线程级并行性(1)

引言

- 计算机体系结构的新时代——多处理器(multiprocessor)
  - 起因：指令级并行性的回报率不断下降，对功耗的日益关注
  - 作用：在低端和高端都扮演着重要的角色

- 促进因素
  - 能耗和硅片成本增长速度超过性能的提升——入不敷出
  - 高强度服务的关注度增加，如云计算和SaaS
  - 数据密集型应用的需求增长
  - 对桌面计算性能增长的关注不再重要，因为高端计算和计算密集型应用可以在云端完成
  - 如何有效地使用多处理器在理解上的变化，大型数据库中固有的并行性、科学和工程代码中客观世界的并行性和大量独立请求中的并行性
  - 通过复制而不是独特设计来利用设计投资的优势

引言

- 线程级并行性(Thread-Level Parallelism, TLP)
  - 存在多个程序计数器(Program Counter)
  - 主要通过 MIMD 模型开发
  - 焦点：紧耦合(tightly coupled)共享内存(share memory)多处理器(multiprocessors)
    - 两颗到几百颗处理器
    - 通过共享内存进行通信和协作
- 紧耦合共享内存多处理器
  - 单个操作系统控制多处理器的协作和使用
  - 通过共享地址空间来共享内存
  - 利用线程级并行性的两个软件模型
    - 一组紧耦合线程在单个任务上协作执行——平行处理(parallel processing)
    - 来自一个或多个用户的多个相对独立进程的执行——请求级并行性(request-level parallelism)

引言

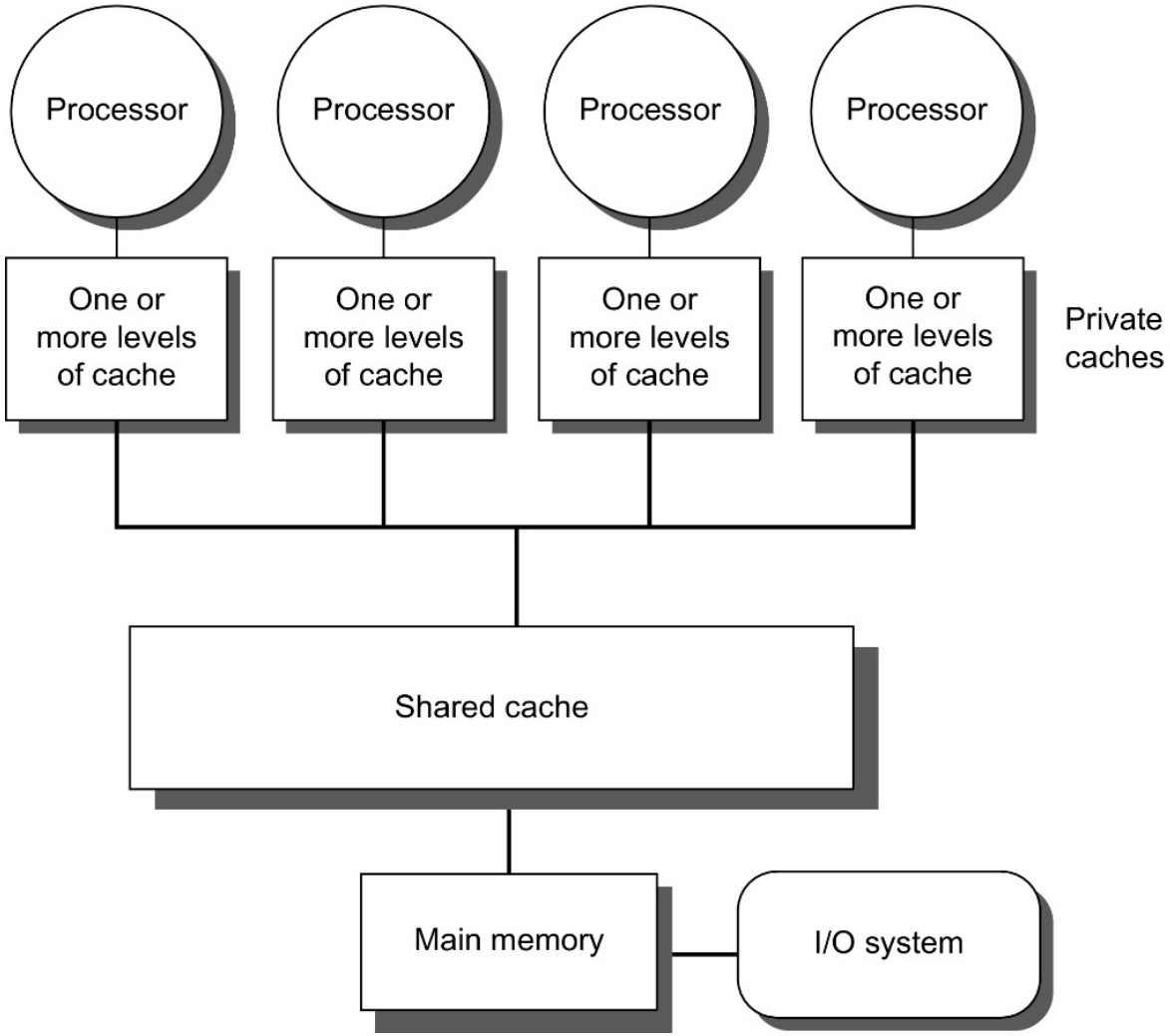
- n 颗处理器构成的 MIMD 多处理器需要至少 n 个线程或进程来执行
  - 对于多线程处理来说，则需要更多的线程。

- 粒度(grain size)：分配给每个线程的计算量
  - 线程级并行性是操作系统或程序员来发现的
  - 线程由几百到几百万条并行执行的指令组成
  - 线程可以用于数据级并行性，但是可能开销超过收益（入不敷出）

