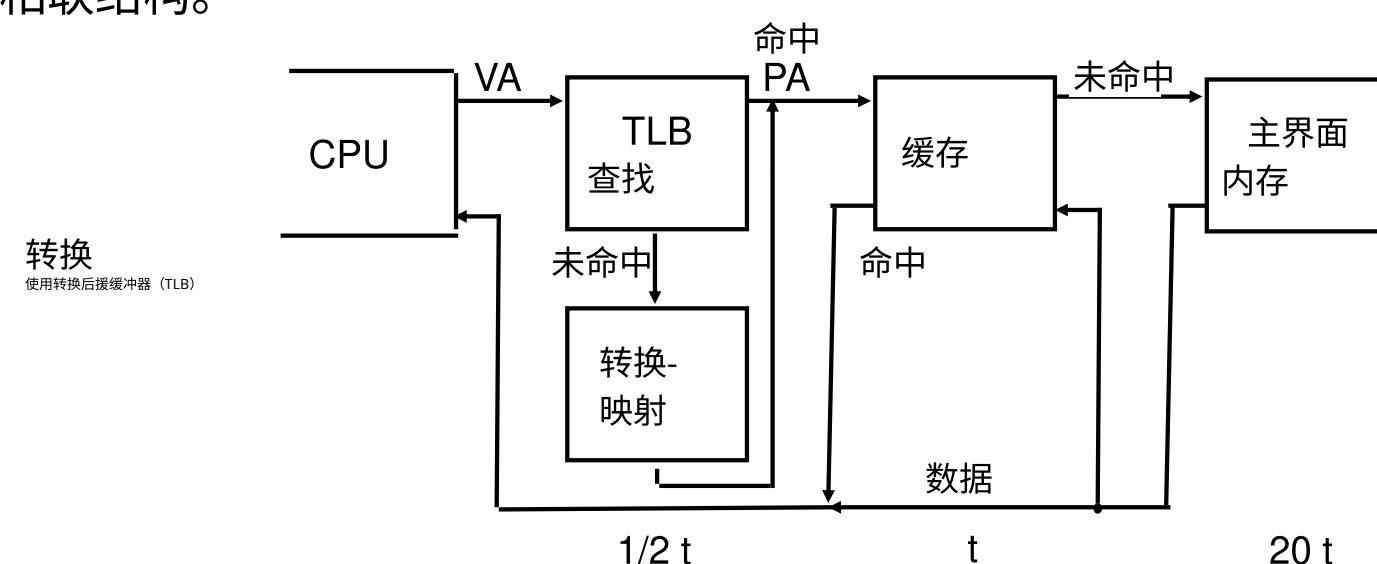
实际上只是页表映射的一个缓存

TLB访问时间与缓存访问时间相当（远小于主存访问时间）

转换后备缓冲器

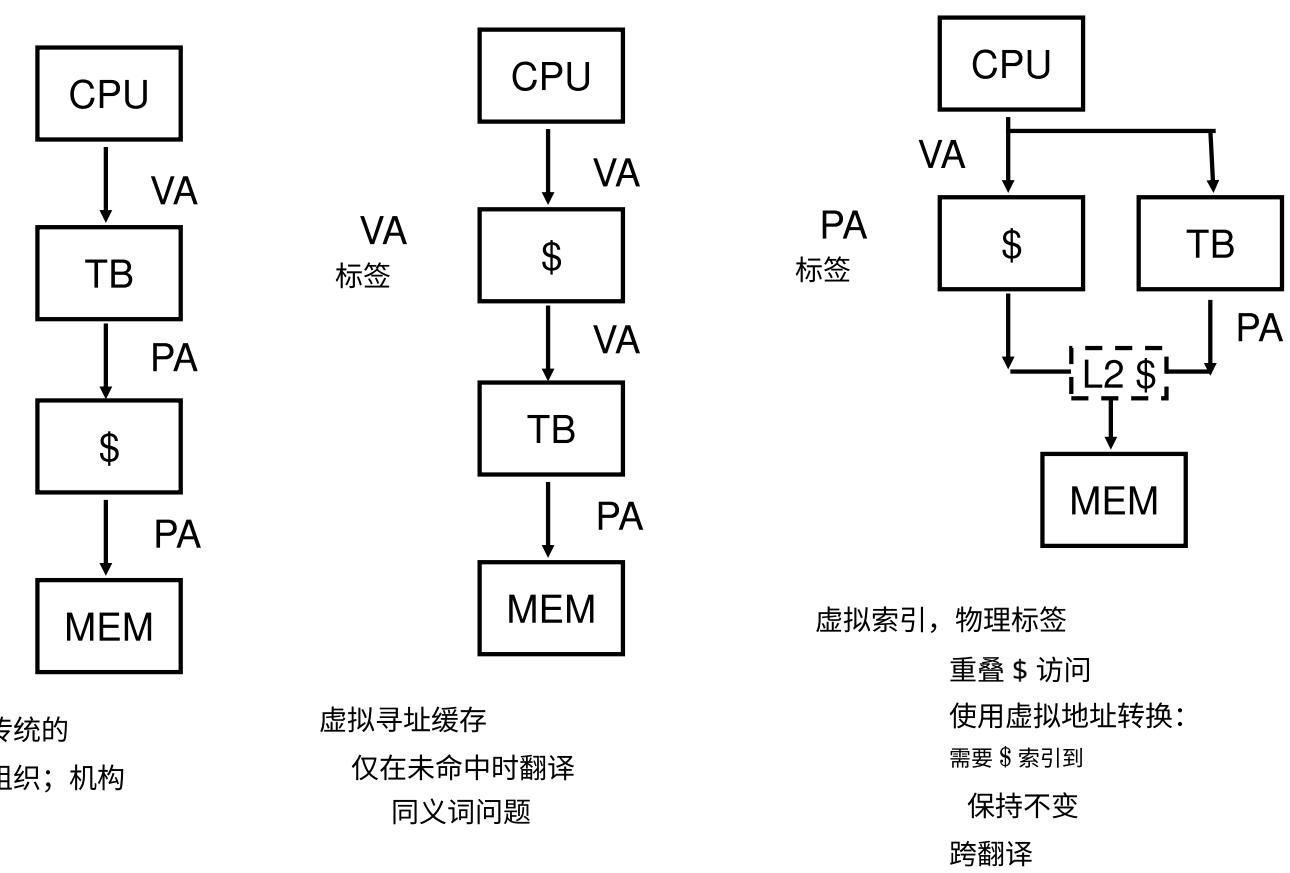
与其他任何缓存一样，TLB可以组织为全相联、组相联或直接映射

TLB通常很小，即使在高端机器上，条目通常也不超过128 - 256个。这允许在这些机器上进行全相联查找。大多数中端机器使用小型n路组相联结构。



33

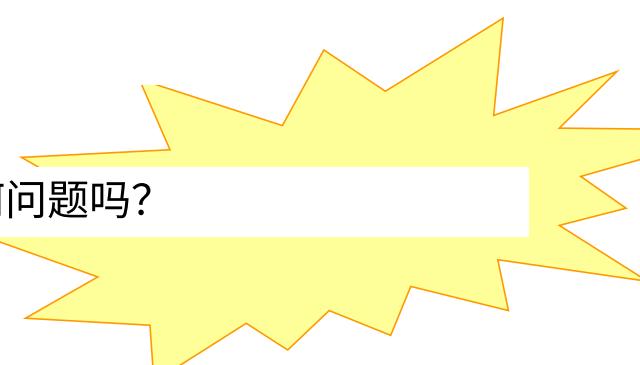
通过避免地址转换实现快速命中



34

虚拟地址缓存

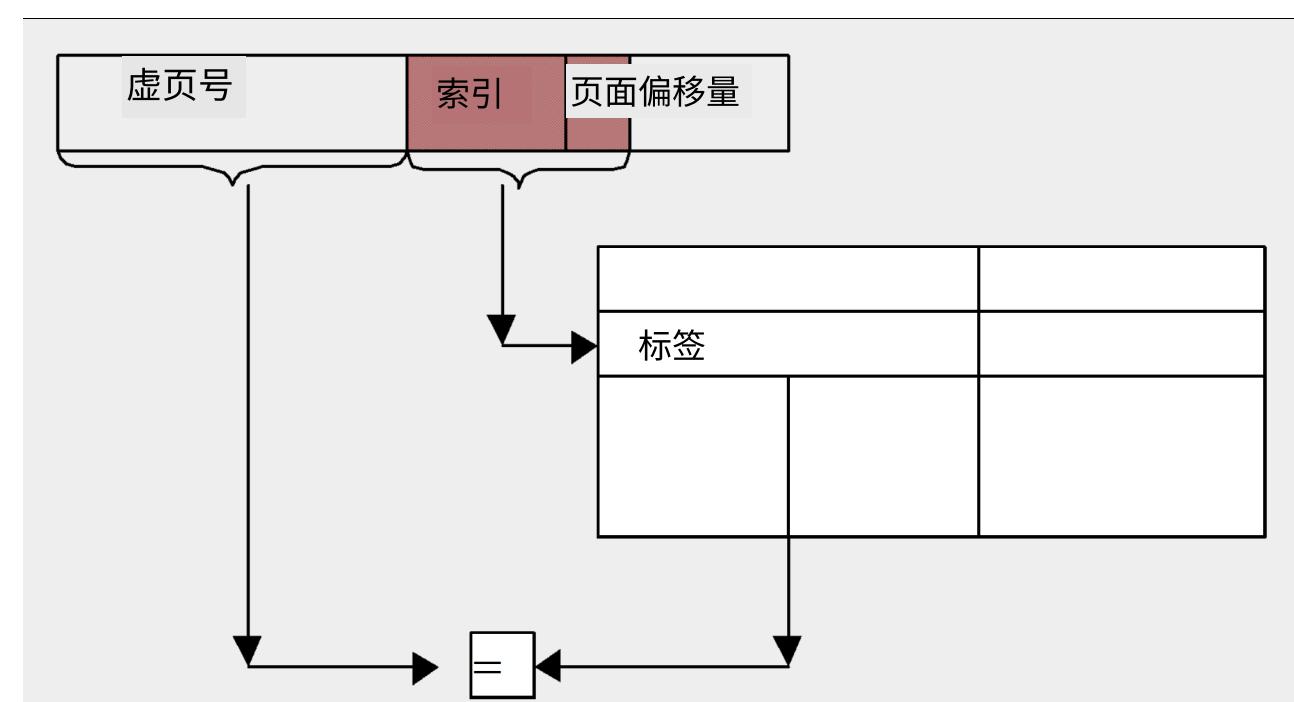
- 将虚拟地址发送到缓存？称为虚拟-有任何问题吗？
缓存或
仅虚拟缓存（相对于物理缓存）



- 每次逻辑上切换进程时，必须刷新缓存；
否则会出现误命中
> 代价是刷新时间 + 空缓存导致的“强制性”缺失
> 添加既能标识进程又能标识地址的进程标识符标签
在进程中：如果进程错误则无法命中

35

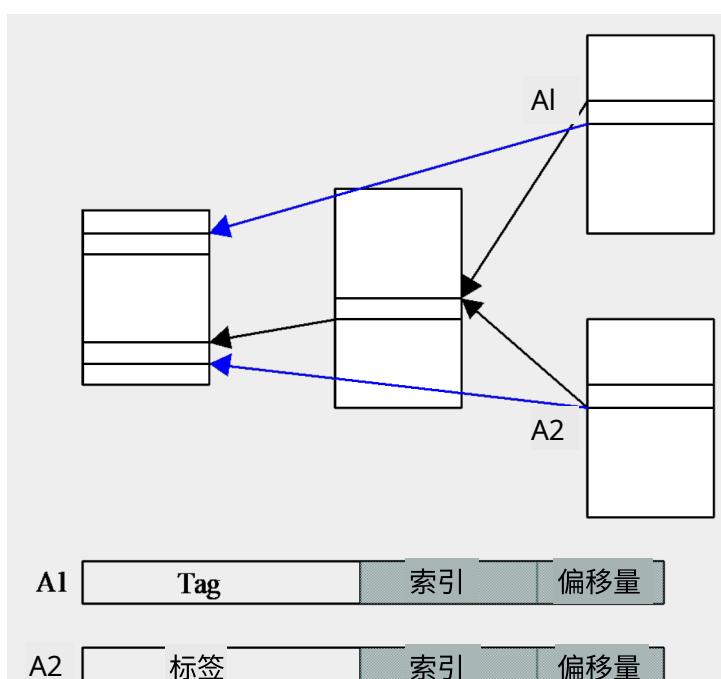
虚拟缓存



36

处理别名

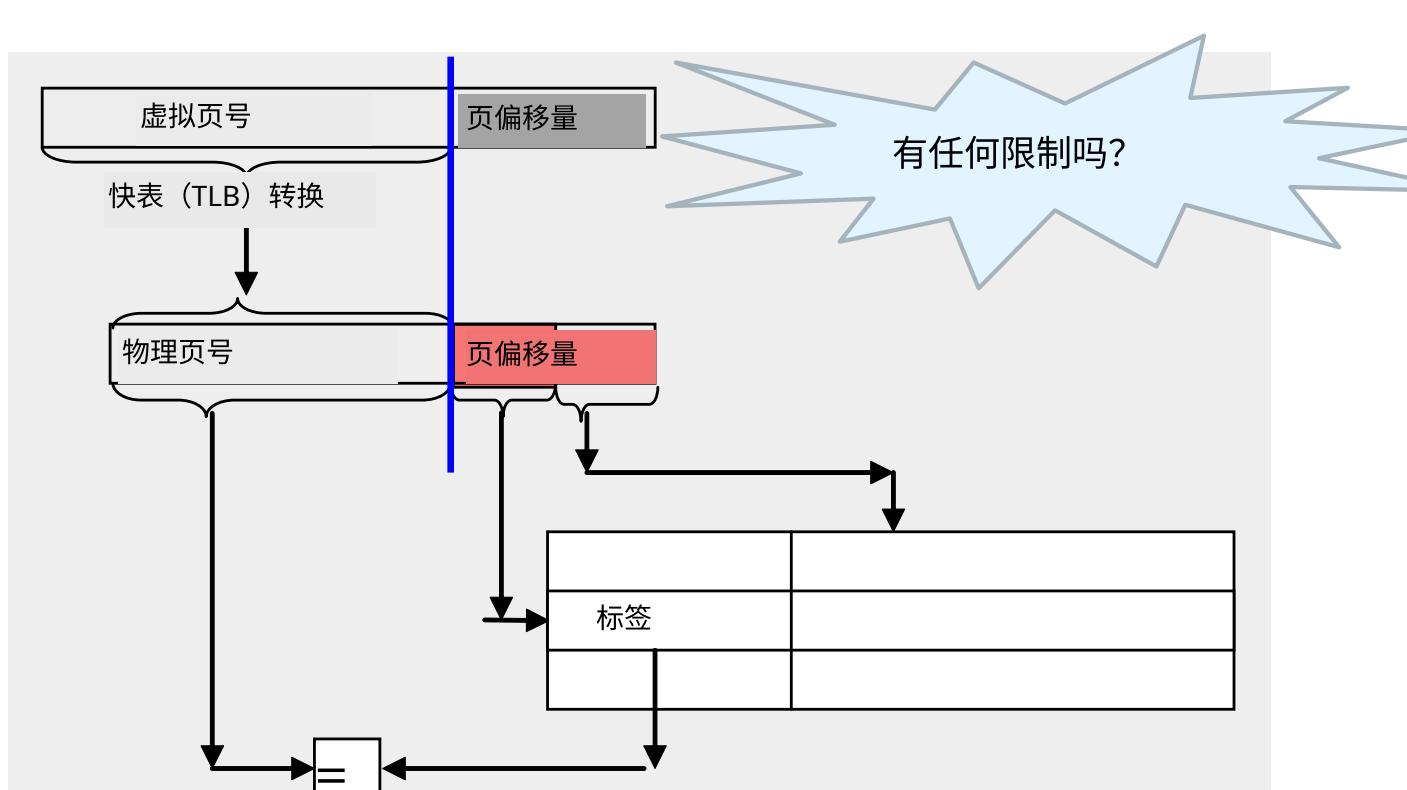
- 处理别名（同义词）；两个不同的虚拟地址映射到相同的物理地址
- 无别名！这有什么影响？
→ HW 反别名：
保证每个缓存块都有唯一的地址 - 在未命中时进行验证
(而不是每次命中时) - 缓存组大小 \leq 页面大小？ - 如果它变大了会怎样？
 软件如何简化这个问题？(称为页面着色)
> I/O 必须与缓存交互，因此需要虚拟地址



