Computerarchitectuur

H&P Ch 5. Thread-Level Parallelism

Kristian Rietveld

http://ca.liacs.nl/

Universiteit Leiden
The Netherlands

Thread-Level Parallelism

- In het geval van thread-level parallelism (TLP) gaan we uit van meerdere *threads*.
  - Er zijn dus meerdere program counters binnen 1 programma, of meerdere programma's actief.
  - In geval van threads die onafhankelijk van elkaar moeten opereren, is data-level parallelism geen oplossing.

Wat betreft hoofdstuk 5 in het boek, we leggen de nadruk op de opzet van multicores & cache coherence. Ruim de helft slaan we over.

Multiprocessor computer

Als we het hebben over een *multiprocessor computer*, wat bedoelen we dan?

- Één computer, dat één operating system (OS) draait.
- De computer bevat meerdere processoren.
  - Meerdere cores op één chip, meerdere chips of een combinatie van beide.
- Een gedeeld geheugen, benaderbaar door alle processoren.
- MIMD model.

TLP op multiprocessors

Je kunt twee vormen van TLP op multiprocessors onderscheiden:

- *Parallel processing*: nauw verbonden threads die samen werken aan 1 dezelfde taak en daarvoor het werk hebben verdeeld.
- *Multiprogramming*: meerdere programma's die onafhankelijk van elkaar in uitvoering zijn.
  - Of een enkel programma dat op meerdere processoren aan meerdere verzoeken tegelijkertijd werkt. Denk aan DBMS, web-server (request-level parallelism: RLP).

Dit onderscheid wordt ook wel verwoord als *data parallelism* vs. *task parallelism*.

Multiprocessor vs. Cluster computer

Multiprocessor: altijd 1 computer, 1 OS

- Single-chip multi cores (eventueel met threading)
- Multiple-chip single cores
- Multiple-chip multi cores

Cluster computer

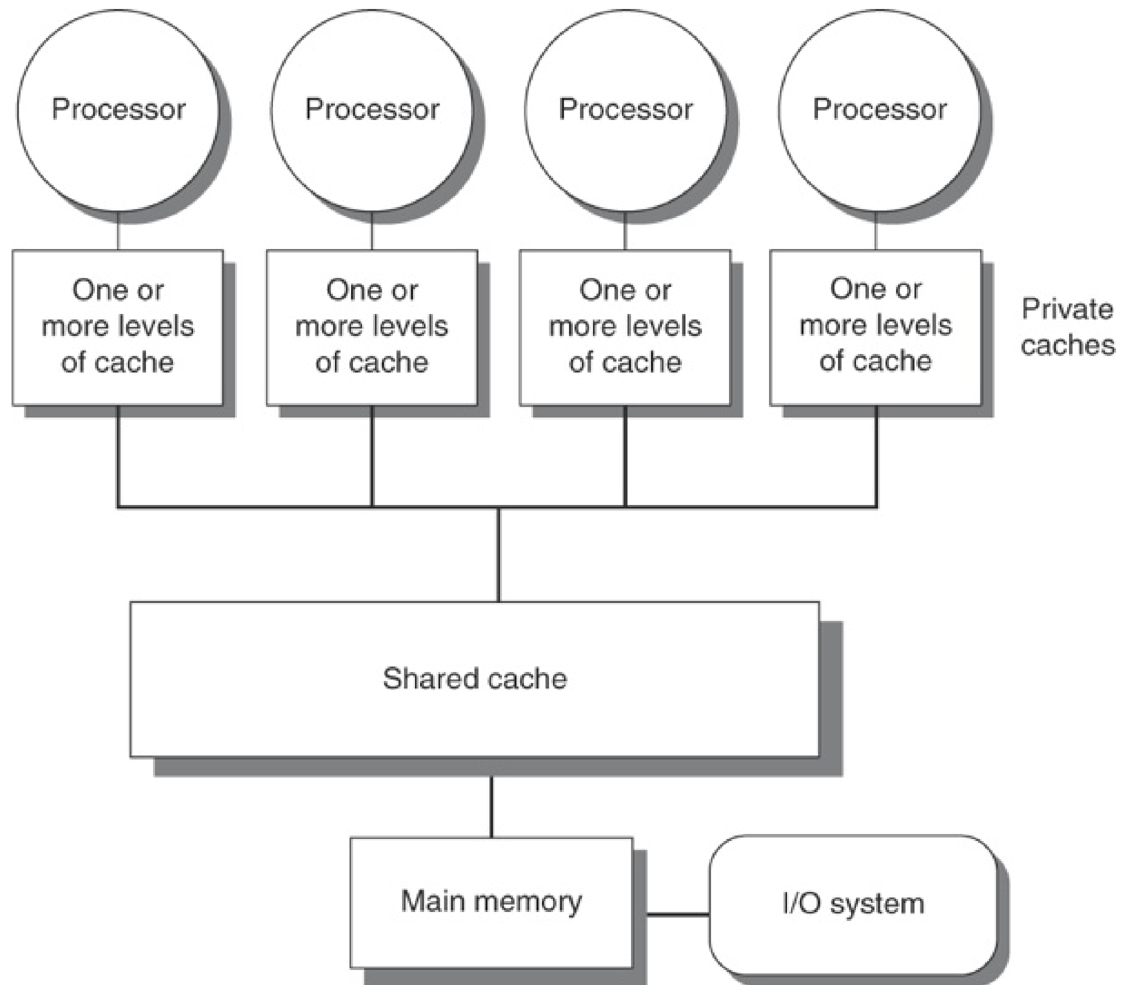
- Bestaat juist uit meerdere multiprocessor computers.
- Elke computer wordt vaak een *node* genoemd.
- Processoren in de ene *node* kunnen niet direct bij het geheugen van een andere *node*.
- De nodes zijn met elkaar verbonden met een zeer snel interconnctie netwerk, dit wordt gebruikt om data uit te wisselen uit de geheugens van de verschillende nodes.