- 共享内存多处理器可以按照处理器数目分为两类：
  - 对称（共享内存）多处理器：小到中规模处理器数目
  - 分布式共享内存多处理器：大规模处理器数目

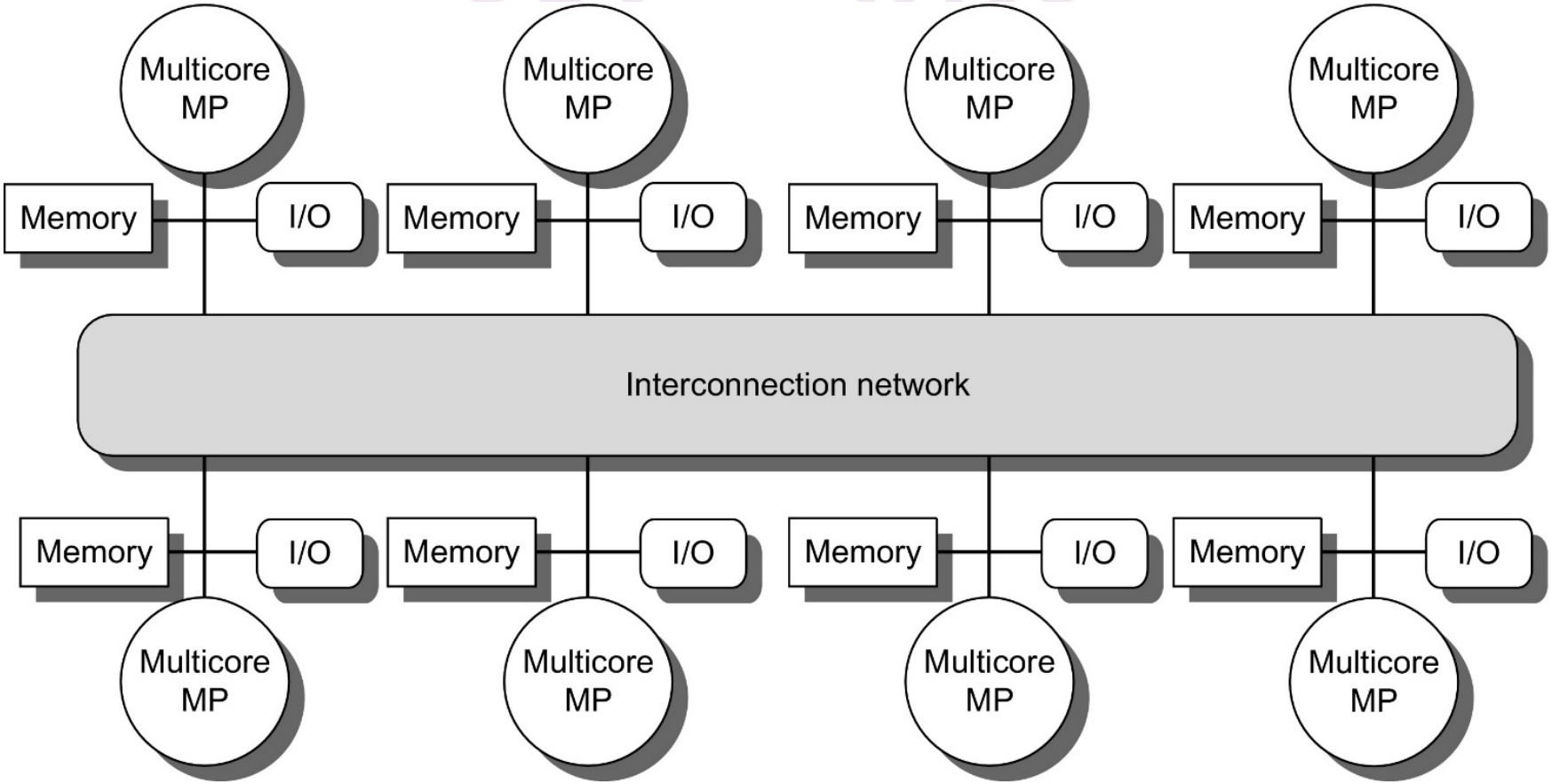
对称（集中式共享内存）多处理器(SMP)

- 一致的内存访问(uniform memory access, UMA)：延迟相同



分布式共享内存(DSM)多处理器

- 非一致的内存访问(non-uniform memory access, NUMA)：延迟不同
- 处理器通过交换(直接)或多跳(非直接)互联网络相连接



挑战——有效并行性的限制

- 例：
  - 期望使用 100 颗处理器获得加速比 80，串行计算的比例是多少？
  - 根据 Amdahl 定理，有

$$Speedup = \frac{1}{\frac{fraction_{enhanced}}{Speedup_{enhanced}} + (1 - fraction_{enhanced})}$$

$$80 = \frac{1}{\frac{fraction_{parallel}}{100} + (1 - fraction_{parallel})}$$

$$0.8 \times fraction_{parallel} + 80 \times (1 - fraction_{parallel}) = 1$$

$$fraction_{parallel} = \frac{80-1}{79.2} \approx 0.9975$$

  - 原始计算中只允许有 $1 - 99.75\% = 0.25\%$ 的串行计算

并行处理面临的挑战

- 例：
  - 某应用在拥有 100 颗处理器的多处理器中运行，假设应用可以使用 1、50 或 100 颗处理器。
  - 如果假设 95% 时间可以使用所有 100 颗处理器，那么如果希望达到加速比 80，剩余 5% 执行时间中必须使用 50 颗处理器的比例是多少？
  - 使用 Amdahl 定理，有

$$Speedup = \frac{1}{\frac{fraction_{100}}{Speedup_{100}} + \frac{fraction_{50}}{Speedup_{50}} + (1 - fraction_{100} - fraction_{50})}$$

$$80 = \frac{1}{\frac{0.95}{100} + \frac{fraction_{50}}{50} + (1 - 0.95 - fraction_{50})}$$

$$fraction_{50} \approx 0.048$$

  - 5% 总执行时间中的 4.8% 需要 50 颗处理器，串行计算只占 0.2%。

- 办法：通过开发采用新算法的软件来解决

挑战——远程访问延迟

- 例：
  - 拥有 32 颗处理器的多处理器系统访问远程存储器延迟为 $100ns$。假设除了远程访问之外所有访问都在局部存储层次中命中，远程请求时处理器停止执行，处理器时钟是 $4GHz$。假如基本 CPI（所有访问均在Cache中）为 0.5，不存在远程访问相比存在 0.2% 远程访问快多少？
  - 存在 2% 远程访问的有效 CPI

    $$CPI = CPI_{base} + rate_{remote\ request} \times cost_{remote\ request}$$
    $$= 0.5 + 0.2\% \times cost_{remote\ request}$$
  - 远程请求代价

    $$\frac{cost_{remote\ request}}{time_{cycle}} = \frac{100ns}{0.25ns} = 400cycles$$
  - 计算CPI：$CPI = 0.5 + 0.2\% \times 400 = 1.3$
  - 全部局部访问快 $1.3 \div 0.5 = 2.6$ 倍
- 办法：需要体系结构和程序员共同努力