如果两个别名的索引位和偏移位被强制设为相同，那么这些别名地址将映射到缓存中的同一个块。

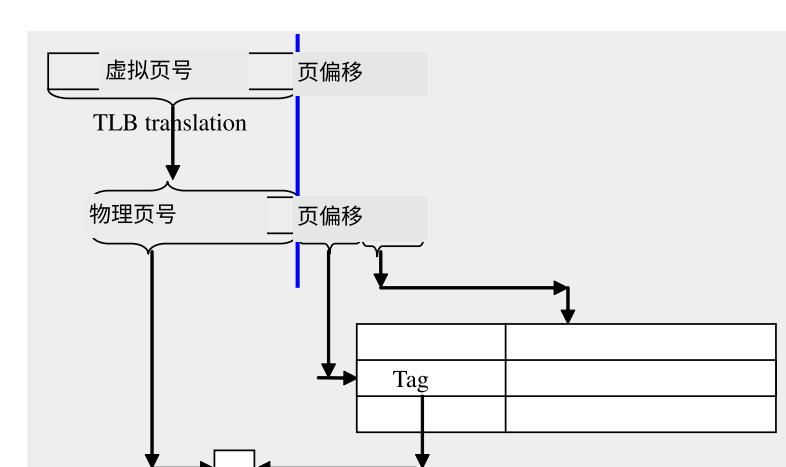
37

重叠地址转换和缓存访问（虚拟索引，物理标记）



39

有什么限制？

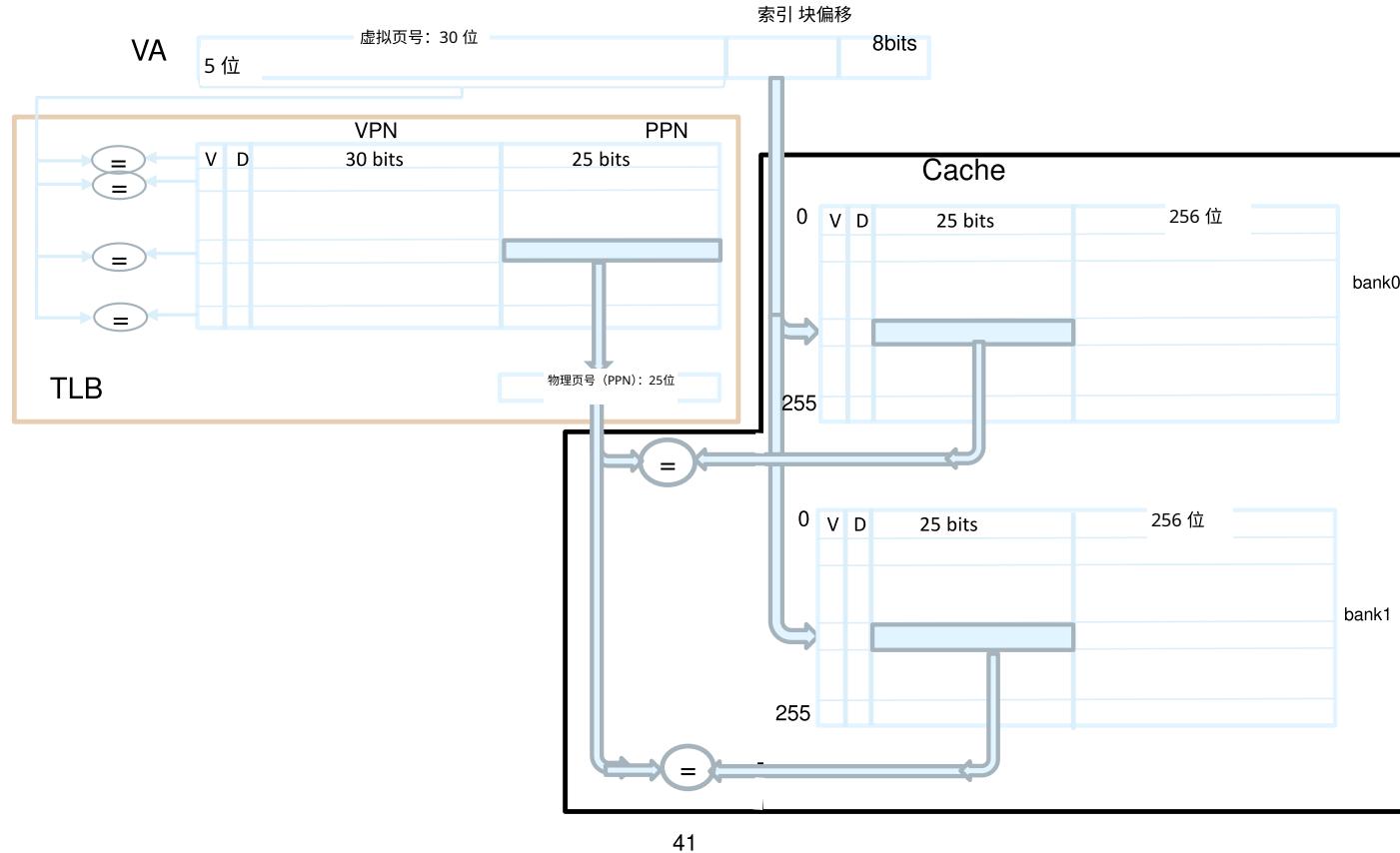


如果是直接映射缓存，那么
 $\text{缓存大小} = 2^{\text{index}} * 2^{\text{blockoffset}} \leq 2^{\text{pageoffset}}$

如何解决这个问题？
使用更高的关联性。假设是四路关联，那么缓存大小可以在索引、标签或偏移量不变的情况下达到4倍的2页偏移。

40

示例：虚拟索引、物理标记缓存。虚拟地址宽度 = 43 位，内存物理地址宽度 = 38 位，页面大小 = 8KB。缓存容量 = 16KB。如果使用虚拟索引、物理标记缓存，且该缓存为 2 路组相联写回缓存，块大小为 32 字节。



41

4th 命中时间减少技术：跟踪缓存

- 找到包含已执行分支的动态指令序列，以加载到缓存块中。
- 块由CPU而非内存布局决定。
- 复杂的地址映射机制

42

为何需要跟踪缓存？

- 每个周期提取 N 条指令
 - > 无指令缓存缺失
 - > 无预测失误
 - > 无数据包中断！

由于每5条指令就有分支，因此缓存只能在一个周期内提供一个数据包。

43

什么是跟踪？

MTrace: 动态指令序列

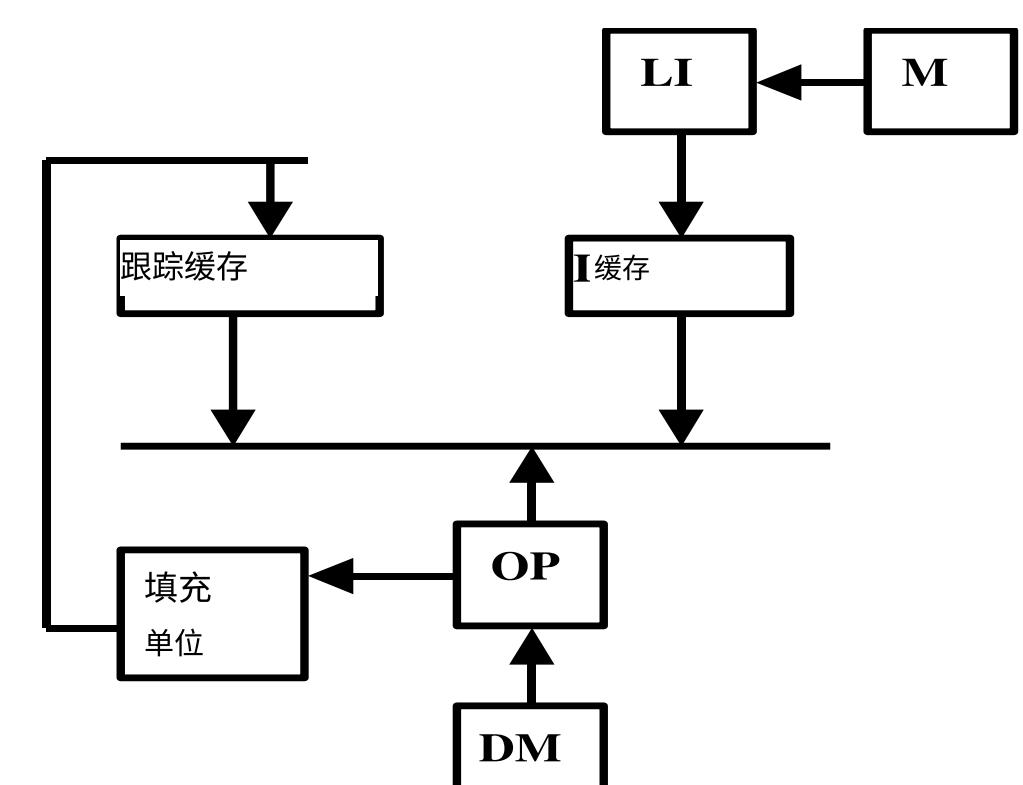
- MTrace: 动态指令序列
 - 当指令（操作）从流水线退出时，将指令段打包成跟踪信息，并将其存储在跟踪缓存中，包括分支指令。
 - 尽管分支指令可能指向不同的目标，但大多数情况下，下一个顺序操作与上一个顺序操作相同。（局部性）

