Efficient and Scalable Cache Coherence for Many-Core Architectures

Alberto Ros

Research assistant
Computer Engineering Department
Technical University of Valencia
aros@gap.upv.es

Adjunct professor
Computer Engineering Department
University of Murcia
a.ros@ditec.um.es

Uppsala, May 27, 2011

Outline

Outline

TRENDS

- The increasing number of transistors per chip can be used to obtain more performance.

TRENDS

- The increasing number of transistors per chip can be used to obtain more performance.

EXPLOITING ILP

- Very complex core
- Small improvements

Trends

- The increasing number of transistors per chip can be used to obtain more performance.

Exploiting ILP

- Very complex core
- Small improvements

$\Longrightarrow$

Exploiting TLP

- Many simple cores
- Programming effort

Trends

- The increasing number of transistors per chip can be used to obtain more performance.

Exploiting ILP

- Very complex core
- Small improvements

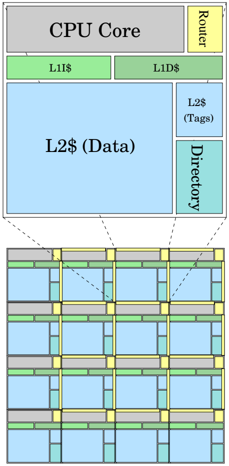
$\Rightarrow$

Exploiting TLP

- Many simple cores
- Programming effort

- Chip Multiprocessors (CMPs) constitute the new trend for increasing performance.
- Tiled CMPs are a scalable alternative for building CMPs.
  - Designed as arrays of replicated tiles.
  - Cores connected through a direct network.

TILED-CMPS
ARCHITECTURE

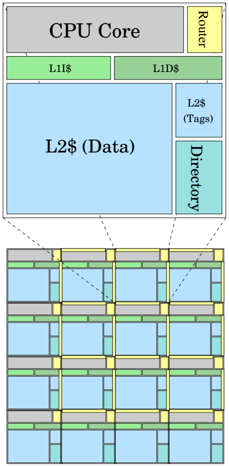


- Each tile contains:
  - A processing core.
  - A private L1 cache (both instruction and data caches).
  - A shared or private L2 cache bank, and a directory.
  - A network interface (router).
- All tiles are connected through a scalable point-to-point interconnection network.

Cache Coherence Problem

- Most parallel software in the commercial market relies on a shared-memory programming model.
- The presence of private caches requires to keep coherence among data stored in them.
- Solution $\Rightarrow$ Keep cache coherence in hardware.
- Problem $\Rightarrow$ Cache coherence protocols introduce extra overhead:
  - In terms of execution time.
  - In terms of area requirements.
  - In terms of power consumption.
  - In terms of designing and verification time.

Cache Coherence Problem

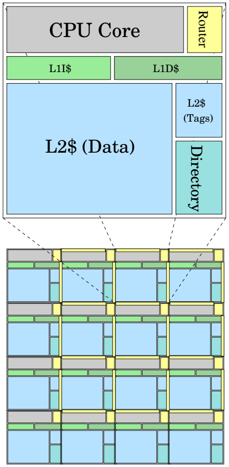


- Directory-based cache coherence protocols constitute the most scalable alternative.
  - But they have some inefficiencies and constraints:

1. Scalability of the directory structure.
2. Indirection to the home node.
3. Large verification time.

Memory Hierarchy Organization

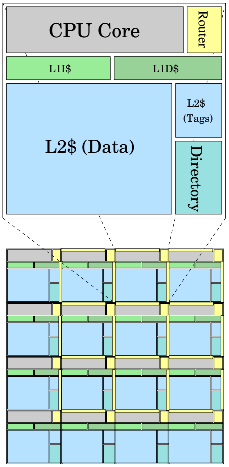
Shared Vs. Private Lastl-Level (L2) Cache Organization



Private Organization

- ☺ L2 hits have short latencies (local accesses).
- ☹ Blocks potentially replicated in multiple L2 banks.
- ☹ Load balancing problems.

Memory Hierarchy Organization
Shared Vs. Private Lastl-Level (L2) Cache Organization



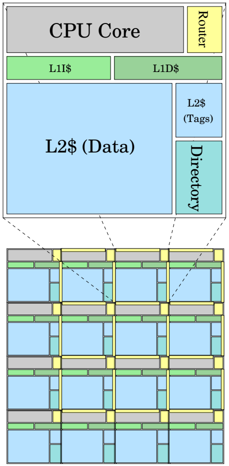
Private Organization

- ☺ L2 hits have short latencies (local accesses).
- ☹ Blocks potentially replicated in multiple L2 banks.
- ☹ Load balancing problems.