缓存一致性

- 问题：每个处理器在自己的缓存中可能看到不同的数值

Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

- 两种状态
  - 全局状态：在主存中
  - 局部状态：在 Cache 中

两个一致性模型

- Coherence ([kəʊˈhɪərəns])模型
  - 处理器的所有读操作必须返回最近写入的数值。
  - 任意两颗处理器对相同位置的写操作必须以相同的时间顺序被其它处理器看到。
  - 关注点：对同一位置的写操作对所有处理器的可见性。

- Consistency([kənˈsɪstənsi])模型
  - 如果处理器写入位置 A 之后写入位置 B，那么看到位置 B 新数值的处理器也必须看到位置 A 的新数据。
  - 关注点：写入的数值何时能读到。

两个一致性模型的比较

- Coherence模型和Consistency模型的区别
  - 存储Coherence模型
    - 对于同一内存位置的写操作对所有处理器的可见性。
  - 存储Consistency模型
    - 不仅针对同一内存位置，而且包括读和写操作。

- Coherence模型和Consistency模型是互补的
  - Coherence 定义了读和写内存同一位置的行为
  - Consistency 定义了访问内存其它位置的读和写行为

缓存一致性的前提假设

- 所有处理器看到写结果之后，写操作完成。
- 处理器不会改变与任何其它内存访问有关的写操作顺序。
- 允许处理器改变读操作顺序
- 强迫处理器以程序顺序完成写操作

强制一致性

- 一致性缓存(coherence cache)方法
  - 迁移(migration)：数据移动
    - 减少访问远程共享数据的延迟时间
    - 降低对共享存储器的带宽要求
  - 复制(replication)：多个数据副本
    - 减少访问延迟时间
    - 降低对读共享数据的争用

- 缓存一致性协议(cache coherence protocols)
  - 实现关键：跟踪任何共享数据块状态
    - 与单处理器缓存dirty标志相似
  - 两种类型：
    - 基于目录(directory based)
    - 监听(snooping)

两种缓存一致性协议

- 基于目录协议
  - 目录：指定物理内存块的共享状态保存在一个位置
  - SMP(对称多处理器)：使用一个集中式目录，位于内存或多核的最外层 Cache 中。
  - DSM(分布式共享内存多处理器)：分布式目录比单个集中式目录复杂（后面详细介绍）。

- 监听协议
  - 每个拥有物理内存块数据副本的 Cache 都可以跟踪块的共享状态
  - SMP(对称多处理器)：
    - 可以通过广播媒介访问 Cache
    - Cache 控制器通过监听媒介来决定是否拥有总线或交换访问请求的内存块数据副本

监听一致性协议

- 特点：
  - 应用于单核微处理器
  - 缓存通过总线连接到单个共享内存组成的多处理器系统。

- 写无效协议(write invalidate protocol)：
  - 保证处理器在写入数据项之前对该数据项具有独占访问权限
  - 一旦写数据项发生，该数据项的副本均无效

- 写更新(write update)协议：又称写广播(write broadcast)协议
  - 一旦发生写数据项操作，更新所有该数据项的缓存副本。
  - 广播写操作到所有共享缓存，消耗更多带宽，故很少使用。

写无效协议

- 例：单个写回Cache，监听总线上的写无效协议

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
				0
Processor A reads X	Cache miss for X	0		0
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

  - 处理器 B 第二次读数据 X 失效时，处理器 A 负责提供数据而取消内存对该读操作的响应
  - 多个处理器同时写同一数据项的独占问题由总线裁决胜者

关键点与步骤

- 关键：使用总线或其它广播媒介执行无效操作
  - 多片多处理器：共享内存访问总线
  - 单片多核多处理器：私有Cache和共享外层Cache之间的连接总线
- 步骤
  - 处理器获得总线访问权，广播无效数据项地址；
  - 所有处理器监听总线，获得无效数据项地址；
  - 检查自己 Cache 是否包含该数据项；如果包含该数据项，则无效该数据项。
- 竞争：由总线来裁定两颗企图同时写共享数据块的处理器谁将获得总线访问权。
  - 如果写相同数据块，则胜者写操作将该数据块无效即可
  - 如果写不同数据块，则按顺序完成写无效操作
- 注意：直到获得总线访问权才算完成共享数据块的写操作

Cache 不命中情形

- 当发生 Cache 不命中时，需要定位数据项
  - 不同的 Cache 写策略带来不一样的情形
- 写透 Cache
  - 由于数据项的最新数值总是发送到内存，所以一定可以获得。
  - 写缓冲器(write buffer)的存在带来额外的复杂性，可以看作一级缓存。
- 写回 Cache
  - 数据项的最新数值可能会在私有Cache中而不在共享Cache或内存中
  - 对 Cache 不命中和写操作采用相同的监听方法
    - 每个处理器监听共享总线上的地址
    - 如果处理器发现是请求 Cache 块的脏副本，则响应该 Cache 块的读请求，导致内存（或 $L3$）访问中止。
  - 复杂性来自于在另一个处理器私有 Cache ($L1$ 或 $L2$) 中对 Cache 块的检索，比在 $L3$ 中检索要花更长时间。

状态标志

- 使用 Cache 标签的有效位(valid bit)来实现监听
  - 读操作不命中：直接依赖监听能力
  - 写操作：是否存在该数据块副本被缓存（即：数据块被共享）。
    - 如果没有，写回 Cache 就不需要在总线上广播该数据块，省时省“力（带宽）”。

- 每个 Cache 块添加一个额外状态位，跟踪该块是否共享。
  - 当写操作对象为共享数据块时，则在总线上产生无效操作，且把该数据块标记为独占(exclusive)。
  - 拥有数据块唯一副本的核称为数据块的拥有者(owner)
  - 当其它处理器再次请求该数据块时，状态由独占变成共享。

写无效协议

- 监听一致协议通过每个处理器核中的有限状态(finite-state)控制器实现。

- 三种状态（对于写回 Cache）
  - 无效(invalid)：私有 Cache 中的数据块无效
  - 共享(shared)：私有 Cache 中的数据块被共享
  - 修改(modified)：私有 Cache 中的数据块被修改，暗示独占状态
    - 对于写透 Cache，将该状态解释为独占状态