44

谁的提议？

- 佩莱格·魏泽尔 (1994年，英特尔公司)
- 帕特尔/帕特 (1996年)
- 罗滕伯格/J. 史密斯 (1996年)
- 论文：ISCA'98

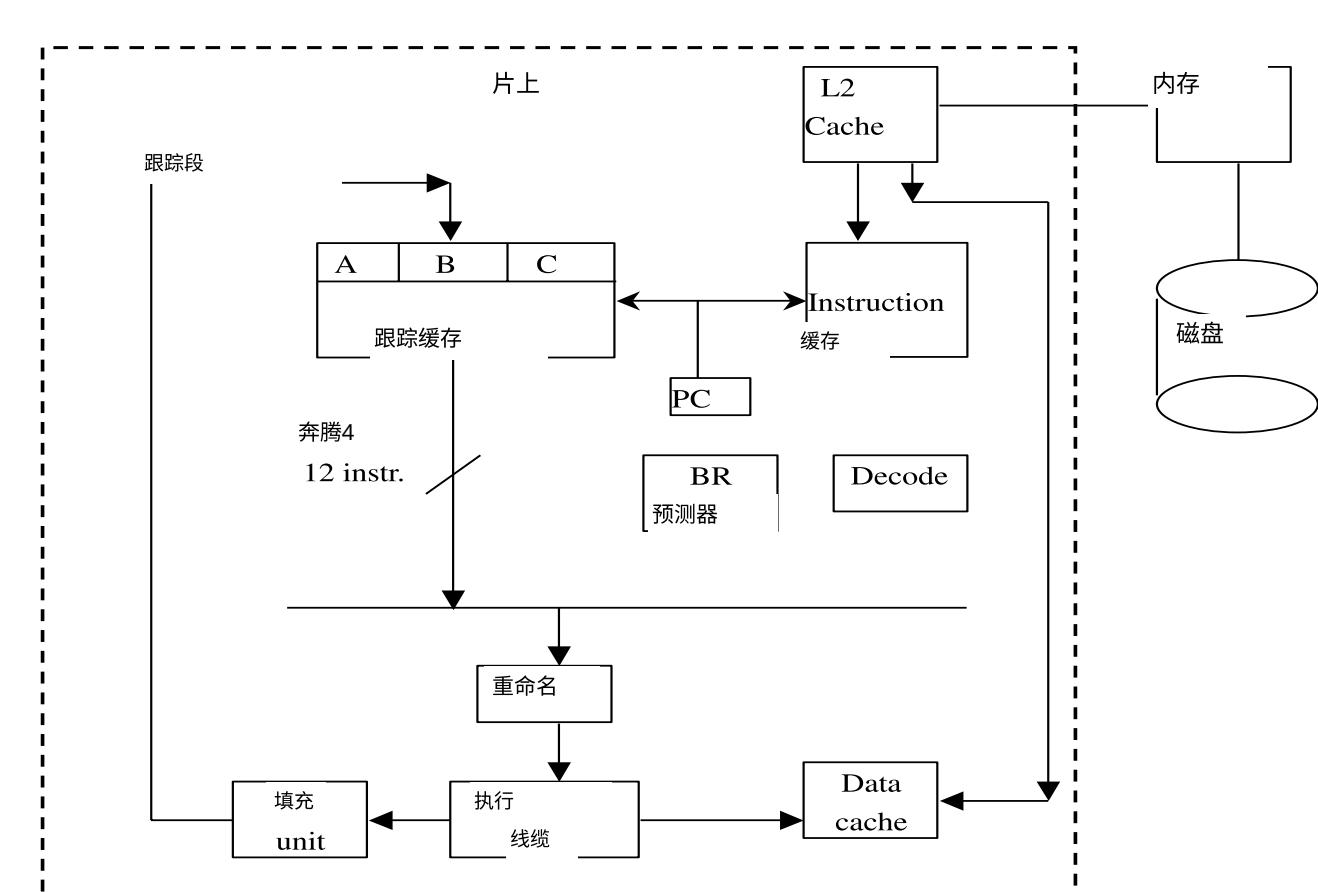
CPU中的跟踪



45

46

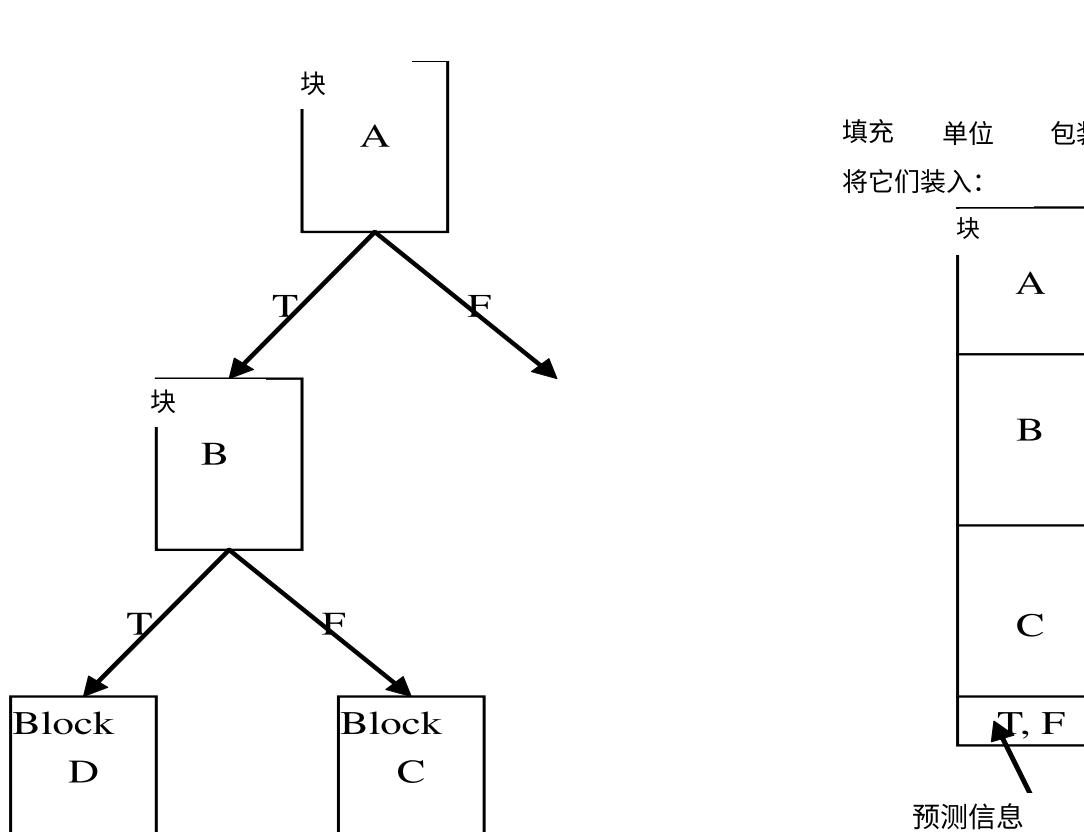
奔腾4：
跟踪缓存，每周期12条指令



47

48

指令段



如何提高缓存性能?

平均访存时间 = 命中时间 + 失效率 × 失效开销

□减少命中时间(4)

> 小型简单的一级缓存、路预测 > 避免地址

转换、踪迹缓存

□增加带宽(3)

> 流水线缓存、多体缓存、非阻塞缓存

□降低缺失代价(5)

> 多级缓存，读缺失优先于写操作

> 关键字优先、合并写缓冲区和牺牲缓存

□降低缺失率(4)

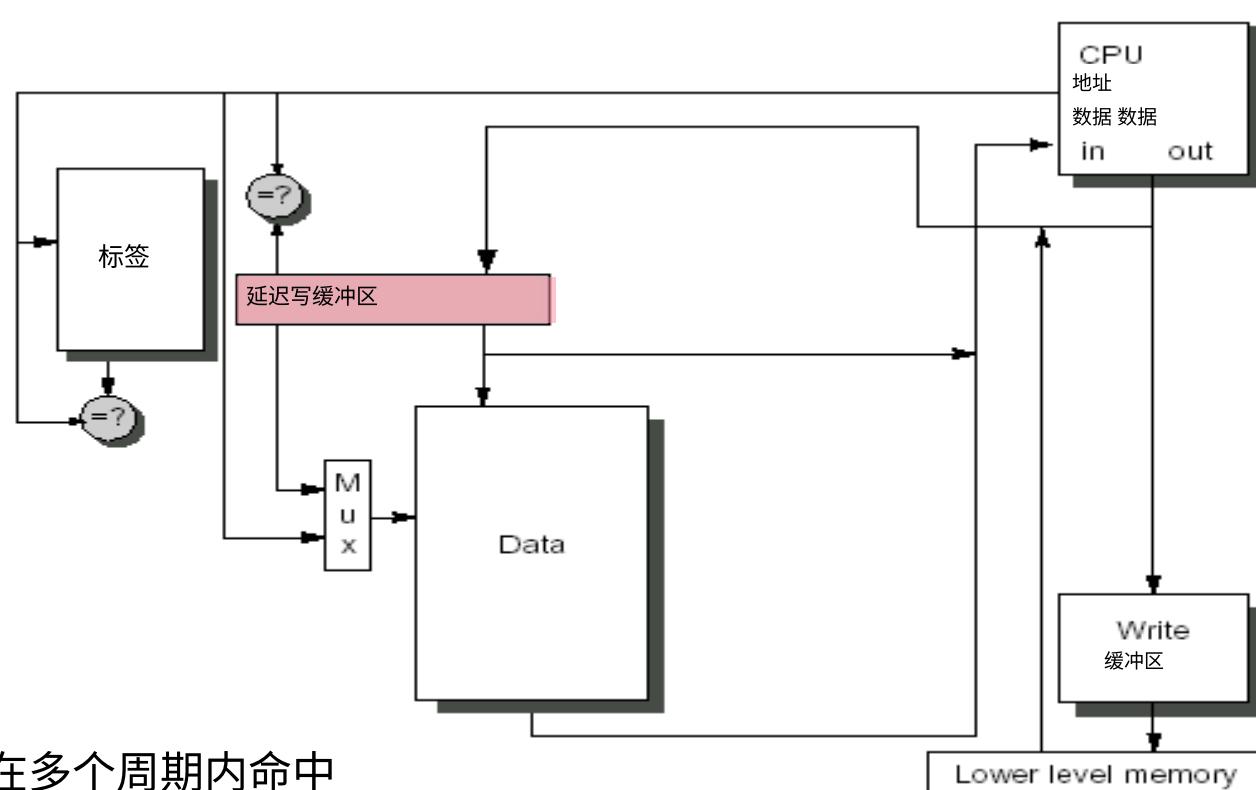
> 更大的块大小、更大的缓存容量、更高的关联性

> 编译器优化

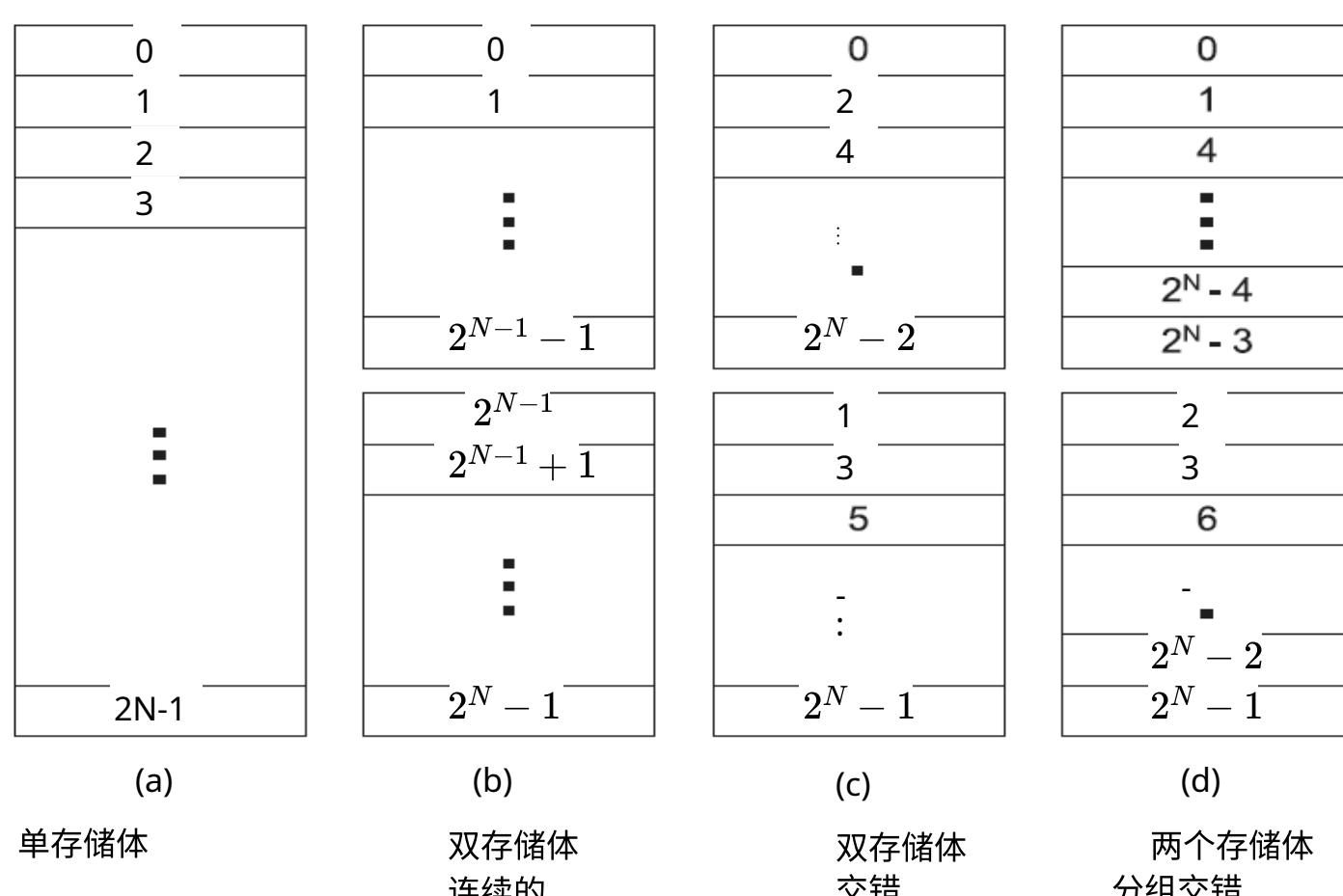
□通过并行化降低缺失代价或缺失率 (1) > 硬件或编译器预取

49

1. 增加缓存带宽：流水线缓存



51



53

非阻塞缓存

假设要执行下面一段程序：

1 寄存器1 := 加载内存(A);

2 寄存器2 := 加载内存(B);

3 寄存器3 := 寄存器1 + 寄存器2;

当执行第一行时，CPU发现地址A不在缓存中，就需要去内存读取。但读取内存的时间很长，此时CPU也不会闲置，就去执行了第二行。然后发现B也不在缓存中。那么此时缓存会怎么做呢？

-(a). 缓存阻塞，等待先把A读进来，然后再去读B。这种叫做阻塞式缓存

-(b). 缓存同时去内存读取B，最终B和A一起进入缓存。这种叫做非阻塞缓存。

流水线缓存

□采用流水线缓存访问以提高带宽 > 示例：

•奔腾处理器：1个周期

•奔腾Pro - 奔腾III处理器：2个周期

•奔腾4 - 酷睿i7处理器：4个周期

□增加分支预测错误的代价 更易于提高关联性

50

2nd 增加缓存带宽：多组缓存

将缓存组织成独立的组以支持同时访问 > ARM Cortex - A8的二级缓存支持1 - 4组 > 英特尔i7的一级缓存支持4组，二级缓存支持8组

□当访问自然地分散在各个组之间时，分组方式效果最佳
→ 地址到组的映射会影响内存系统的性能

□根据块地址对组进行交错，效果良好的简单映射方式是“顺序交错”



52

3rd 增加缓存带宽：非阻塞缓存

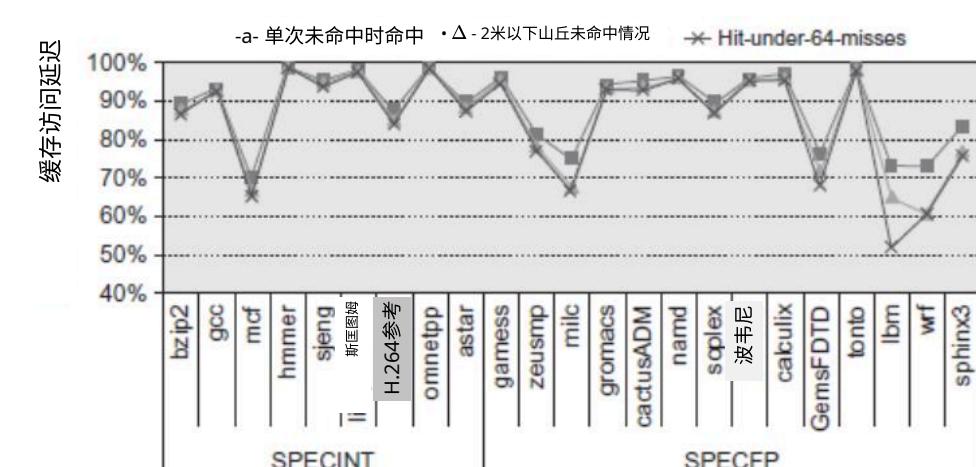
□允许在前一次未命中完成前进行命中

> “未命中时命中”，“多次未命中时命中”

□L2必须支持此功能

□一般来说，处理器可以隐藏L1未命中的代价，但不能隐藏L2未命中的代价

□非阻塞与乱序执行相结合，可以使CPU在数据缓存未命中后继续执行指令。



54

如何提高缓存性能?

