Cache Coherence

Cache Coherence

Shared Memory Model 是指许多并行的程序通过共享内存来交流

cache coherence 问题即是多个处理器缓存了同一个 cache block 时，如何确保其中的数据是一致的

- 基于软件的解决方法
  - 在 cache 对软件不可见的情况下保持一致性
  - ISA 提供 flush 指令
  - 使得架构设计简单，但是编程复杂
- 基于硬件的解决方法
  - Straw-man Approach：所有处理器共享 cache
    - 不需要 cache coherence
    - 共享 cache 会成为瓶颈
    - 系统难以扩展
  - Write propagation：确保对 cache 的更新会传播
  - Write serialization：维护一个所有处理器可见的全局写序列

Update vs. Invalidate

有两种解决方案

- update：对所有副本 push 一个更新
- invalidate：确保全局只有一个副本

读的时候如果本地的副本是无效的，则发出 request，如果其他节点有有效的副本则返回，否则由内存返回

写的时候将 cache block 读入

- 如果是 update，写 block，同时将写的数据和地址广播出去，其余节点如果有副本则更新
- 如果是 invalidate，写 block，同时将地址广播，其余节点如果有副本则让其无效

tradeoff 取决于写的频率

- update

    - 如果共享的数据集固定且更新不频繁，代价较小
    - 如果写之间没有读，则广播的 update 是无用的
    - 总线会成为瓶颈
- invalidate
    - 写的处理器独占访问权，且只有更新之后要读的处理器会再去获取有效的副本
    - 写频繁的话会导致频繁地 invalidation-reacquire

有两种解决 cache coherence 的方案

- Snoopy Bus
    - 基于总线，将所有访存请求单点串行化
    - 处理器观察其余处理器的行为
- Directory
    - 对每个 block 进行单点串行化，分布式
    - 处理器显式请求 block，由 directory 记录哪些 cache 有对应 block
    - directory 来调度 invalidate 和 update

Snoopy Cache Coherence

基本思路是

- 所有 cache 监听其他的 cache 的读写，确保一致性
- 每个 cache 有 coherence metadata
- tag 是双口的（一端连处理器，一端连总线）

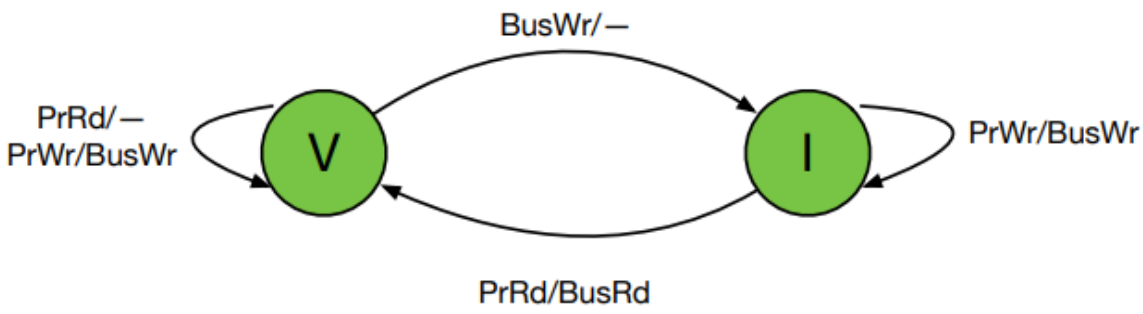
每个处理器与 cache 相连，cache 与内存总线相连，总线上还有主存和 DMA 等。cache 将读写请求在总线上广播