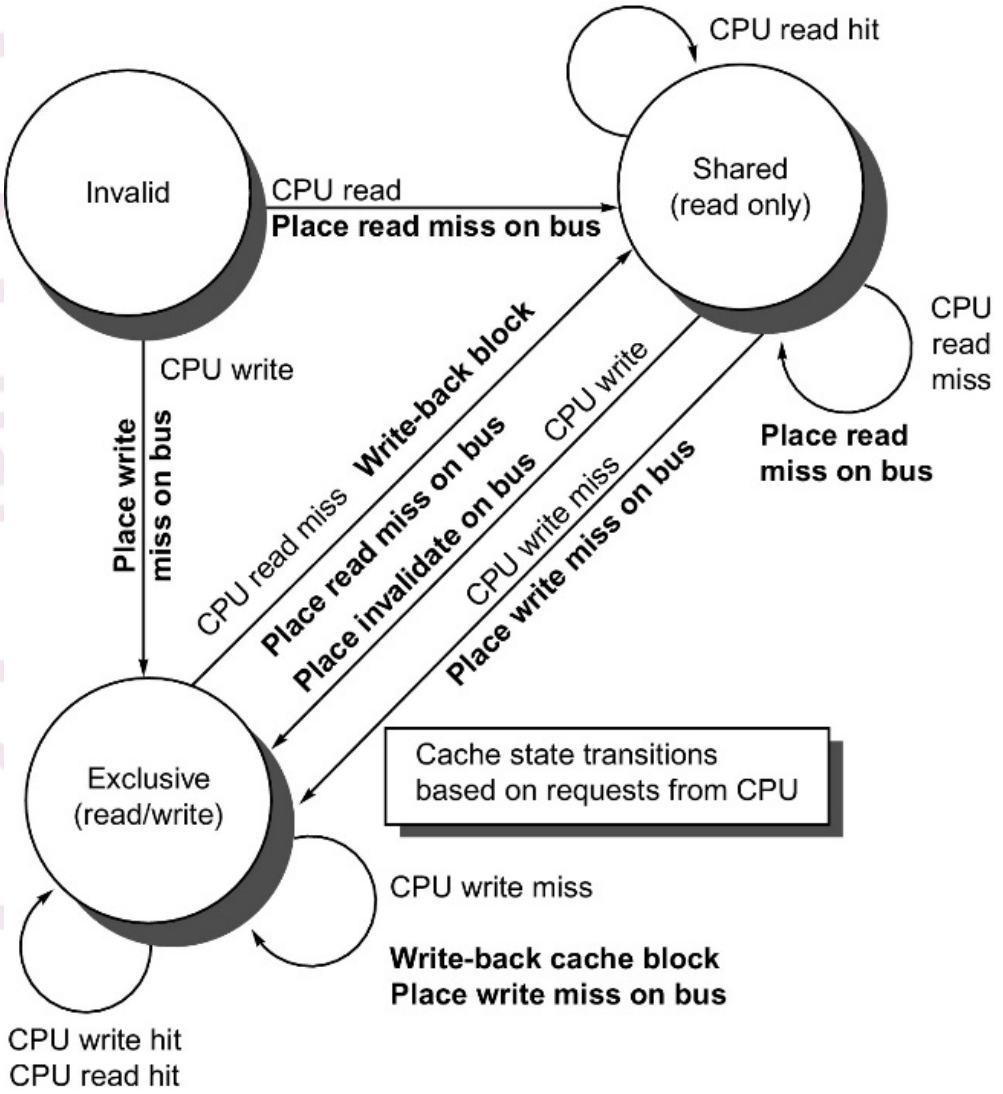
- 协议扩展
  - 独占(exclusive)：未被修改且只在一个私有 Cache 中存在的数据块。

缓存一致性机制

Request	Source	State of addressed cache block	Type of cache action	Function and explanation
Read hit	Processor	Shared or modified	Normal hit	Read data in local cache.
Read miss	Processor	Invalid	Normal miss	Place read miss on bus.
Read miss	Processor	Shared	Replacement	Address conflict miss: place read miss on bus.
Read miss	Processor	Modified	Replacement	Address conflict miss: write-back block; then place read miss on bus.
Write hit	Processor	Modified	Normal hit	Write data in local cache.
Write hit	Processor	Shared	Coherence	Place invalidate on bus. These operations are often called upgrade or *ownership* misses, because they do not fetch the data but only change the state.
Write miss	Processor	Invalid	Normal miss	Place write miss on bus.
Write miss	Processor	Shared	Replacement	Address conflict miss: place write miss on bus.
Write miss	Processor	Modified	Replacement	Address conflict miss: write-back block; then place write miss on bus.
Read miss	Bus	Shared	No action	Allow shared cache or memory to service read miss.
Read miss	Bus	Modified	Coherence	Attempt to read shared data: place cache block on bus, write-back block, and change state to shared.
Invalidate	Bus	Shared	Coherence	Attempt to write shared block; invalidate the block.
Write miss	Bus	Shared	Coherence	Attempt to write shared block; invalidate the cache block.
Write miss	Bus	Modified	Coherence	Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache.