Shared memory systemen

Symmetric (shared-memory) multiprocessor (SMP)

- Ook wel: Centralized shared-memory multiprocessor.
- Systeem met 1 gedeeld geheugen, gebruikelijk < 8 cores.
- UMA: Uniform Memory Access, de geheugenlatency is even groot voor elke core.

Shared memory multiprocessor



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 5.1.

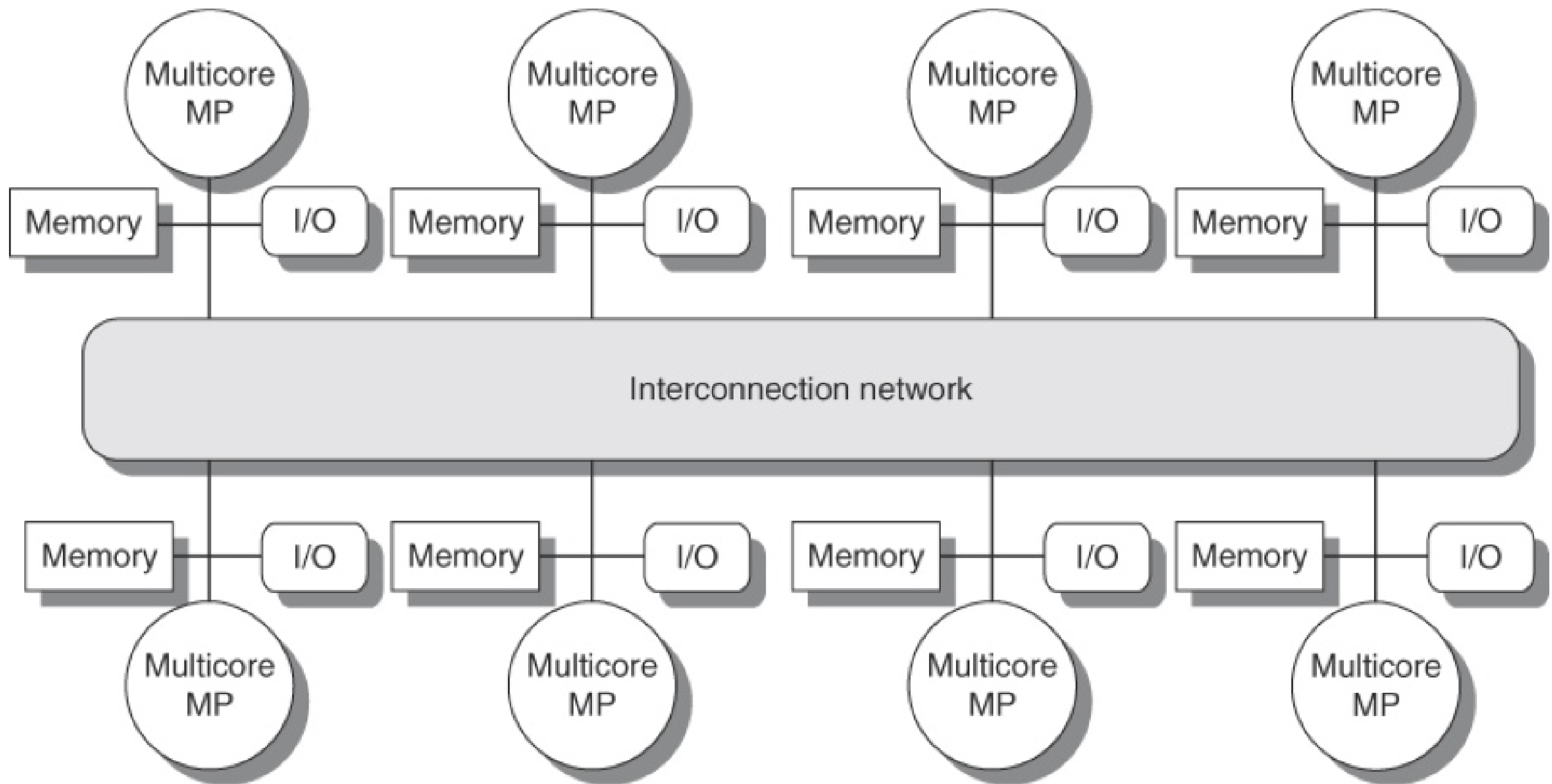
Distributed-memory systemen

Distributed shared-memory multiprocessor (DSM)

- Nog steeds één computer, maar het geheugen is gedistribueerd.
- Voor grotere aantallen cores kan een enkel centraal geheugen niet de vereiste bandbreedte leveren.
- Alle processoren kunnen nog wel al het geheugen direct benaderen.
- NUMA: Non-Uniform Memory Access
  - Memory access time is afhankelijk van in welk geheugen de gevraagde data zich bevindt.

(Contrasteer cluster computers: ook een gedistribueerd geheugen, maar processoren hebben niet direct toegang tot geheugens van andere nodes. Hier wordt message-passing toegepast via het interconnctie netwerk).

Distributed memory multiprocessor



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 5.2.

Uitdagingen

Bij het ontwikkelen van software voor multiprocessors zijn er twee grote uitdagingen.

(1) Bevat het programma wel voldoende parallelisme?

Rekensom aan de hand van Amdahl's Law. Stel we willen een speedup van 80 halen met 100 processoren. Hoe groot gedeelte van het originele programma mag dan sequentieel zijn?

Vereenvoudiging: of alle processors worden gebruikt, of maar 1.

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

$$80 = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{100}}$$

$$0.8 \times Fraction_{enhanced} + 80 \times (1 - Fraction_{enhanced}) = 1$$