适用于小规模的多处理器

Simple VI Protocol

cache 两个状态，Valid 和 Invalid

write through，no-write-allocation 的 cache，FSM 如下

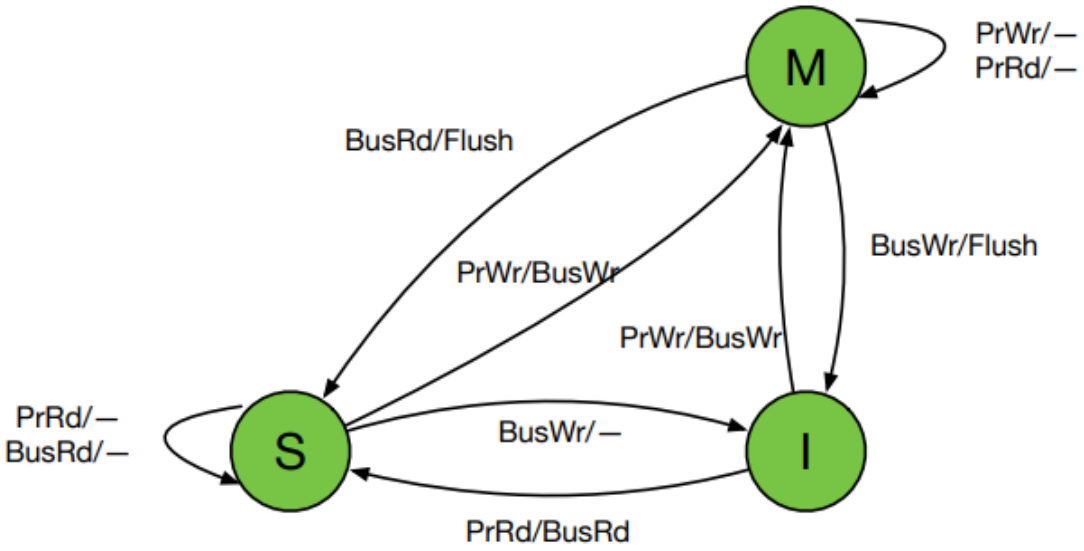


MSI Protocol

三个状态

- Modified：当前 block 是唯一的副本，且被修改了
- Shared：当前 block 是多个副本之一
- Invalid：当前 block 无效

FSM 如下



相比 VI，可以实现 write back 策略

MESI Protocol

对于 MSI 来说，如果一个处理器读写 cache，仍需要在总线上发送请求，浪费带宽

MESI 有四个状态

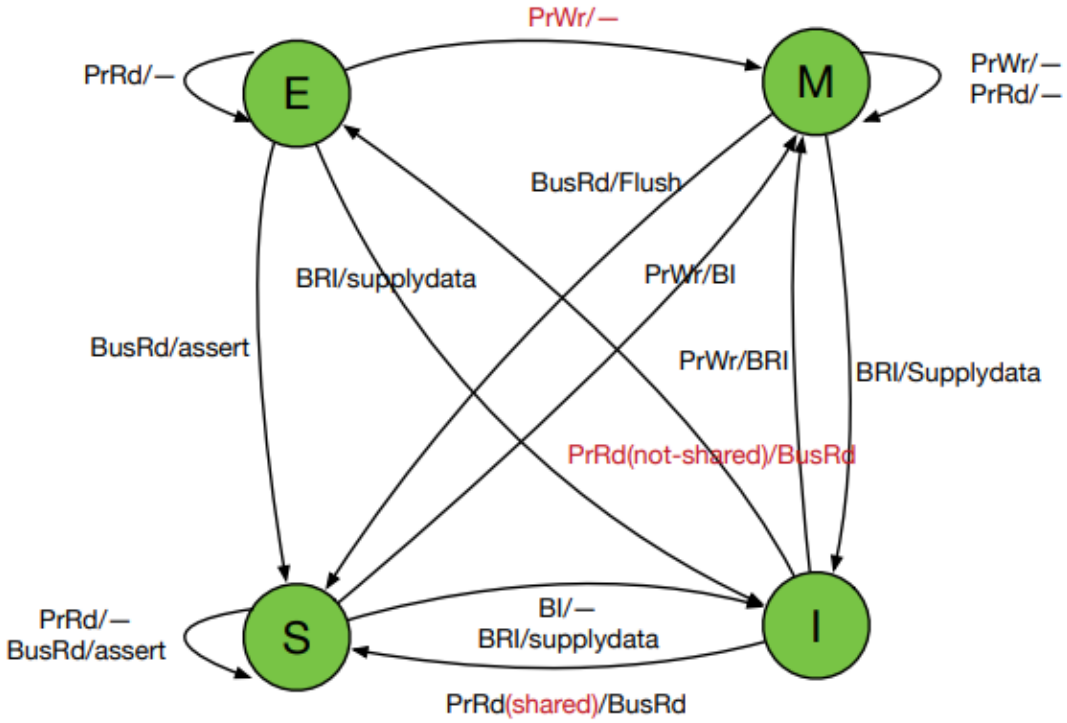
- Modified：当前 cache line 是唯一副本，且被修改
- Exclusive：当前 cache line 是唯一副本，且未被修改
- Shared：当前 cache line 是多个副本之一
- Invalid：当前 cache line 无效