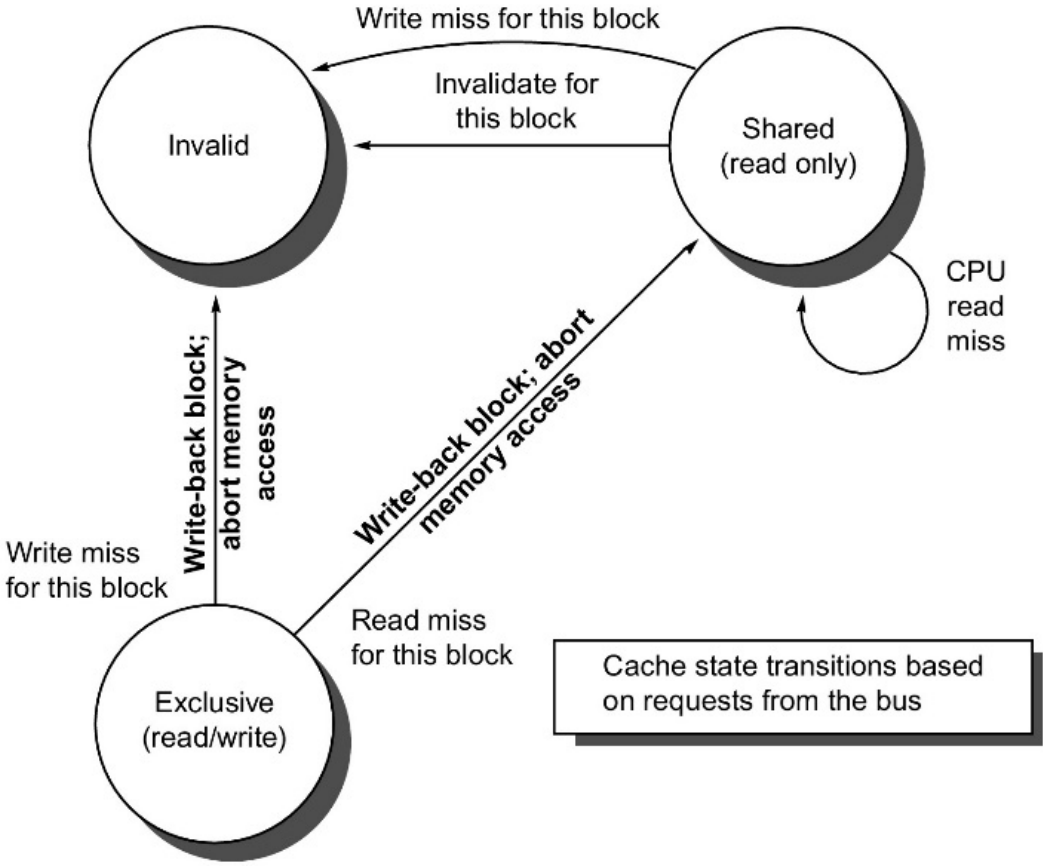
写无效协议

- 有限状态转移图（单个私有Cache块，写回Cache）
  - 圆圈内标记Cache状态
    - 圆括号内表示允许局部存储器的访问类型且不发生状态转移
  - 弧上正常字体表示引起状态改变的“激励”
  - 弧上**加粗字体**表示状态转移产生的总线动作



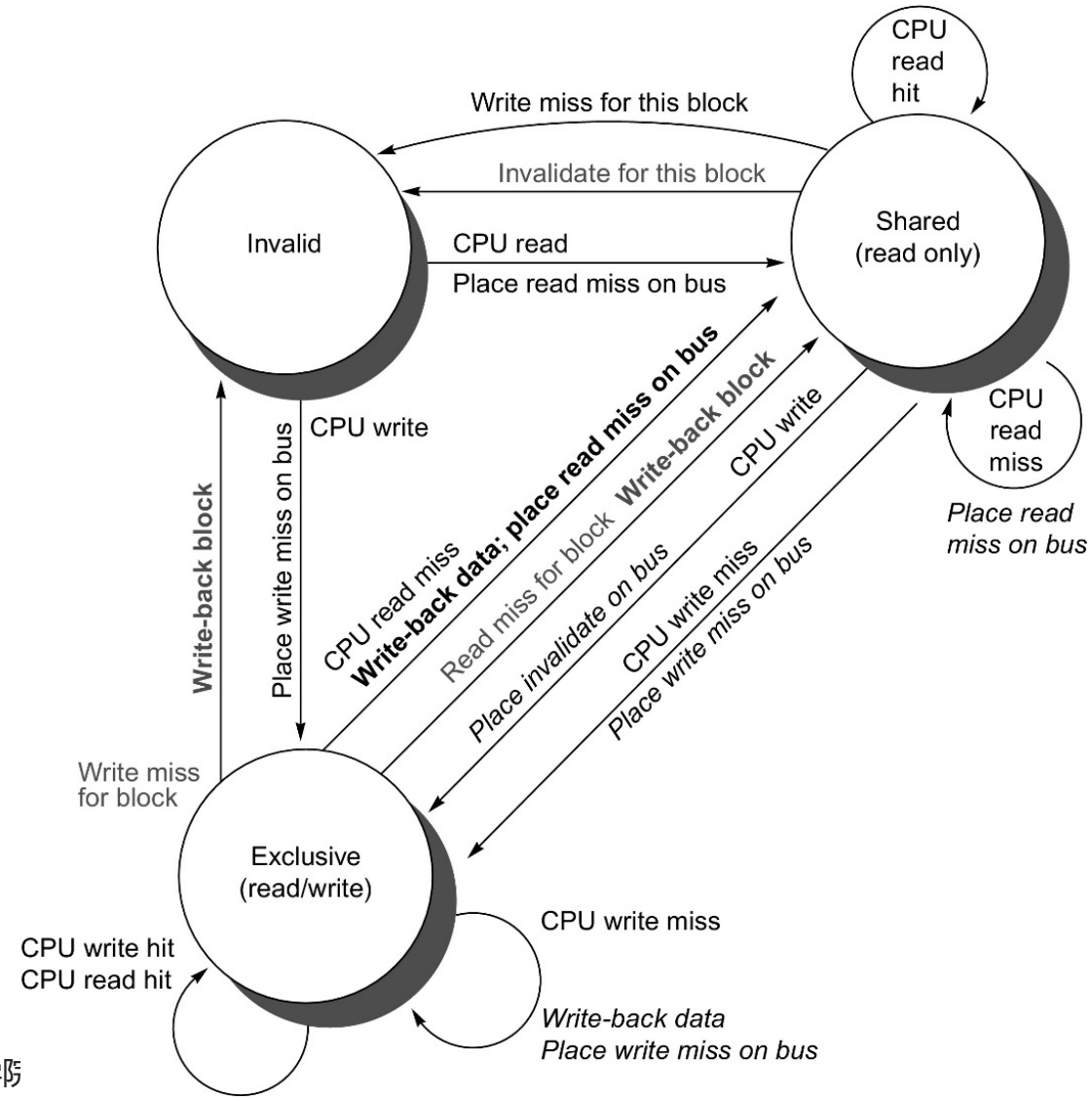
写无效协议

- 有限状态转移图（单个私有Cache块，写回Cache）（续）



写无效协议

- 缓存一致状态转移图（前面两张图的集成）



基本 MSI 协议的复杂性

- 名词：
  - 原子(atomic)操作：该操作执行期间不会执行其它操作
- **MSI**：**M**odified, **S**hared, **I**nvalid
- 操作不是原子操作
  - 例如：检测未命中，获取总线，接收响应
- 造成发生死锁和竞争的可能性
- 解决方法：发出无效操作的处理器可以保持总线访问权，直到其它处理器接收到无效操作信号为止。