$$Fraction_{enhanced} = 0.9975$$

0.25% !!!

Uitdagingen (2)

(2) Kosten van communicatie

Vele parallele algoritmen vereisen communicatie. Communicatie wordt duurder naarmate cores verder uitelkaar liggen:

- Cores op zelfde chip: ~35 tot 50 cycles
- Cores op verschillende chips: > 100 cycles
- Cores in verschillende nodes: > 500 cycles

Dit geeft al significante verschillen wanneer < 1 % van de instructies remote communicatie betreft.

Caching in shared-memory computers

- In een shared-memory computer komen we private data (gebruikt door maar één processor) en shared data tegen.
- Het is gebruikelijk dat zowel private als shared data wordt gecached.
- Processoren kunnen communiceren via shared data.
- We zagen al dat elke processor ook een eigen, privé cache heeft.
  - *Kan hier een probleem ontstaan en zo ja wat?*

Cache coherence

- Er is sprake van meerdere caches in het systeem. In welke cache wordt shared data opgeslagen?
  - In maar 1 van de caches?
  - Of in meerdere? Wat is dan de meest recente kopie? Of worden alles caches up-to-date gehouden?
- Dit is het cache coherence probleem.
  - Het kan voorkomen dat twee processoren twee verschillende waarden zien voor één en dezelfde geheugenlocatie.

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

Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 5.3

Cache coherence (2)

- *Coherence*: welke waarde kan door een leesactie worden opgeleverd?
- *Consistency*: wanneer wordt een geschreven waarde zichtbaar voor een leesactie?

- Een en ander kan heel formeel worden vastgelegd.
- Wat we willen voor *coherence* is dat een read van een bepaalde locatie altijd de meest recente waarde oplevert.

Cache coherence (3)

Formeler: wanneer is een memory systeem coherent?