> 如何区分 shared 和 exclusive：线或，如果其他 cache 有这个数据，则 assert 一下

将 BusWr 变为两个指令

- BI：只是 invalidate，不需要数据
- BRI：既 invalidate 也需要数据

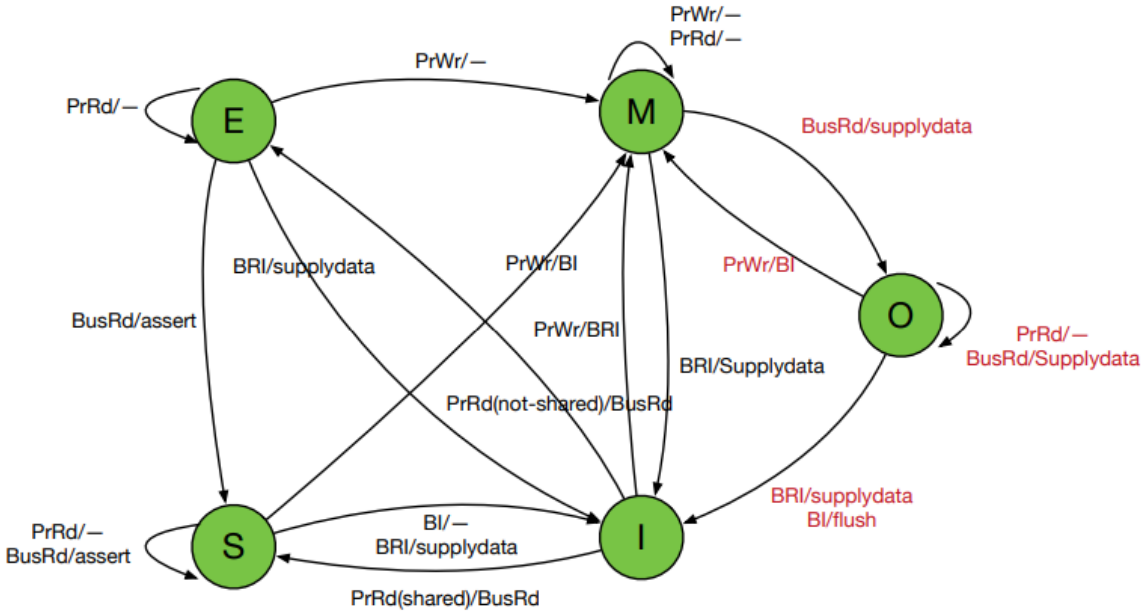
FSM 如下



MOSEI Protocol

当 block 状态为 M 时，如果收到 BusRd，需要将数据写回内存，延时大

可以将数据在 cache 间传送，指定许多共享副本中的其中一个为 Owner，在换出时将 block 写回内存。FSM 如下



False Sharing

当一个 cache block 有多个 word 时，cache coherence 是在 block 层面而不是 word 层面

coherency miss 可以分为两种

- True sharing misses：由 cache coherence 的机制产生：CPU 写一个 block 时会将其他 cache 中的该 block 无效
  - 如果 block 大小为 1 个 word，仍会发生
- False sharing misses：当 cache 因为要读的 word 以外的 word 被写而无效时产生的 miss
  - 没有传递新的值，只是 cache miss
  - 当 block 大小为 1 个 word 时不会发生

当增大 cache size 时，只会减少单处理器时的 cache miss，不影响 sharing miss

当 CPU 增多，sharing miss 增加

Directory Cache Coherence

Snoopy 每次发生 cache miss 都要广播，需要很大的总线带宽和很大的 cache 带宽

Directory：将内存分块，为每个 block 分配一个 directory，使用点对点信息维护一致性

NUMA

NUMA (Non-Uniform Memory Access)：访问内存的延时根据访存的节点和访存地址不同

为每个处理器分配一个最近的 directory

确定 directory 可以根据地址的低位或者高位

- 高位决定
  - OS 可以控制 NUMA 的内存分布（e. g. 将一个进程的地址空间映射到运行该进程的处理器的 directory）
  - 负载不均衡
- 低位决定
  - OS 不能控制分布
  - 负载均衡