基本 MSI 协议的扩展

- **MESI**：添加独占(Exclusive)状态，表示 Cache 块只在单个 Cache 中且是干净的。
  - 如果数据块处于独占状态，则对其写操作不需要生成无效操作。
  - 对处于独占状态数据块的读操作不命中，则数据块状态改变为共享状态，所有访问将被监听。

- **MOESI**：添加拥有(Owned)状态，表示数据块被该 Cache 拥有，内存中该数据块是过时的。
  - 数据块状态从修改到拥有不必写回到内存
  - 在对该数据块的读操作不命中时，数据块拥有者必须响应对该数据块的读请求操作

例

- 问题：
  - 8 颗处理器组成的多处理器系统，每颗处理器拥有自己的 $L1$ 和 $L2$ 缓存，监听在 $L2$ 之间的共享总线上完成。
  - 假设：无论一致性不命中还是其它原因不命中的 $L2$ 请求均需要 15 个时钟周期；时钟频率 $3GHz, CPI = 0.7$，load/store 操作频度为 40%。
  - 问：如果不超过 50% 的 $L2$ 带宽用于一致性通信，则每颗处理器的最大一致性不命中率是多少?
- 解答：
  - 能使用的 Cache 周期数（CMR表示一致性不命中率）

$$Cycles_{cache} = \frac{Clock\ rate}{Cycles_{per\ request} \div 0.5} = \frac{3.0GHz}{30} = 0.1 \times 10^9$$

$$Cycles_{cache} = Memory\ Reference_{per\ clock\ per\ processor} \times rate_{clock} \times Num_{processor} \times CMR$$

例

$$Cycles_{cache} = \frac{0.4}{0.7} \times 3.0GHz \times 8 \times CMR = 13.7 \times 10^9 \times CMR$$

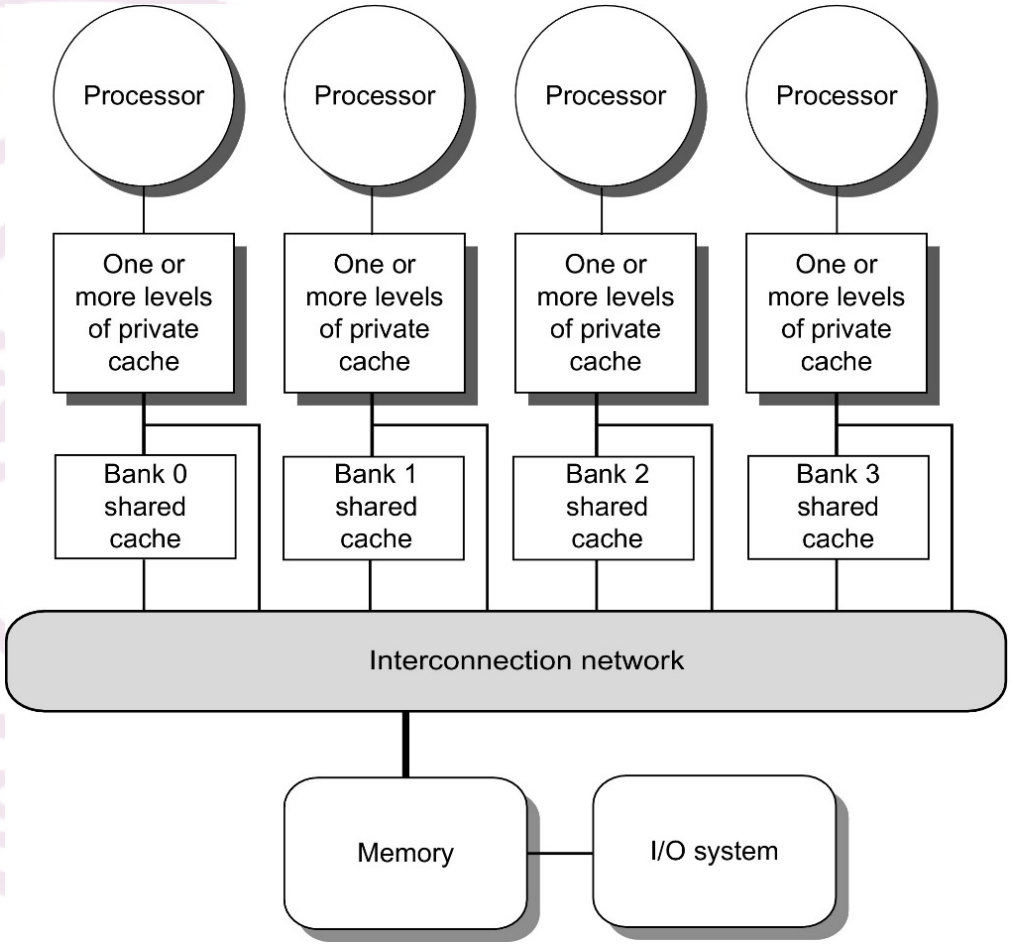
$$CMR = \frac{0.1}{13.7} \approx 0.0073 = 0.73\%$$

- 结论：一致性不命中率不能超过 0.73%。

- 问题：如果期望 $CMR \approx 1\%$，那么最多可以多少颗处理器？
  - $\frac{0.4}{0.7} \times 3.0GHz \times PC \times 1\% = \frac{3.0GHz}{30} \Rightarrow PC = \frac{0.7}{0.4 \times 0.01 \times 30} \approx 5.8$

SMP架构的扩展瓶颈：共享内存总线或监听带宽

- 增加带宽方法
  - 标签复制方法将使有效缓存监听带宽加倍。
    - Cache访问和监听访问分离
  - 如果最外层为共享缓存，则可将它们分布到每个处理器中，使得每个处理器处理它所管理存储空间的监听任务（见右图）
    - 造成 NUCA(Non-Uniform Cache Access)
  - 将目录放在最外层共享缓存中（如 $L3$），它将充当监听请求的过滤器，且必须是包容的。



监听一致协议

- 支持超过 8 核多处理器系统在使用互联网络实现监听一致协议时所面临的挑战：
  - 顺序化变得困难
  - 写操作未命中不是原子操作
  - 处理器如何知道所有无效操作何时完成?
  - 当两个处理器同时写操作时，如何解决争用问题?
- 解法：每个存储块只与单个总线相关联，确保两个访问相同数据块的企图必须通过公共总线串行化。