- Een schrijfactie van processor P naar X die wordt gevolgd door een leesactie van P van X, zonder dat er tussentijds door een andere processor naar X wordt geschreven, levert altijd de waarde geschreven door P op.
  - *Hier wordt eigenlijk normale program order vastgelegd die ook in uniprocessor systemen moet gelden.*
- Een leesactie van een processor van X die volgt op een schrijfactie naar X door een andere processor levert de geschreven waarde op wanneer er voldoende tijd tussen de twee acties zit en er vinden tussentijds geen andere schrijfacties naar X plaats.
  - Voldoende tijd? Hoeveel tijd precies? Wanneer een geschreven waarde precies gezien moet worden door alle lezers moet worden vastgelegd in een *memory consistency model.*

Cache coherence (4)

- Schrijfacties naar dezelfde locatie zijn serialized: twee schrijfacties naar dezelfde locatie door verschillende processoren worden altijd in dezelfde volgorde gezien door alle processoren.
  - Wanneer dit niet gebeurt, kan het voorkomen dat een processor eerst de schrijfactie van P2 ziet, er daarna die van P1 waardoor de waarde voor X in die processor altijd op die van P1 blijft staan.
  - We noemen dit *write serialization*.

Voor nu om het eenvoudig te houden nemen we aan dat:

- Een write is pas compleet wanneer alle processoren op de hoogte zijn gebracht van deze write. Pas hierna kan de volgende write plaatsvinden.
- De processor past de volgorde waarin writes worden gedaan niet aan.

Cache coherence protocol

- Voor programma's die op meerdere processoren draaien zullen we hun data terugvinden in meerdere caches.
  - Er bestaan dus kopiën.
  - De caches hebben de mogelijkheid om deze data te migreren naar een andere cache en ook de repliceren uit een andere cache.
- We moeten er wel voor zorgen dat de caches coherent blijven. Hiervoor ontwerpen we een *cache coherence protocol* dat in de hardware wordt geïmplementeerd.
  - (Met andere woorden: dit is niet het probleem van de programmeur. Waarom niet? Caches zijn transparant).

Snooping

- In een eenvoudige multiprocessor zijn meerdere processoren verbonden met een gedeelde geheugenbus.
- Elke processor kan dus alle geheugenrequests op de bus zien.
  - Elke processor kijkt via "snooping" op de bus of er een geheugenrequest langs komt voor een blok dat in zijn/haar cache aanwezig is.
  - Zo ja, dan kan er tot actie worden overgegaan.

- In feite is het idee hier dat elke cache zelf verantwoordelijk is om de status van elk cache blok in de gaten te houden.
  - Is de data in het blok nog wel up-to-date?
  - Wordt de data in het blok gedeeld met anderen?

Snooping (2)

- Een manier om cache coherence te implementeren is om ervoor te zorgen dat wanneer een processor gaat schrijven naar een locatie, het exclusieve toegang heeft tot die locatie.
  - De locatie wordt dus niet gedeeld met anderen; er bestaan geen andere kopiën van deze data.
  - *Write invalidation protocol*: bij een write worden alle andere kopiën ongeldig verklaard.

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

Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 5.4.

Snooping (3)