平均访存时间 (AMAT) = 命中时间 + 失效率 × 失效开销

□减少命中时间(4)

> 小型且简单的一级缓存，路预测

> 避免地址转换，踪迹缓存

□增加带宽(3)

> 流水线缓存、多体缓存、非阻塞缓存

□降低缺失代价(5)

> 多级缓存，读缺失优先于写操作

> 关键字优先、合并写缓冲区和牺牲缓存

□降低缺失率(4)

> 更大的块大小、更大的缓存容量、更高的关联性

> 编译器优化

□通过并行化降低缺失代价或缺失率 (1)

> 硬件或编译器预取

55

56

第一种缺失惩罚降低技术：多级缓存

多级缓存相关参数

- 此方法聚焦于缓存与主存之间的接口。
- 在主存和一个小型、快速的一级缓存之间添加二级缓存，使缓存既快速又大容量。
- 较小的一级缓存速度足够快，能与快速CPU的时钟周期时间匹配，并能与CPU一同集成在芯片上，从而减少命中时间。
- 二级缓存可以足够大，能捕获许多原本会访问主存的内存访问请求，从而降低有效缺失惩罚。

57

二级缓存的两个概念

□ 定义：

- > 局部缺失率——此高速缓存中的缺失次数除以对该高速缓存的内存访问总次数（缺失率 L_2 ）
- > 全局缺失率——此高速缓存中的缺失次数除以CPU生成的内存访问总次数

使用上述术语，一级高速缓存的全局缺失率就是缺失率 L_1 ，但对于二级高速缓存而言，其全局缺失率为：

$$\begin{aligned} Miss\ rate_{Global+L2} &= \frac{Misses_{L2}}{M} = \frac{M \times Miss\ rate_{L1}}{M} \times \frac{Misses_{L2}}{M \times Miss\ rate_{L1}} \\ &= \frac{Misses_{L1}}{M} \times \frac{Misses_{L2}}{M \times Miss\ rate_{L1}} = Miss\ rate_{L1} \times Miss\ rate_{L2} \end{aligned}$$

59

写缓冲区

-回写时使用写缓冲区来保存被替换的块-读缺失且要替换脏块-正常情况：将脏块写入内存，然后进行读操作-改为将脏块复制到写缓冲区，然后进行读操作，最后进行写操作

-自重启后CPU停顿减少，因为一旦进行读取操作就会立即恢复

-直写模式下，写缓冲区 ⇒ 在缓存未命中时与主存读取存在RAW冲突

- 如果只是等待写缓冲区清空，可能会增加读缺失代价（旧的MIPS 1000会增加50%）
- 读取前检查写缓冲区内容；
- 如果没有冲突，让内存访问继续

61

示例：关键字优先

假设：

缓存块 = 64 字节

L2：获取关键的 8 字节需要 11 个时钟周期（AMD 速龙处理器），然后每获取后续 8 字节需要 2 个时钟周期

不会有对该块其余部分的其他访问。计算优先获取关键字的平均缺失代价。

然后假设后续指令从该块的其余部分每次顺序读取 8 字节数据

比较优先获取关键字和不优先获取关键字的时间。

- 二级缓存的性能计算方式与一级缓存的性能计算方式类似。

□ 二级缓存公式

因此，一级缓存的缺失代价是使用二级缓存的命中时间、缺失率和缺失代价来计算的缓存。

平均访存时间 (AMAT) = 命中时间 $L_1 +$ 缺失 Rate $L_1 \times$ 缺失 Penalty L_1 缺失惩罚 $L_1 =$ 命中 Time $L_2 +$ 缺失 Rate $L_2 \times$ 缺失 Penalty L_2

$$\text{缺失率} = \frac{\text{Misses}_{L1}}{M}$$

$$\text{Miss rate}_{L2} = \frac{\text{Misses}_{L2}}{M_{L2}} = \frac{\text{Misses}_{L2}}{M \times \text{Miss rate}_{L1}}$$

平均访存时间 (AMAT) = 命中时间 $L_1 +$

$$\text{缺失 Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{MissRate}_{L2} - * \text{MissPenalty}_{L1})$$

58

第二次缺失惩罚降低技术：读缺失优先于写操作

- 如果系统有写缓冲区，写入操作可以延迟到读取操作之后进行。

- 然而，系统必须仔细检查写缓冲区，查看正在读取的值是否即将被写入。

60

3nd 缺失代价降低技术：关键字优先与提前重启

- 在重启CPU之前，无需等待整个块加载完成

> 关键字优先——首先从内存中请求缺失的字，并在其到达后立即发送给CPU；在填充块中其余字的同时，让CPU继续执行。也称为环绕式取指和请求字优先

提前重启——一旦块中请求的字到达，就将其发送给CPU并让CPU继续执行

- 通常仅在大块数据中有用

- 空间局部性 ⇒ 倾向于获取下一个顺序字，因此尚不清楚提前重启是否有益

62

4th 缺失惩罚降低技术：合并写缓冲区

- 单字写入被多字写入取代，这提高了缓冲区的效率。

在直写模式下，当发生写缺失时，如果缓冲区包含其他已修改的块，可以检查地址，看这个新数据的地址是否与有效的写缓冲区条目地址匹配。如果匹配，则将新数据与该条目合并。

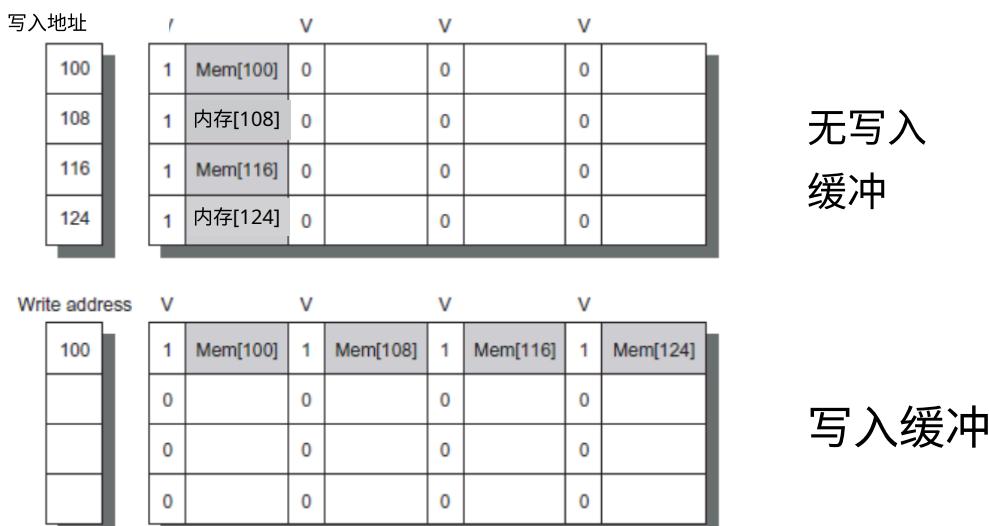
- 该优化还减少了因写缓冲区已满而导致的停顿。

63

64

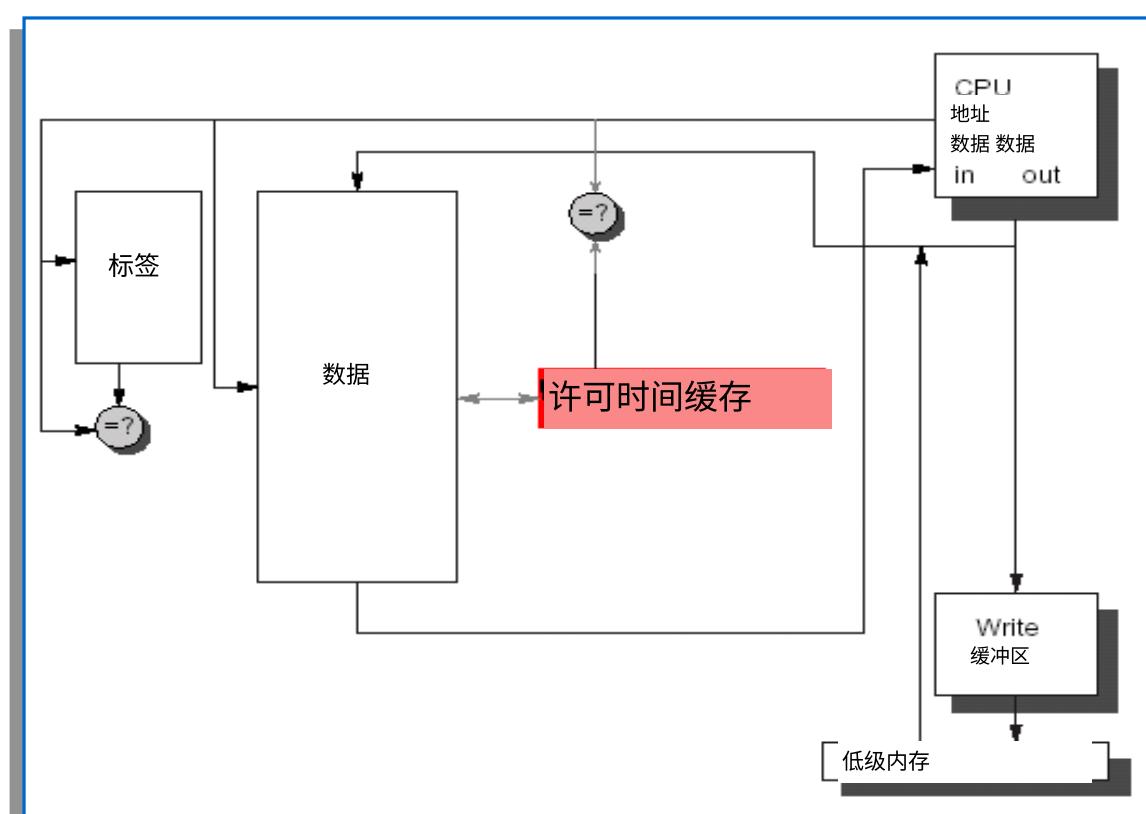
合并写缓冲区

- 当存储到写缓冲区中已处于待处理状态的块时，更新写缓冲区
- 减少因写缓冲区已满而导致的停顿
- 不适用于I/O地址



65

受害者缓存



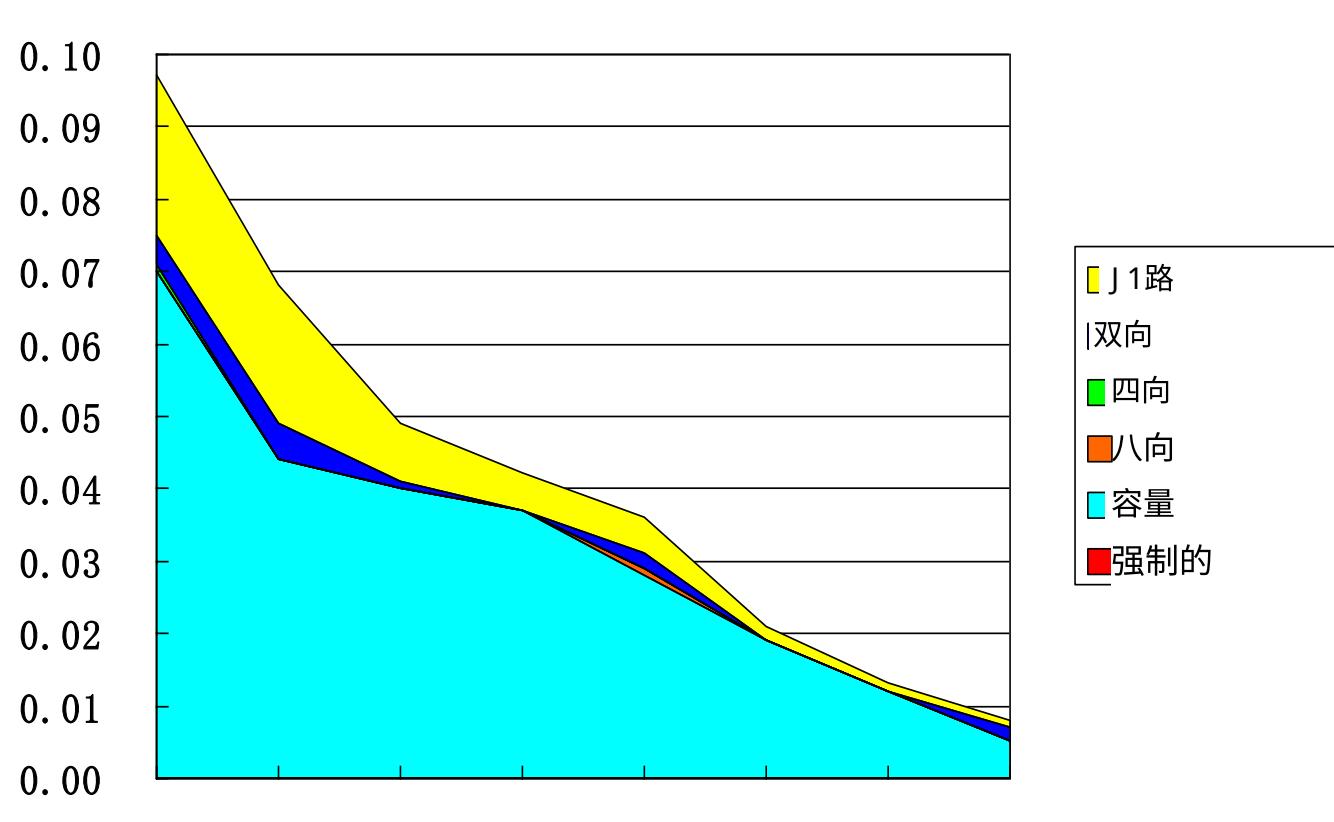
67

如何提高缓存性能？

- 平均访存时间 = 命中时间 + 失效率 × 失效开销
- 减少命中时间(4)
 - > 小型且简单的一级缓存，路径预测
 - > 避免地址转换，跟踪缓存
- 增加带宽(3)
 - > 流水线缓存、多体缓存、非阻塞缓存
- 减少缺失损失(5)
 - > 多级缓存，写操作前的读缺失
 - > 关键字优先、合并写缓冲区和牺牲缓存
- 降低缺失率(4)
 - > 更大的块大小、更大的缓存容量、更高的关联性
 - > 编译器优化
- 通过并行化降低缺失代价或缺失率(1)
 - > 硬件或编译器预取