Shared Organization (NUCA architecture)

- ☺ Better use of the aggregate L2 cache capacity.
- ☹ Long latencies when compared to a private L2 design.
  - The access latency to the L2 depends on where the requested block is mapped.

Shared Cache Organization Challenges



- Tiled-CMPs distribute the shared last-level cache among the different tiles (Non Uniform Cache Access or NUCA architecture).
  - The access latency to the last-level cache depends on where the requested block is mapped.
  - Blocks requested by different threads competing for the same resources.

4. Reduce long access latencies.
5. Manage conflicting data requests from different threads.

OUTLINE

Direct Coherence
Motivation

Remember

Directory protocols introduce indirection in the critical path of cache misses.

- This indirection impacts on applications' performance.

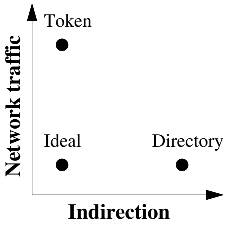
Direct Coherence
Motivation

Remember

Directory protocols introduce indirection in the critical path of cache misses.

- This indirection impacts on applications' performance.

- Token protocols have been proposed to avoid the indirection problem.
  - But they rely on broadcasting requests to all nodes.
  - They are not scalable in terms of network-traffic.
- An ideal protocol should avoid indirection while keeping traffic requirements low.



Direct Coherence
Indirection Problem

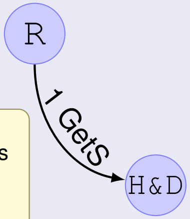
Cache-to-cache transfer in directory-based protocols

Direct Coherence

Indirection Problem

Cache-to-cache transfer in directory-based protocols

Cache miss



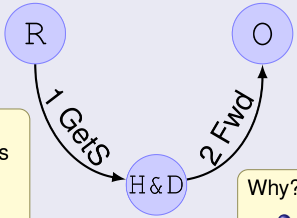
Why?

- To order requests
- To get directory information
- To provide main memory storage

Direct Coherence
Indirection Problem

Cache-to-cache transfer in directory-based protocols

Cache miss



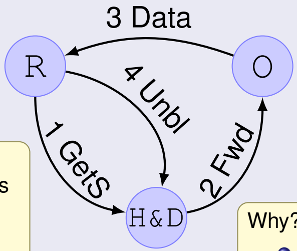
Why?
- To order requests
- To get directory information
- To provide main memory storage

Why?
- To get a *fresh* copy of the block

Direct Coherence
Indirection Problem

Cache-to-cache transfer in directory-based protocols

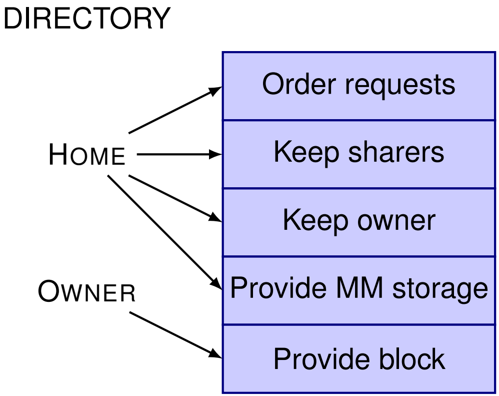
Cache miss



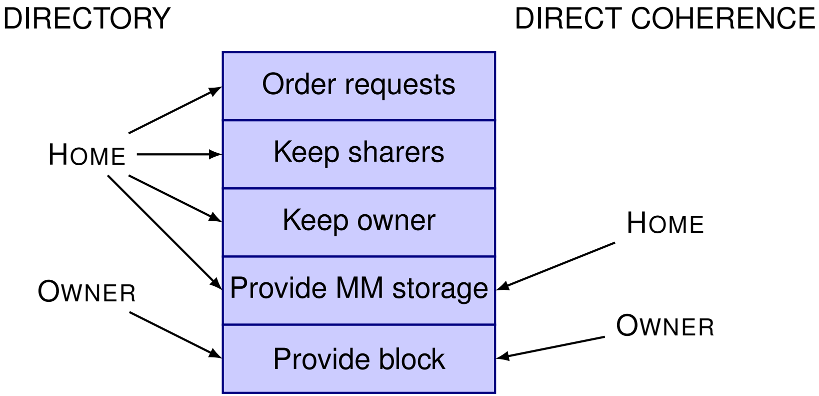
Why?
- To order requests
- To get directory information
- To provide main memory storage