Directory

directory 中的 cache line 有三种状态

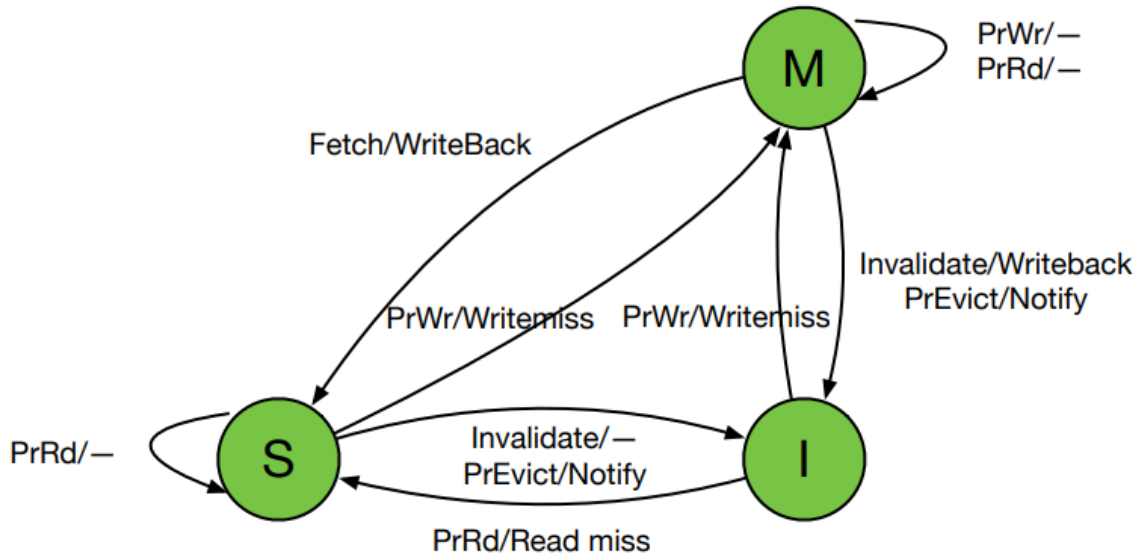
- Shared
- Uncached
- Exclusive

可以使用位向量存储 shared 信息，每个节点一个 bit

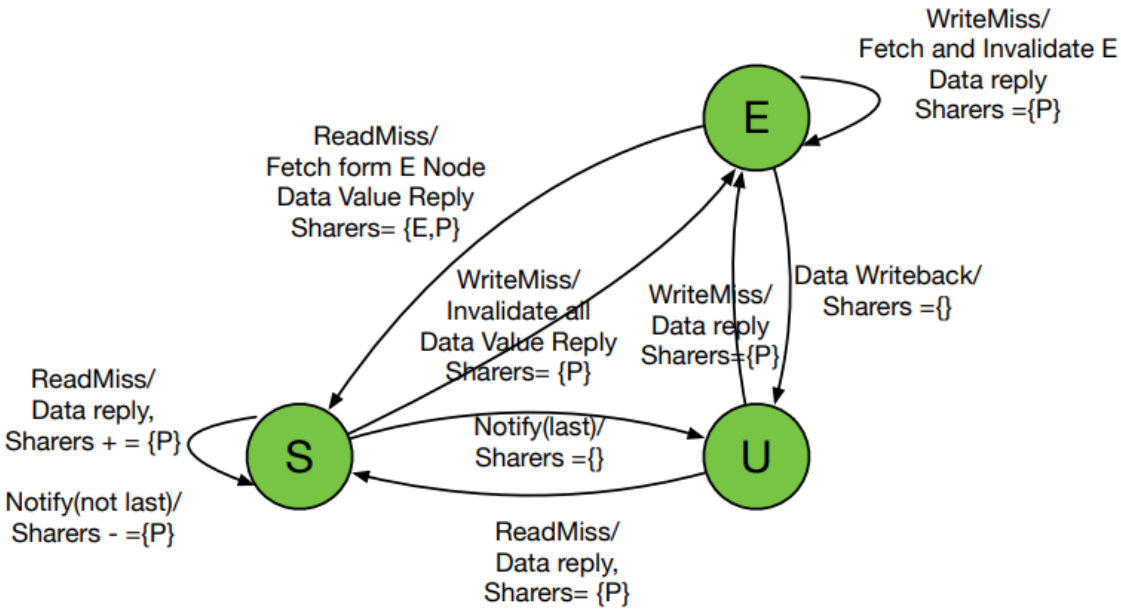
或者存储 cache line 的二进制编号，但如果列表长度有限的话可能会 overflow

Exclusive 状态只存储唯一的节点

cache block 的 FSM 为



directory 的 FSM 为



Issues in Implementing Directory Coherence

需要多条逻辑上的信道，并对不同的信号分离。操作的 queuing

避免死锁

确保状态的转移具有原子性

确保系统无死锁

Snoopy vs. Directory

Snoopy

- critical path 较短，请求通过总线直接传输到内存
- 全局串行化简单：bus 直接实现
- 简单
- bus 难以扩展

Directory

- critical path 长：请求-directory-内存
- 需要额外的空间存储 shared set（可以使用 bloom filter）
- 控制更复杂
- 不需要对所有 cache 广播
- 可扩展性好