69

3C绝对缺失率 (SPEC92)



71

5th 缺失惩罚降低技术：牺牲缓存

- A 牺牲缓存是一个小的（通常但不一定）全相联缓存，它保存着主缓存中最近被替换出去的几个块，即牺牲块。

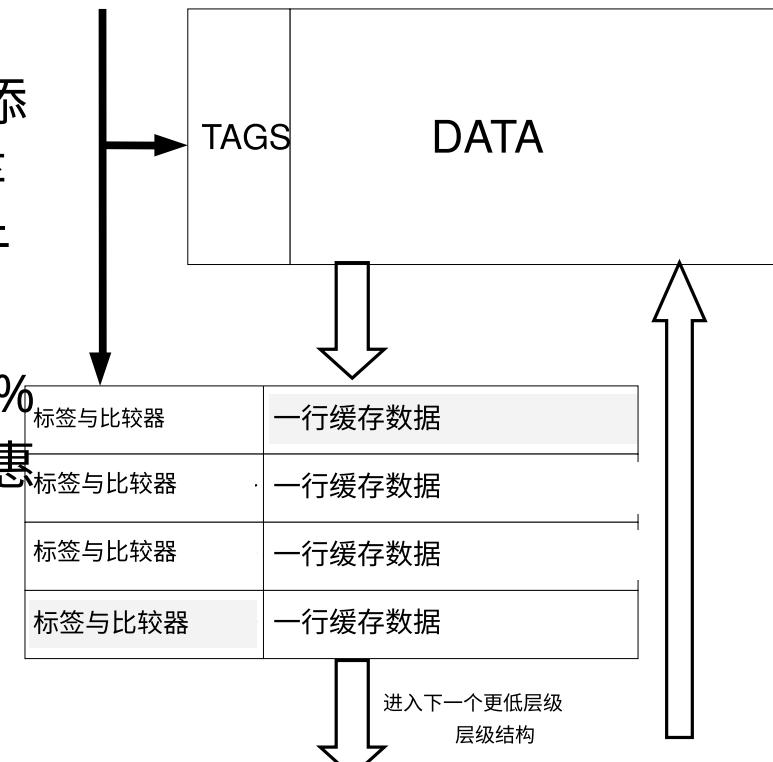
- 在访问下一级更低层次的内存（主存）之前，会在数据缺失时检查这个缓存。
 - 查看它们是否有所需的项
 - > 如果找到，牺牲块和缓存块会进行交换。
 - > AMD速龙处理器有一个包含8项的牺牲缓存（用于回写块的写缓冲区）。

66 如何组合受害者缓存？

如何结合直接映射的快

速命中时间

- 但仍能避免冲突缺失吗？
- 添加缓冲区以存放从缓存中丢弃的数据
- Jouppi [1990]: 对于一个4 KB的直接映射数据缓存，4项的牺牲缓存消除了20%到95%的冲突
- 用于Alpha、惠普机器



68

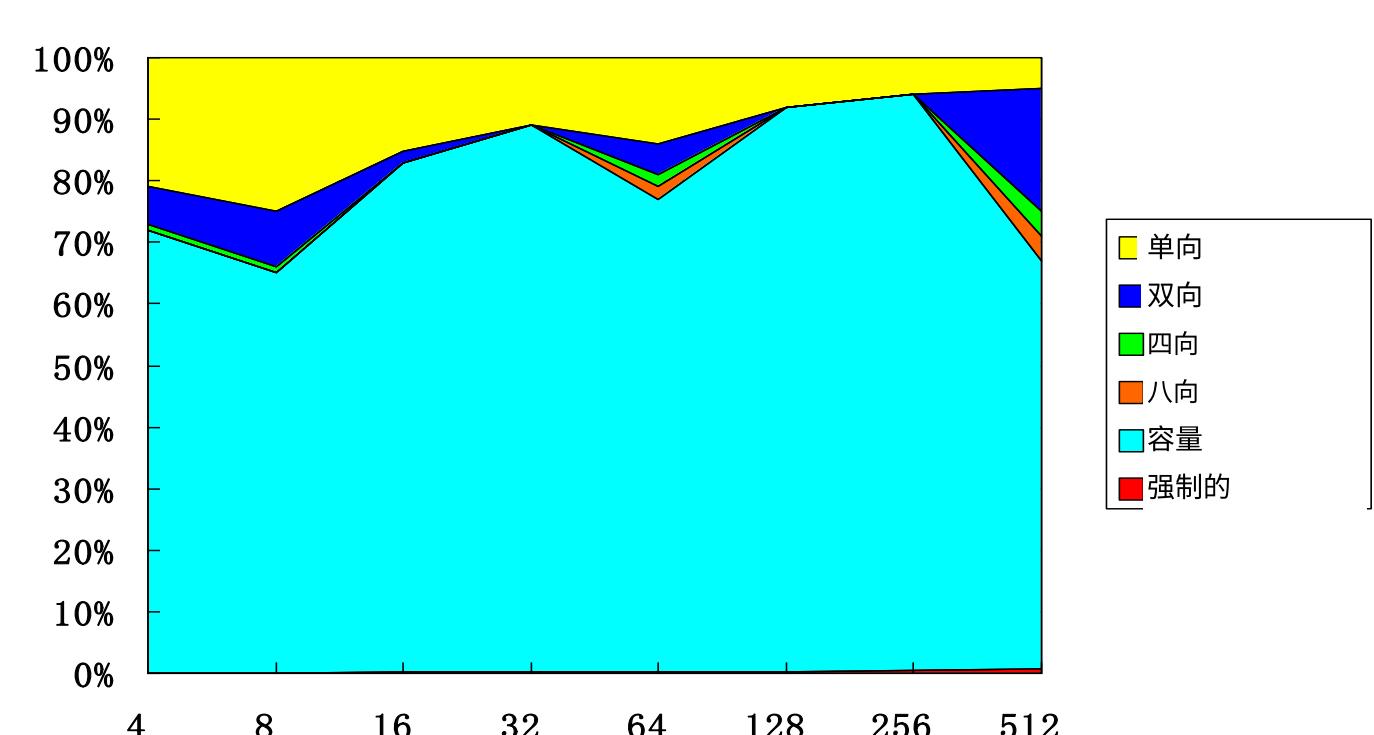
缺失从何而来？

缺失分类: 3C法则

- 强制性缺失——首次访问的块不在缓存中，因此必须将该块调入缓存。也称为冷启动缺失或首次引用缺失。
 - (即使是无限缓存也会出现的缺失)
- 容量缺失——如果缓存无法容纳程序执行期间所需的所有块，由于块被丢弃并在稍后被重新获取，就会发生容量缺失。
 - (全相联大小为X的缓存中的缺失)
- 冲突——如果块放置策略设置为组相联或直接映射，除了强制性缺失和容量缺失之外，还会发生冲突缺失，因为如果有太多块映射到同一组，一个块可能会被丢弃，之后又需要重新获取。也称为碰撞缺失或干扰缺失。(N路组相联、大小为X的缓存中的缺失)
- 第四种“C”：
 - > 一致性——由缓存一致性导致的缺失。

70

3C相对缺失率



缺陷：对于固定块大小

优点：富有洞察力 ⇒ 具有创新性

72

降低缓存缺失率

□ 为了降低缓存缺失率，我们必须消除由三个C因素导致的部分缺失情况。

【除了增大缓存容量，我无法大幅减少容量缺失。
增大缓存容量。

然而，我可以减少冲突缺失并以多种方式处理强制性缺失：

73

第一种降低缺失率的技术：
更大的块大小（固定大小和关联性）

□ 更大的块通过利用空间局部性来降低强制性缺失率。

□ 缺点——曲线呈U形

> 然而，它们可能会因每次缺失时需要获取更多数据而增加缺失代价。

> 此外，由于缓存中能存储的块更少，它们几乎肯定会增加冲突缺失。

> 对于小容量缓存，甚至可能会增加容量缺失

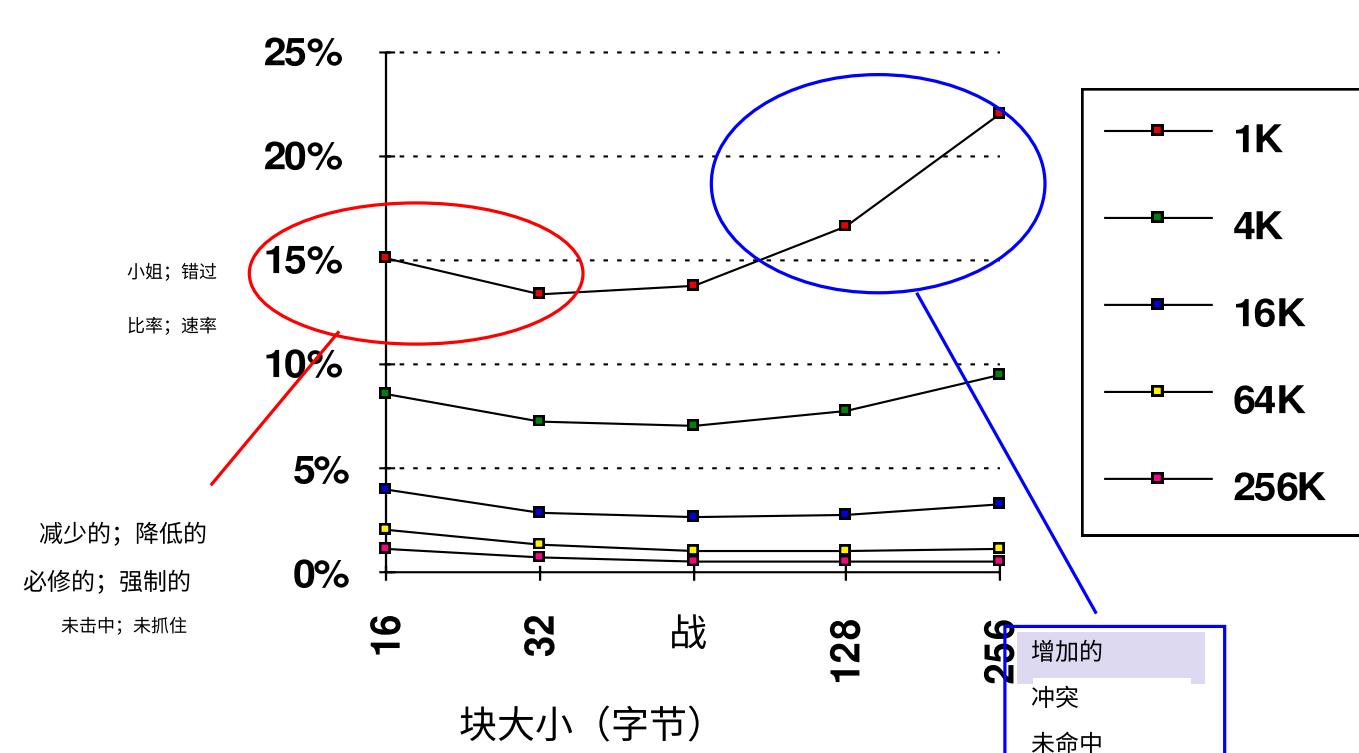
□ 权衡

> 试图同时最小化失效效率和失效代价。

> 块大小的选择取决于下层存储器的延迟和带宽

75

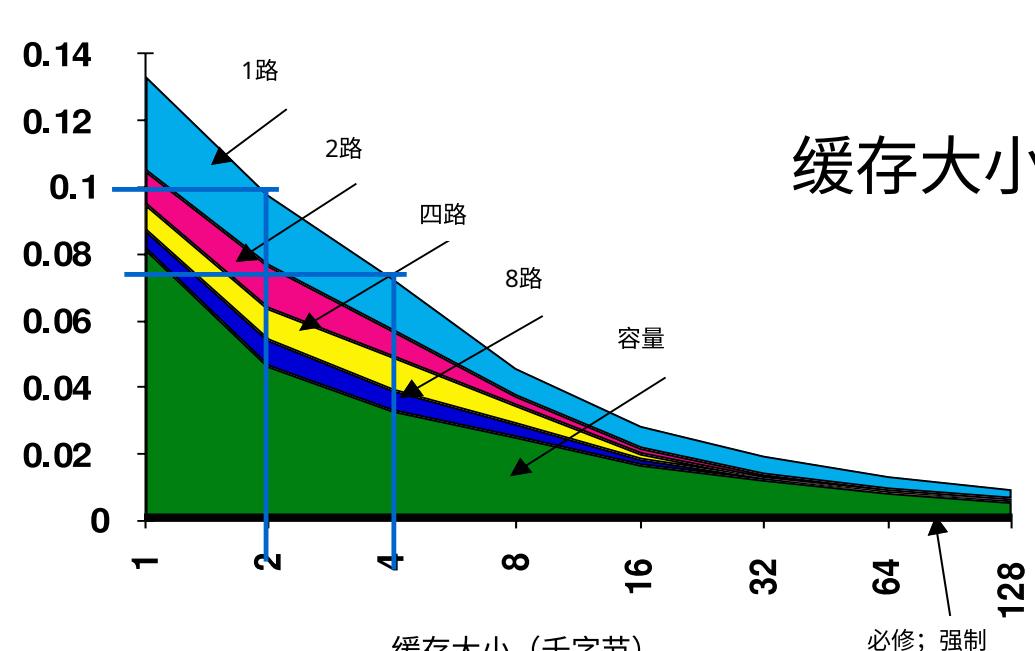
性能曲线呈U形



还有什么因素会提高块大小？

77

2nd 未命中率降低技术：
更大的缓存



□ 经验法则：大小增加一倍，缺失率减半
□ 它能降低什么？

缓存组织方式？

□ 假设缓存总大小不变：
□ 如果出现以下情况会怎样：

1) 更改块大小：

2) 更改关联性：

3) 更改编译器：

3C 中的哪一个明显受到影响？

74

缺失率与块大小有关

块大小	缓存大小			
	4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

76

示例：更大的块大小 (C - 26)

□ 假设：内存有80个时钟周期的开销，然后每2个周期传输16字节。

□ 命中时间为1个时钟周期，与块大小无关。

□ 图5.17中，每种大小对应的哪种块大小的平均访存时间 (AMAT) 最小？

□ 答案：

$$\text{AMAT}_{16\text{-byte block}, 4 \text{ KB}} = 1 + (8.57\% * 82) = 8.027$$
$$\text{AMAT}_{256\text{-byte block}, 256 \text{ KB}} = 1 + (0.49\% * 112) =$$