性能

- Cache 性能
  - 单处理器 Cache 未命中流量
  - 失效操作和后续 Cache 未命中流量

- 一致性未命中(coherence miss)：由处理器间通信引起的未命中
  - 真实共享不命中(true sharing misses)：通过 Cache 一致性机制由数据通信引起的未命中
  - 虚假共享不命中(false sharing misses)：使用每个缓存块单个有效位的失效一致性算法引起的未命中

一致性不命中

- 真实共享不命中
  - 处理器第一次对共享 Cache 块写操作引起的不命中操作
  - 另一个处理器对该 Cache 块中被修改数据字读操作未命中所引起的数据块传输
- 虚假共享不命中
  - 对数据块中某个数据字进行写操作（且后续访问会导致不命中）而引起不命中操作
  - 如果写操作和读操作的数据字不同，无效操作不会引起新数值的传输，只能引起额外的 Cache 未命中

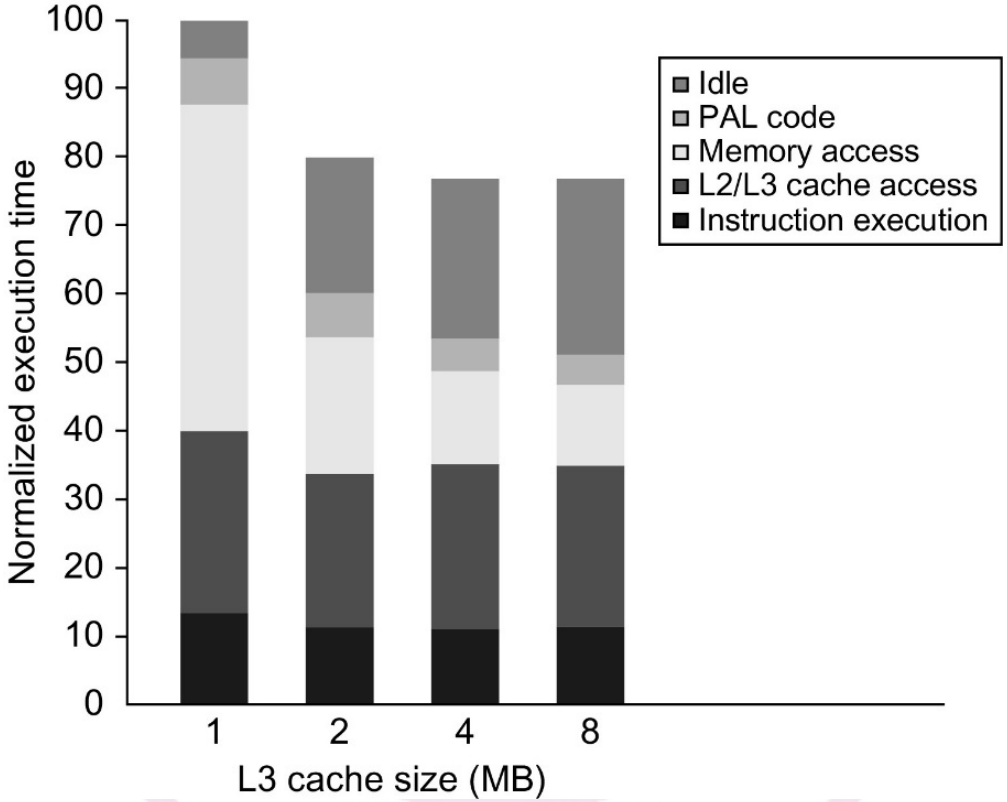
性能：联机事务处理工作量

- OnLine Transaction Processing, OLTP
  - 环境：4 颗处理器的 AlphaServer 4100系统（CPU为Alpha 21164）
  - 目的：多处理器 Cache 活动，特别是 $L3$ 行为，多为一致性相关流量
  - Alpha 21164 和 Intel i7 的 Cache 层次特性

Cache level	Characteristic	Alpha 21164	Intel i7
L1	Size	8 KB I/8 KB D	32 KB I/32 KB D
	Associativity	Direct-mapped	8-way I/8-way D
	Block size	32 B	64 B
	Miss penalty	7	10
L2	Size	96 KB	256 KB
	Associativity	3-way	8-way
	Block size	32 B	64 B
	Miss penalty	21	35
L3	Size	2 MiB (total 8 MiB unshared)	2 MiB per core (8 MiB total shared)
	Associativity	Direct-mapped	16-way
	Block size	64 B	64 B
	Miss penalty	80	~100

性能

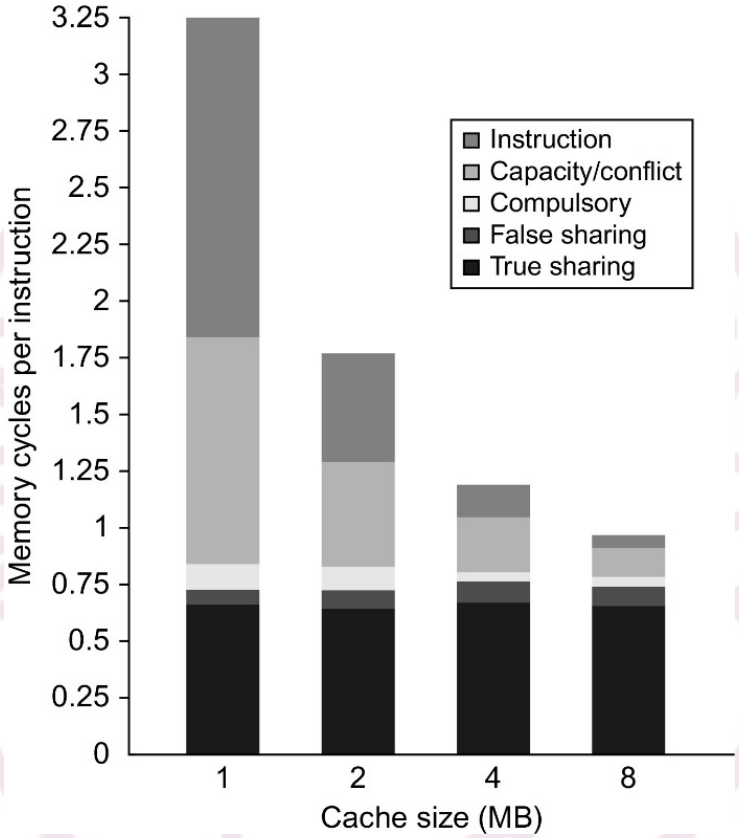
- 随着 $L3$ 容量增加，OLTP工作量的相对性能



  - 随着 Cache 容量增加，空闲时间也增加，减少了一些性能提升
  - 减少了 $L3$ 未命中
  - 几乎所有提升都发生在 $1\sim2\ MiB$，为什么?