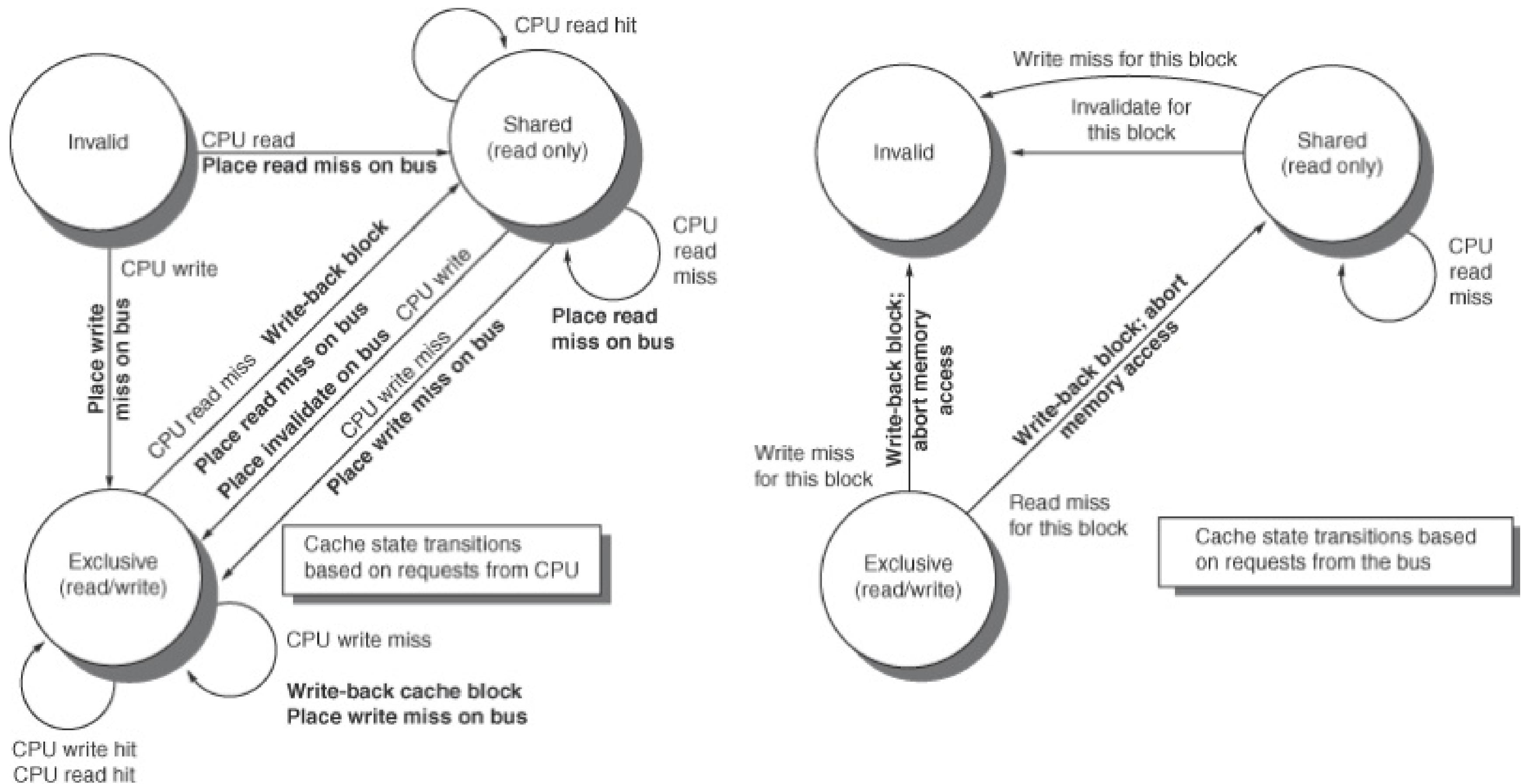
- Als er twee processoren tegelijkertijd proberen te schrijven zal er 1 winnen.
  - De andere zal de data opnieuw uit het geheugen moeten laden en dan opnieuw pogen te schrijven. Dit enforceert write serialization.
  - Hoe zit dat met het laden van de data?
    - Write-through cache: meest recente waarde altijd in het geheugen te vinden.
    - Write-back cache: cache moet ook snooping doen voor read acties van andere processoren.
- Een alternatief voor write invalidation is *write update* of *write broadcast*.
  - Kost veel meer bandbreedte. Daarom wordt er veel meer gebruik gemaakt van write invalidation.

Snooping (4)

- Een snooping cache coherence protocol wordt vaak geïmplementeerd als eindige automaat.
  - De automaat acteert op verzoeken van de processor en op verkeer op de shared memory bus.
  - De toestand is steeds *per cache blok* opgeslagen.

- We bekijken een voorbeeld waarin er 3 toestanden bestaan:
  - *Invalid*: cache blok is invalid.
  - *Shared*: cache blok is mogelijk gedeeld.
  - *Modified*: cache blok is aangepast (dirty); impliceert dat cache blok *exclusive* is.

Snooping (5)



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 5.6.

Opmerking: automaat uitgesplitst voor overzichtelijkheid; normaal is er sprake van 1 eindige automaat.

Uitbreidingen Snooping

We hebben zojuist een MSI protocol gezien (Modified, Shared, Invalid). ("modified" was in het voorbeeld "exclusive").

- **MESI** voegt een *Exclusive unmodifed* state toe, deze wordt gebruikt wanneer data in 1 enkele cache beschikbaar is en niet dirty. In dit geval kan er naar worden geschreven zonder invalidate op de bus te zetten.
  - (Core i7 gebruikt een variant van MESI: MESIF).
- **MOESI** voegt een *Owned* state toe. Deze duidt aan dat een cache blok in die bepaalde cache staat en niet up-to-date is in het geheugen. In MOESI kun je van de modified naar owned state zonder het blok naar geheugen te schrijven.
  - Als je bij MSI/MESI het aangepaste block wilt delen, moet je van modified naar shared en een write-back doen.