78

大缓存的优缺点

□ 优点。

> 减少容量缺失

□ 缺点。

> 命中时间更长、成本更高，平均访存时间曲线呈U形
□ 常用于片外缓存

块大小	缺失惩罚	缓存大小			
		4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	7.082	3.411	2.134	1.588
64	88	7.160	3.323	1.933	1.449
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

79

80

□ 对于低关联性（尤其是直接映射）的缓存，冲突缺失可能是个问题。

通过更高的关联性减少冲突缺失以提高失效率

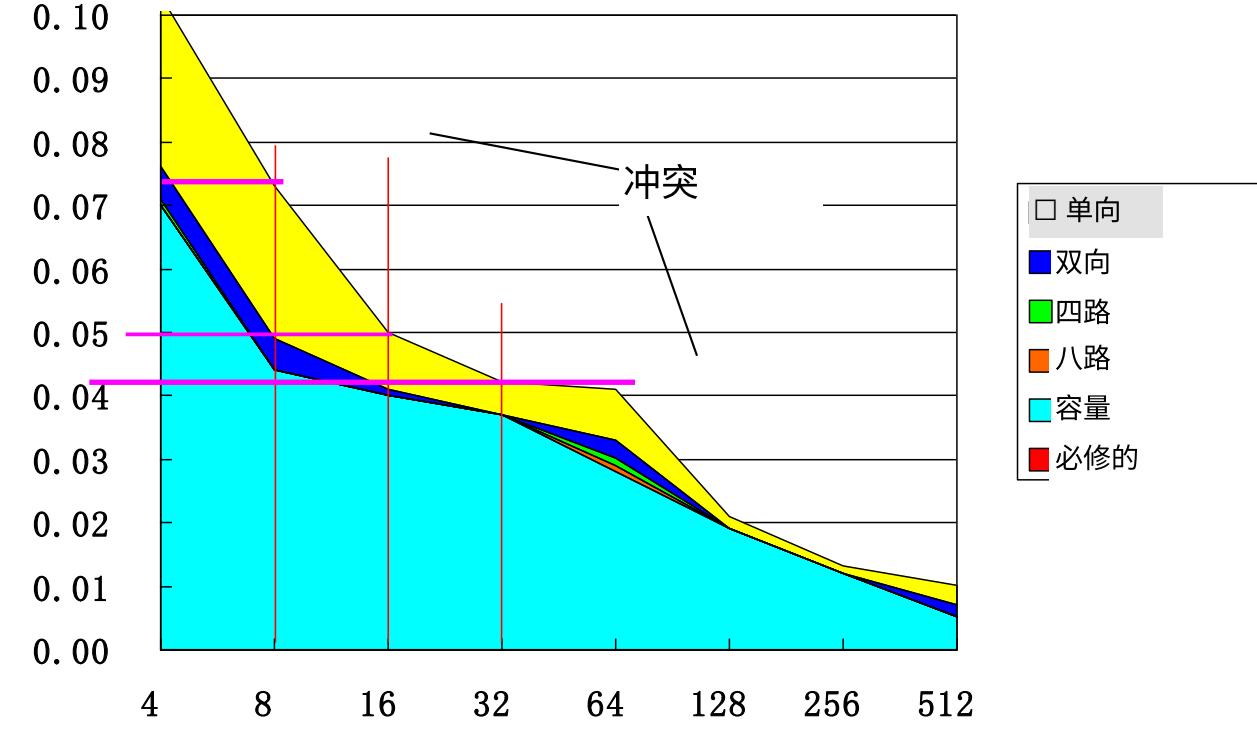
缓存经验法则

□ 2:1 经验法则：大小为 N 的直接映射缓存与大小为 N/2 的 2 路组相联缓存具有相同的失效率。

[对于这些大小的缓存，在减少缺失方面，八路组相联在实际应用中与全相联一样有效。]

关联性

2:1 经验法则



81

秋季学期计算机体系结构

82

关联性与周期时间

□ 注意：执行时间只是最终衡量标准！ □ 为什么周期时间与命中时间相关？

□ 时钟周期时间会增加吗？

> 希尔 [1988] 提出，两路组相联与一路组相联的外部缓存命中时间相比增加 10%，内部 + 2%

> 建议采用大容量简单缓存

平均访存时间与失效率 (P430)

[示例：假设二路组相联的周期时钟时间 (CCT) 为 1.36，四路组相联为 1.44，八路组相联为 1.52，与直接映射的 CCT 对比]

缓存大小 关联性

(KB)	一路组相联	双向	四向	八向
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.33	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.24	2.18	2.25
128	1.52	1.84	1.92	2.00
256	1.32	1.66	1.74	1.82
512	1.20	1.55	1.59	1.66

(红色表示高级微设备公司 (AMD) 未因更高关联性而改善)

83

84

第四种失效率降低技术：编译器优化

□ 该技术无需对硬件进行任何更改即可降低失效率，并通过编译器对指令序列进行重新排序。

□ 指令

> 对内存中的程序进行重新排序，以减少冲突缺失 > 进行分析以查看冲突情况（使用他们开发的工具） □ 数据

> 合并数组：与使用两个数组相比，通过使用复合元素的单个数组来提高空间局部性

> 循环交换：更改循环的嵌套方式，以便按内存中存储的顺序访问数据

> 循环融合：合并两个具有相同循环且部分变量重叠的独立循环

> 分块：通过反复访问数据“块”而非遍历整列或整行来提高时间局部性

□ 将独立矩阵组合成一个复合数组。

□ 提高空间局部性

□ 示例

```
/*修改前*/
整型变量val[SIZE];
整型变量key[SIZE];
```

```
/*修改后*/
结构体merge{
```

```
    整型变量val;
```

```
    整型键;
```

```
}
```

```
结构体合并 合并后的数组[大小]
```

85

86

b. 循环交换

通过改变循环执行的顺序，由于空间局部性的改善，可以减少未命中情况。

/*之前*/

用于 (k = 0; k < 100; k = k + 1)

 用于 (j = 0; j < 100; j = j + 1)

 用于 (i = 0; i < 5000; i = i + 1)

 x [i] [j] = 2 * x [i] [j];

/*之后*/

用于 (k = 0; k < 100; k = k + 1)

 用于 (i = 0; i < 5000; i = i + 1)

 用于 (j = 0; j < 100; j = j + 1)

 x [i] [j] = 2 * x [i] [j];

顺序访问，而非每 100 个单词跨内存访问；

c. 循环融合

□ 通过将代码融合到单个循环中，被提取到缓存中的数据在被换出之前可以被重复使用。

□ 改善时间局部性

□ 示例：

```
/*之前*/
对于 (i = 0; i < N; i = i + 1)           /*之后*/
对于 (i = 0; i < N; i = i + 1)
```

```
对于 (i = 0; j < N; j = j + 1)           对于 (j = 0; j < N; j = j + 1)
```

```
    a [i] [j] = 1/b [i] [j]*c [i] [j]; {
```

```
对于 (i = 0; i < N; i = i + 1)
```

```
    a [i] [j] = 1/b [i] [j]*c [i] [j];
```

```
对于 (j = 0; j < N; j = j + 1)
```

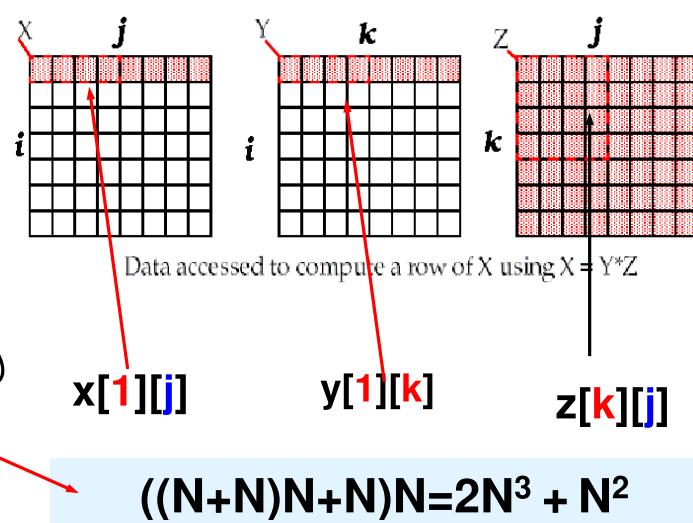
```
    d [i] [j] = a [i] [j]*c [i] [j]; }
```

```
    d [i] [j] = a [i] [j]*c [i] [j]; }
```

d. Unoptimized Matrix Multiplication

```
/* 之前 */
为 (i = 0; i < N; i = i + 1) 为
(j = 0; j < N; j = j + 1)
{r = 0;

    对于 (k = 0; k < N; k = k + 1)
r = r + y[i][k]*z[k][j]; x[i][j] = r; // 两个内
循环: > 写入 X [] 的 1row 的 N 个元素
```



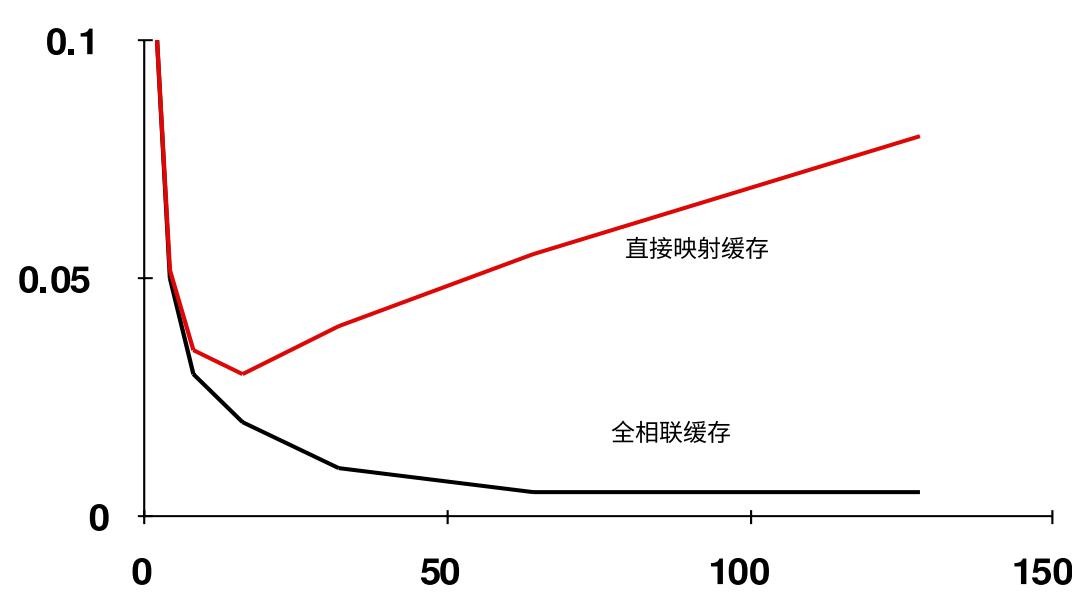
> 重复读取 Y [] 的 1 行中的 N 个元素 > 读取 Z [] 的所有 NxN 个元素

大容量缺失: N 和缓存大小的函数: > $2N^3 + N^2 \Rightarrow$ (假设无冲突; 否则...)

□ 思路: 对合适的 Bx B 子矩阵进行计算

89

通过分块减少冲突缺失



□ 非全相联缓存中的冲突缺失与块大小对比

> 林等人 [1991] 发现, 块因子为 24 时的缺失率是块因子为 48 时的五分之一, 尽管两者都能放入缓存

91

第五种降低缺失率的技术:

路预测与伪相联缓存

使用这两种技术可减少冲突缺失, 同时保持直接映射缓存的命中速度

预测位 - 伪关联

□ 路预测 (Alpha 21264)

> 缓存中保留额外的位, 以预测下一次缓存访问的路或组内块。

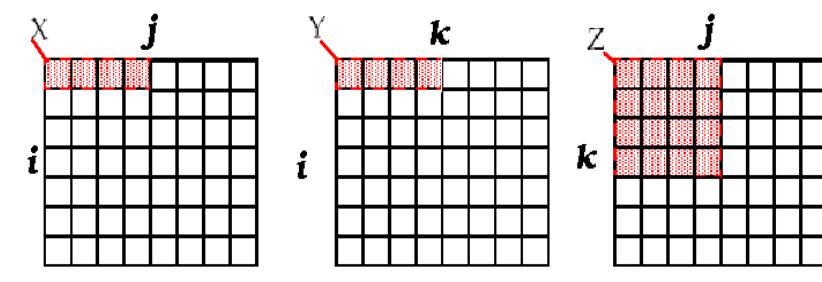
> 如果预测器预测正确, 指令缓存延迟为 1 个时钟周期。

> 如果预测错误, 它会尝试另一个块, 更改路预测器, 并且延迟为 3 个时钟周期。

> 使用SPEC95进行的模拟表明, 组预测准确率超过 85%, 因此路预测在超过 85% 的指令提取中节省了流水线阶段。

Blocking optimized Matrix Multiplication

□ 矩阵乘法是通过先将子矩阵相乘来执行。/* 之后 */ 对于 (jj = 0; jj < N; jj = jj + B) 对于 (kk = 0; kk < N; kk = kk + B)



BN BN B × B
B 称为分块因子

对于 (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[j][j] + r;
};