性能

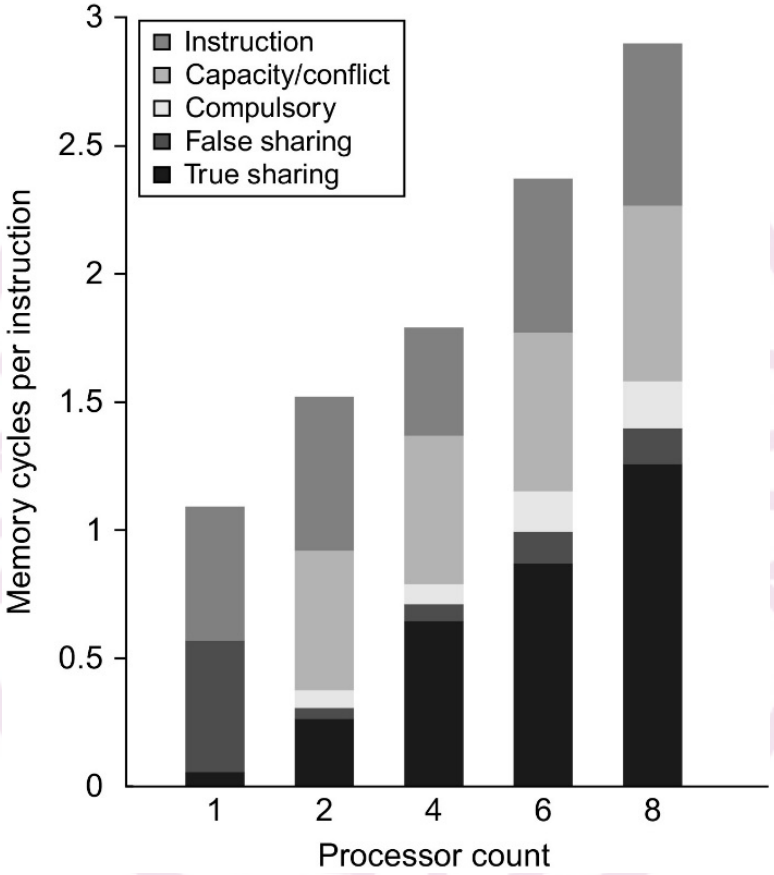
- 随着 Cache 容量增加，内存访问周期偏移的原因



  - 随着容量增加，指令和容量/冲突不命中转化为次要因素
  - 强迫未命中、虚假共享不命中和真实共享不命中基本不受影响
  - 容量增加消除单处理器绝大多数不命中，那么增加处理器数目会怎样?

性能

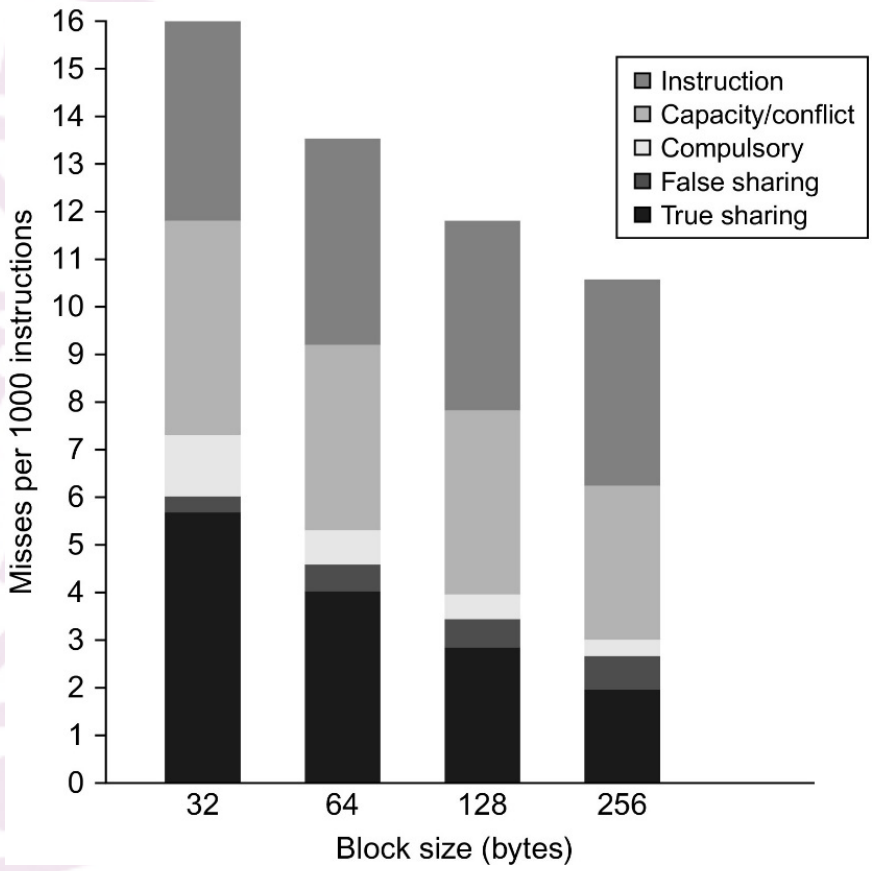
- 随着处理器数目增加，存储器访问周期的贡献



  - 处理器数目增加，增加了真实共享不命中率（单处理器情形不体现）
  - 强制不命中稍微增加

性能

- 随着 $L3$ 数据块容量增加，每千条指令不命中数稳步下降
  - 真实共享未命中率减小，表示在真实共享模式中存在局部性；
  - 强制性/冷启动未命中率明显地下降，如预期；
  - 冲突/容量未命中率较小地减少，表示空间局部性不是很高；
  - 虚假共享未命中率虽然绝对数值较小，但接近两倍。



诚信 创新 实践