Limitaties SMP & Snooping

- Wanneer het aantal cores groter wordt dan 8, wordt gedeeld geheugen een bottleneck.
- Snooping vereist een grote bandbreedte in de caches; elke cache moet elke miss die op de bus wordt geplaatst bekijken.
  - Zonder centrale bus moet elke miss worden gebroadcast naar de andere caches: zeer veel dataverkeer.
  - Merk op: een grotere cache verandert niets aan de hoeveelheid van dit soort dataverkeer.

Distributed Shared-Memory Multiprocessors

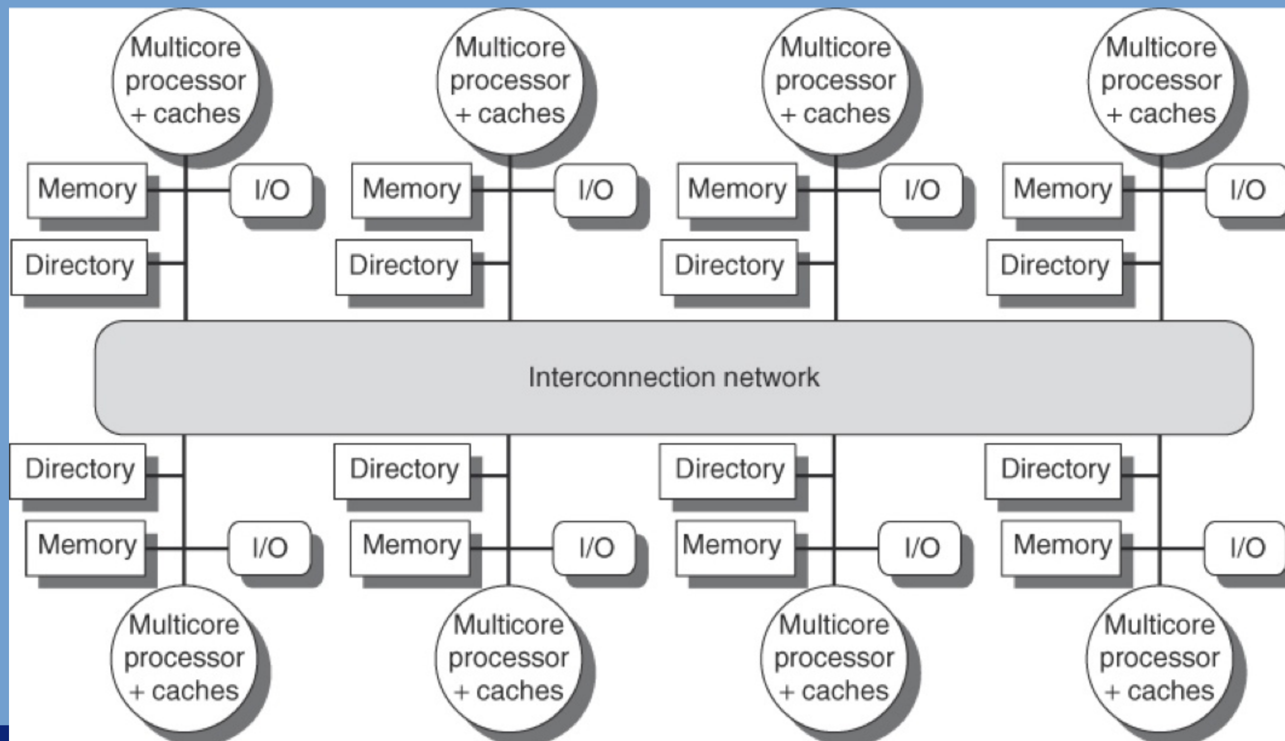
- De bandbreedte die is benodigd voor grote multiprocessors is dermate hoog dat dit niet met een gedeelde bus kan worden opgelost.
- Men is daarom uitgeweken naar gedistribueerd geheugen binnen een enkele computer.
  - Er ontstaat een verschil in geheugen: een core heeft een lokaal en een remote geheugen (NUMA).
- Echter, zolang een coherence protocol elke cache miss moet broadcasten, genereert dit zoveel verkeer dat het distribueren van het geheugen ons niet helpt.