Y 受益于空间局部性, Z 受益于时间局部性

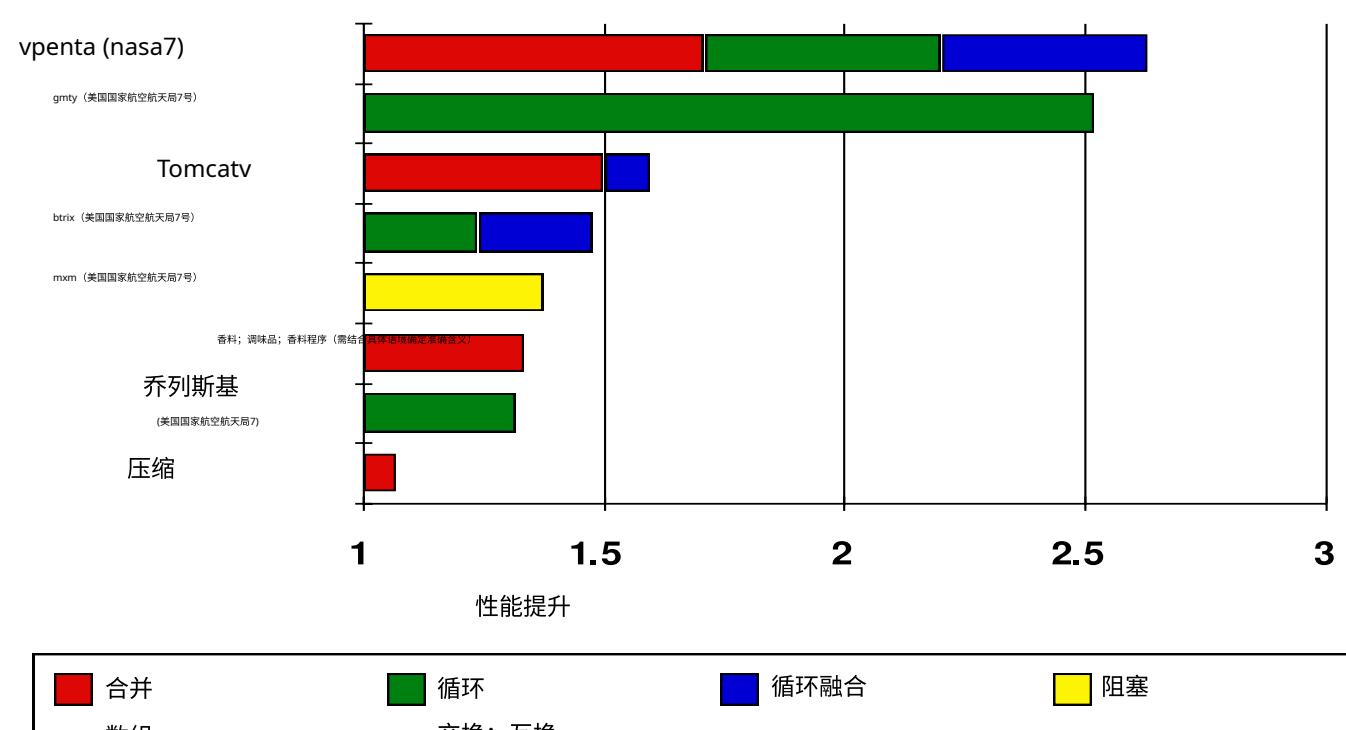
$$(BN + BN) + B^2) \times (N/B)^2 = 2N^3/B + N^2$$

Accessed For N^3/B + 2N^2 90 的容量缺失

从 $2N^3 + N^2$ 到 $N^3/B + 2N^2 90$ 的容量缺失

减少缓存的编译器优化总结

缺失 (手动)



92

伪相联缓存 (列相联)

□ 如何结合直接映射的快速命中时间, 并减少两路组相联缓存的冲突缺失?

| 分割缓存: 发生缺失时, 检查缓存的另一半, 看数据是否存在; 若存在, 则为伪命中 (慢速命中)

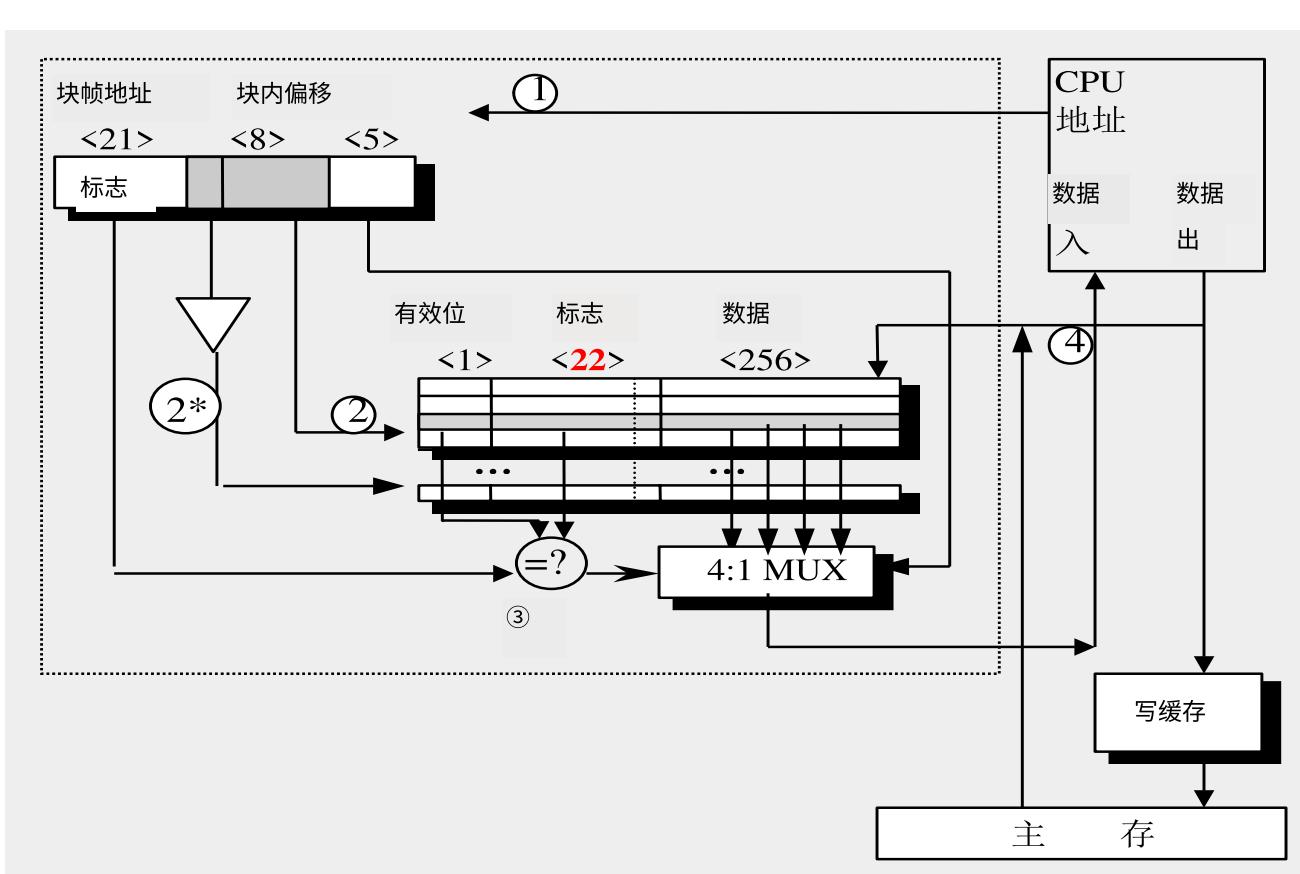


缺点: 如果命中需要 1 或 2 个周期, CPU 流水线会很困难 > 对于不直接与处理器相连的缓存 (二级缓存) 效果更好

用于MIPS R1000二级缓存, 在UltraSPARC中类似

93

伪关联缓存



95

如何提高缓存性能? 平均访存时间 = 命中时间 + 缺失率 × 缺失惩罚

□ 减少命中时间(4)

> 小型简易一级缓存, 路径预测

> 避免地址转换, 跟踪缓存

□ 增加带宽(3)

> 流水线缓存、多体缓存、非阻塞缓存

□ 减少缺失损失(5)

> 多级缓存, 读缺失优先于写操作

> 关键字优先、合并写缓冲区和牺牲缓存

□ 降低缺失率(4)

> 更大的块大小、更大的缓存大小、更高的关联性

> 编译器优化

| 通过并行化降低缺失代价或缺失率 (1) > 硬件或编译器预取

□ 使用高带宽内存 (HBM) 扩展内存层次结构

96

硬件预取

在CPU实际需要数据之前从内存中获取数据的操作。

- 未命中时预取两个块（包括下一个连续块）
- 连续块

通过在请求数据之前检索数据，这减少了强制性缺失。

当然，这可能会通过从缓存中移除有用的块来增加其他缺失。

> 因此，许多缓存将预取的块保存在一个特殊的缓冲区中，直到实际需要它们为止。

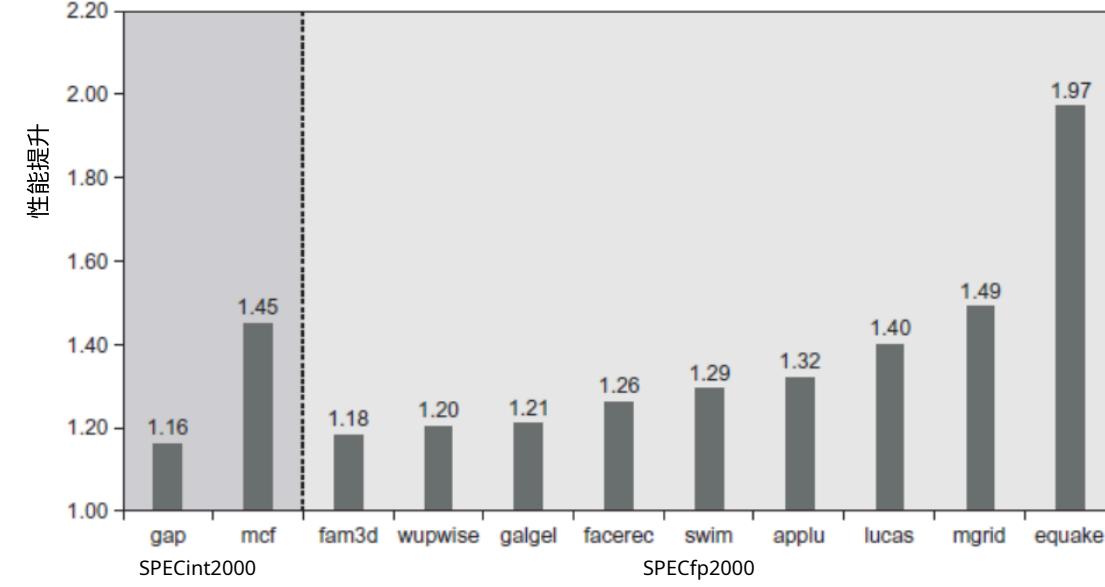
例如，指令预取

> Alpha 21064在未命中时提取2个块

> 额外的块放置在“流缓冲区”中

> 未命中时检查流缓冲区

预取依赖于有额外的内存带宽，且使用时不会产生代价



奔腾4预取

97

2nd 未命中代价/比率降低技术：编译器控制的预取

编译器在需要数据之前插入预取指令

一、非故障性：预取不会引发异常，这是一种推测执行形式

与循环展开和软件流水相结合

寄存器预取（惠普PA - RISC加载）

> 绑定预取：请求直接加载到寄存器中。- 地址和寄存器必须正确！< 缓存预取（MIPS IV、PowerPC、SPARC v. 9）> 非绑定预取：加载到缓存中。- 可能不正确。会出错吗？

发出预取指令需要时间 > 预取指令的成本是否小于减少缺失带来的节省？> 更高的超标量降低了指令带宽的难度

98

示例：

编译器控制的预取

用于 $(i = 0; i < 3; i = i + 1)$

用于 $(j = 0; j < 100; j = j + 1)$

每次迭代7个周期

$a[i][j] = b[j][0] * b[j + 1][0];$

每块16字节，每个元素8字节，每块2个元素

$A[i][j] : 3 * 100$ j 的偶数会发生缺失，

奇数会命中，总共150次缺失

$B[i][j] : 101 * 3$ 对于i的每次迭代，访问相同的元素

$j = 0 B[0][0], B[1][0] 2$

$j = 1 B[1][0], B[2][0] 1$

总共 $2 + 99 = 101$ 次缺失

性能： $300 * 7 + 251 * 100 = 27200$

99

示例续：

编译器控制的预取

对于 $(j = 0; j < 100; j = j + 1)$ {

9个周期/迭代

预取($b[j+7][0]$); 预取($a[0][j+7]$);

$a[0][j] = b[j][0] * b[j + 1][0];$; b 出现 7 次未命中

a 出现 4 次未命中

对于 $(l = 1; l < 3; l = l + 1)$

每次迭代8个周期

对于

$(j = 0; j < 100; j = j + 1)$ { 预

取($a[i][j+7]$);