Distributed Shared-Memory Multiprocessors (2)

- We lossen dit op met een ander coherence protocol: het *directory protocol*.
- In een directory wordt de toestand van elk cache blok bijgehouden.
  - Welke caches bevatten een kopie van dit blok?
  - Dirty of niet?
- Eenvoudig te implementeren binnen multicores met een shared (inclusive) L3 cache: voeg bij elk cache blok simpelweg een bit toe voor elke core.

Distributed Shared-Memory Multiprocessors (3)

- In een grootschalig systeem hebben we echter meerdere multicores en gedeelde L3 caches.
- In zo'n systeem heeft een enkele, centrale directory geen zin, dat geeft weer schalingsproblemen.
- De oplossing: distribueer de directory net als het geheugen.



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 5.6.

Directory-based coherence

We beschouwen een eenvoudig directory-based coherence protocol. Een cache blok is in een van de volgende toestanden:

- *Shared*: een of meer nodes hebben het blok in cache, het geheugen is up-to-date.
- *Uncached*: het blok is niet gecached.
- *Modified*: precies 1 node heeft het blok en het bevat een wijziging. Geheugen is niet up to date. (Deze node is de *owner*).

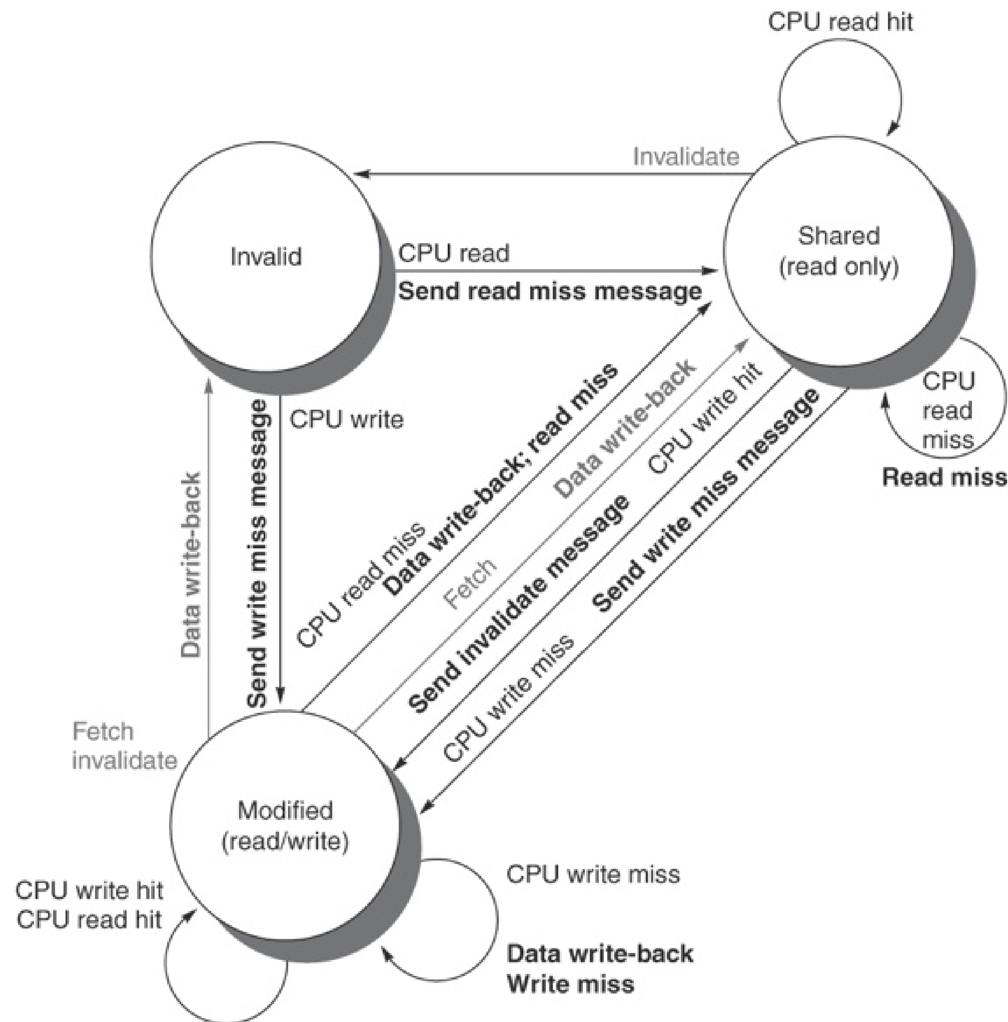
Een node is bijvoorbeeld een multicore chip met gedeelde L3 cache, waar binnen weer coherence is geïmplementeerd.