$a[i][j] = b[j][0] * b[j + 1][0];$; 4 未命中，

$a[1][j] a[2][j]$ 4 次未命中，总计：19次未命中

以400条预取指令为代价节省232次缓存未命中。

性能： $100 * 9 + 200 * 8 + 19 * 100 = 4400$ 周期

101

使用高带宽内存（HBM）扩展层次结构

另一种方法（合金缓存）：> 将标签和数据组合在一起 > 使用直接映射

两种方案在未命中时都需要两次动态随机存取存储器（DRAM）访问

> 两种解决方案：

- 使用映射来跟踪块
- 预测可能的未命中情况

100

使用高带宽内存（HBM）扩展层级结构

HBM：高带宽内存

使用动态随机存取存储器（DRAM）构建大容量的四级缓存（128兆字节至1吉字节）

较小的块需要大量的巨型标签存储

块大小：64 B/4 KB；块编号： $2^{24}/2^{18}$; 16M/256 K

大块大小的问题

> 不必要的内容，传输效率低下

> 更多的冲突和一致性缺失

一种方法（L-H）：

> 每个SDRAM行都是一个块索引

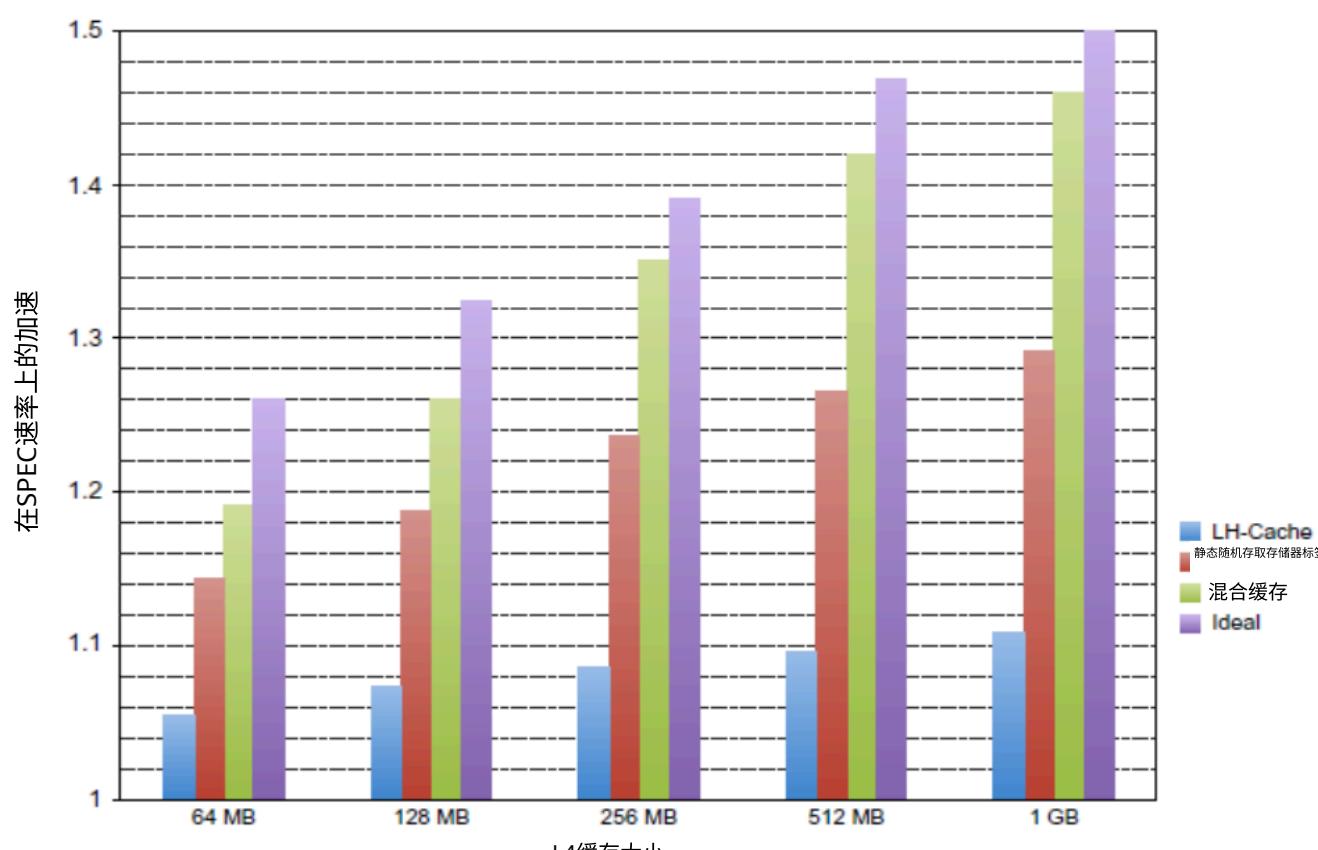
> 每行包含一组标签和29个数据段

> 29路组相联

> 命中需要一次比较并交换操作

102

使用高带宽内存扩展缓存层级



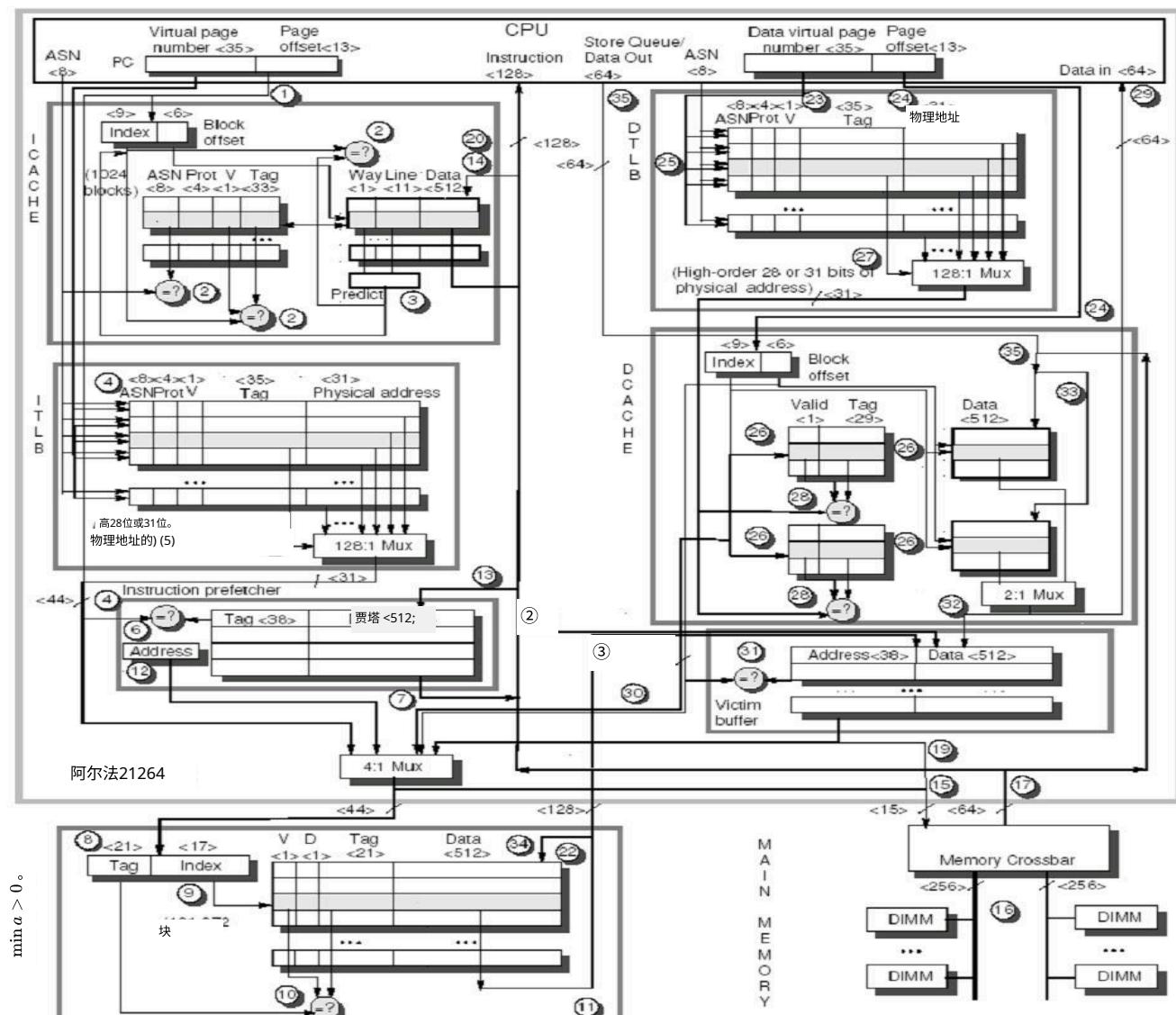
103

104

总结

技术	命中时间	带宽	未命中惩罚	失效率	功率消耗	硬件成本/复杂度	评论
小型简易缓存	+	-	-	+	0	普通；广泛使用	
路径预测缓存	+	-	-	+	1	用于奔腾4	
流水线式和分块式缓存	-	+	-	-	1	广泛使用	
非阻塞缓存	+	+	-	-	3	广泛使用	
关键词单优先和早期重启	+	-	-	-	2	广泛使用	
合并写缓冲区	-	-	+	-	1	与直写方式一起广泛使用	
减少缓存未命中的编译器技术	-	-	-	+	0	软件是一项挑战，但许多编译器可以处理常见的线性代数计算	
指令和数据的硬件预取	+	+	-	-	2条指令/3个数据	大多数处理器提供预取指令；现代高端处理器还能在硬件层面自动预取	
编译器控制的预取	+	+	-	-	3	需要无阻塞缓存：可能存在指令开销：在许多CPU中	
高带宽内存 (HBM) 作为额外的缓存层级	+/-	-	+	+	3	依赖于新的封装技术。效果在很大程度上取决于命中率的提升	

105



虚拟内存与虚拟机

通过虚拟内存进行保护

> 将进程保留在其自身的内存空间中

架构模型

- > 提供用户模式和管理模式
- > 保护CPU状态的某些方面
- > 提供在用户模式和管理模式之间切换的机制
- > 提供限制内存访问的机制
- > 提供TLB以转换地址

109

通过虚拟机进行保护

支持隔离和安全

- > 在许多不相关的用户之间共享一台计算机
- > 得益于处理器的原始速度，使得开销更易接受

允许向用户程序呈现不同的指令集架构和操作系统

- > “系统虚拟机”(在二进制指令集架构层面)
- > SVM软件被称为“虚拟机监视器”或“管理程序”
- > 在监视器下运行的各个虚拟机被称为“客户虚拟机”

示例：Alpha 21264内存层次结构

21264是一款乱序执行处理器

> 每个时钟周期最多提取4条指令，每个时钟周期最多执行6条指令。

虚拟地址

> 48位虚拟地址和44位物理地址

/ 43位虚拟地址和41位物理地址；

> 无论哪种情况，Alpha 都会将物理地址空间一分为二，下半部分用于内存地址，上半部分用于I/O地址。

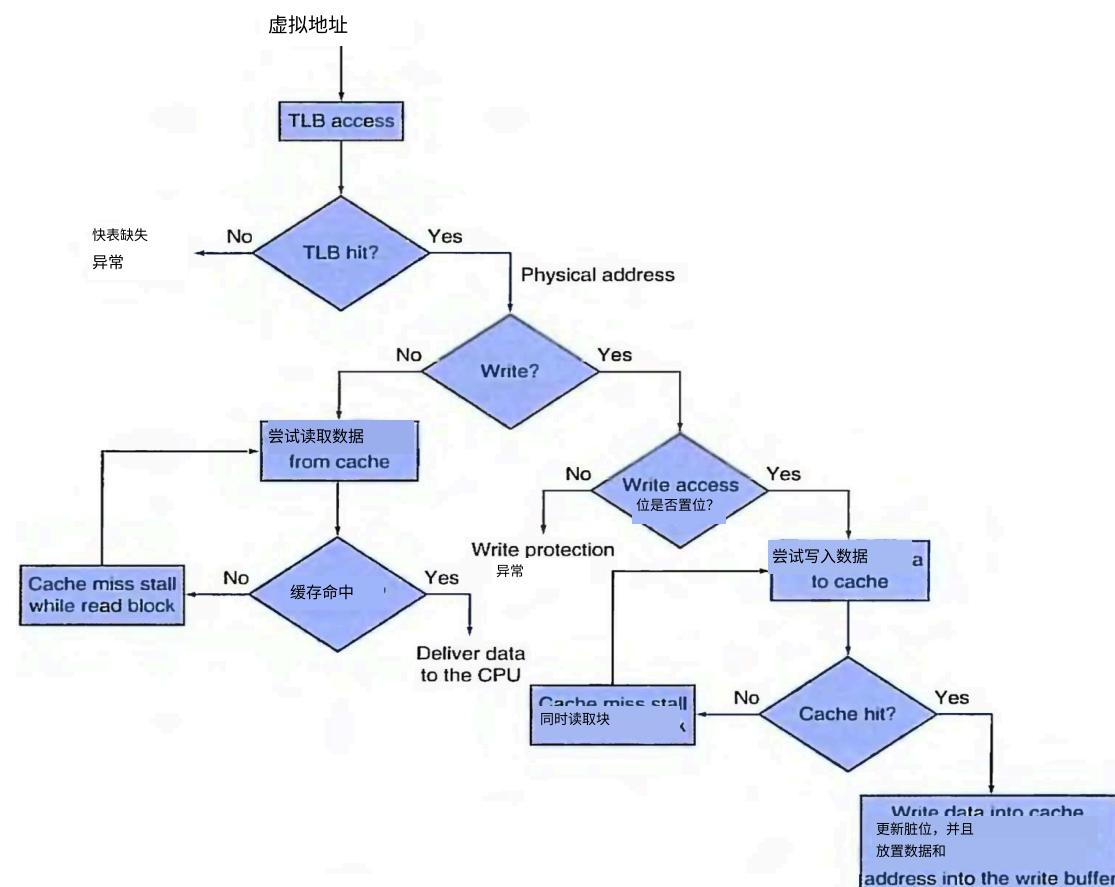
当Alpha开机时。

> 芯片上的硬件从外部PROM串行加载指令缓存。(16K条指令)

> 相同的串行接口(和PROM)还会加载配置信息，这些信息指定了二级缓存的速度/时序、系统端口的速度/时序以及许多其他必要信息

106

重要概念：缓存



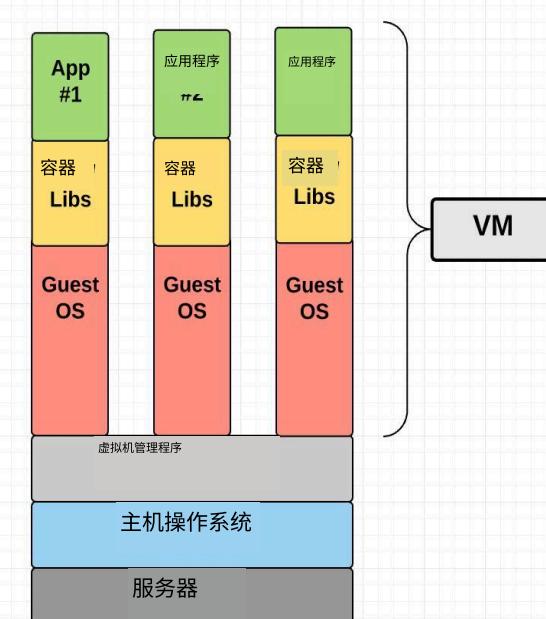
108

虚拟机

一个高效、隔离的副本

真实机器。

- > VMM (虚拟机监视器) > 为程序提供一个与原始机器基本相同的环境。> 在这个环境中运行的程序，其速度最多只会有轻微下降。> VMM完全控制系统资源。



110

虚拟机监视器的要求

客户软件应：

- > 表现得就像在原生硬件上运行一样
- > 无法更改真实系统资源的分配

虚拟机监视器(VMM)应能够对客户机进行“上下文切换”

硬件必须支持：

- > 系统和用户处理器模式
- > 用于分配系统资源的特权指令子集

111

112

虚拟机对虚拟内存的影响

□ 每个客户操作系统维护自己的一组页表

- > 虚拟机监视器在物理内存和虚拟内存之间添加了一层称为“真实内存”的内存
- > 虚拟机监视器维护影子页表，该表将客户虚拟地址映射到物理地址
 - 需要虚拟机监视器检测客户对其自身页表的更改
- 如果访问页表指针是特权操作，则会自然发生

113

谬误与陷阱

□ 根据一个程序预测另一个程序的缓存性能 □ 模拟足够多的指令以获得内存层次结构的准确性能指标

□ 在基于缓存的系统中未能提供高内存带宽

114

结束。

115

116

扩展用于虚拟化的指令集架构 (ISA)

□ 目标：

> 避免刷新转换后备缓冲器 (TLB)

> 使用嵌套页表而非影子页表

> 允许设备使用直接内存访问 (DMA) 来移动数据

> 允许客户操作系统处理设备中断

>>出于安全考虑：允许程序管理代码和数据的加密部分