Directory-based coherence (2)

Definities:

- *Local node*: degene die het verzoek stuurt.
- *Home node*: node waar het geheugen en de directory voor een bepaalde geheugenlocatie is opgeslagen.
- *Remote node*: een andere node waar een kopie van het cache blok zou kunnen staan.

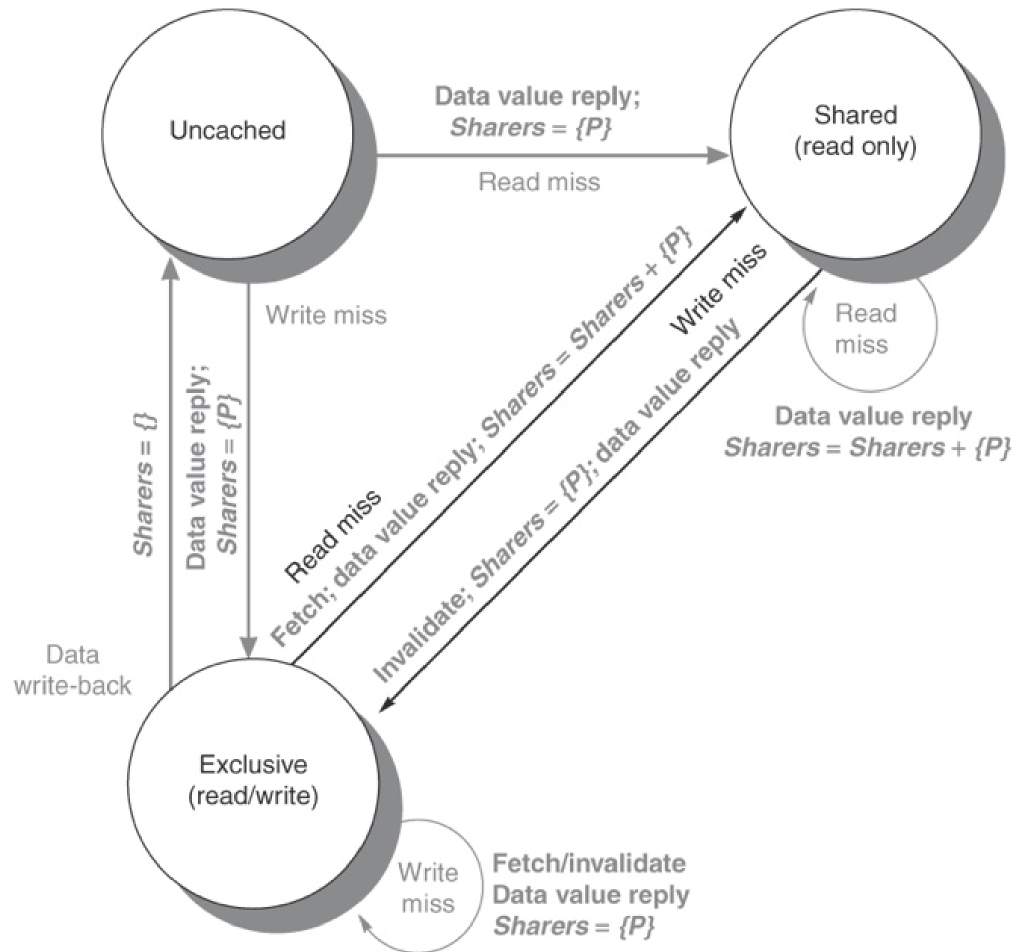
Directory-based coherence (3)



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 5.22.

Zwart: verzoeken van lokale processor / Grijs: verzoeken van home directory.

Directory-based coherence (4)



Taken from Computer Architecture: A Quantitative Approach, fifth edition. Fig. 5.23

Disclaimer

Real-world implementaties zijn nog ingewikkelder:

- Extra optimalisaties geïmplementeerd, waardoor er meer typen berichten bestaan die moeten worden afgehandeld.
- Een write is misschien al “compleet” voordat alle andere processoren hiervan weten.
- Wat wanneer coherence berichten in een andere volgorde aankomen (out-of-order)?
- Omgaan met atomic memory transactions.
- Combinatie multichip / multicore coherence; en dit moet wel allemaal goed samenwerken.