Why?
- To get a *fresh* copy of the block

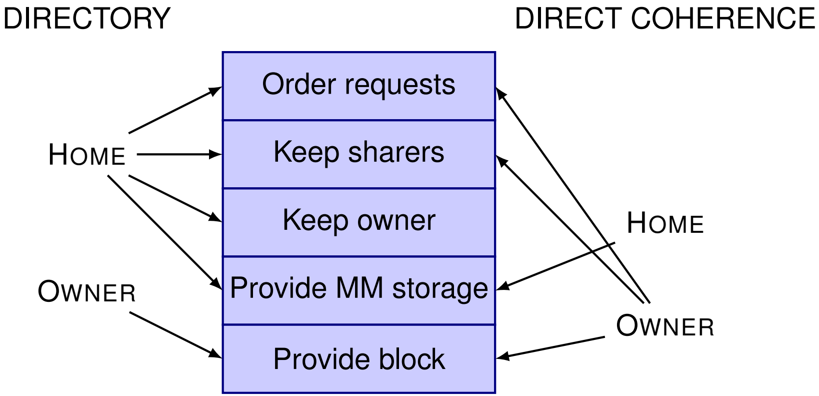
Direct Coherence
The Roles



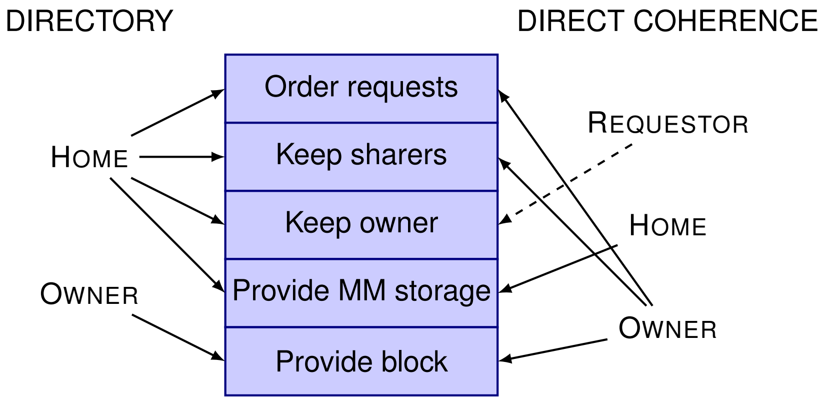
Direct Coherence
The Roles



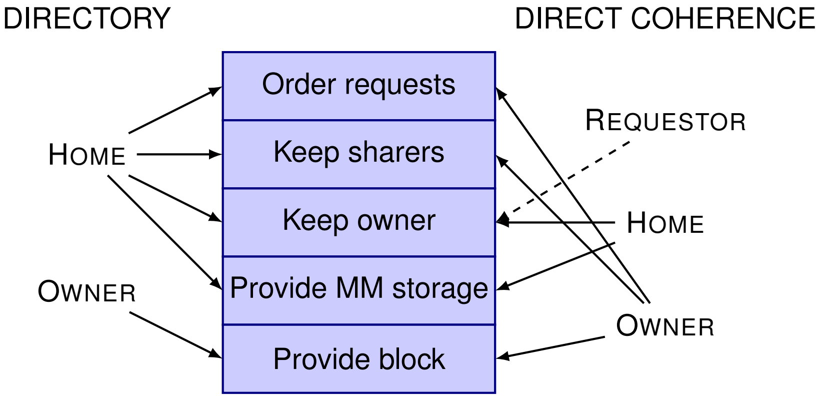
Direct Coherence
The Roles



Direct Coherence
The Roles

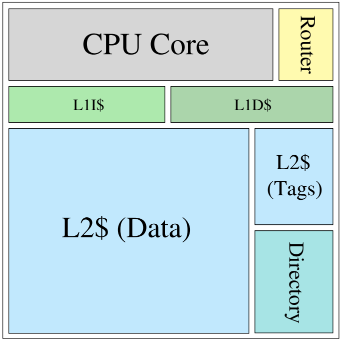


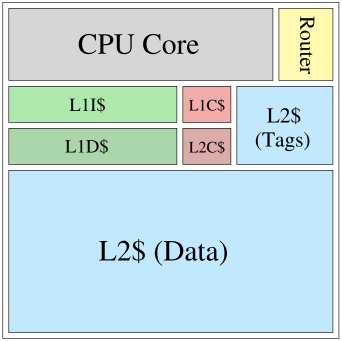
Direct Coherence
The Roles



Direct Coherence
Changes in the Structure of Tiles

- This distribution of the roles in direct coherence implies changes in the structure of each tile.



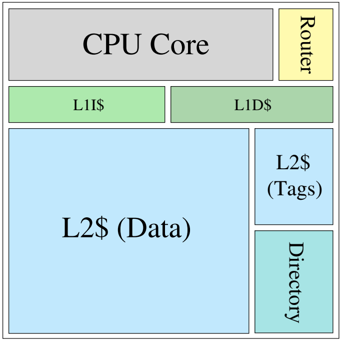


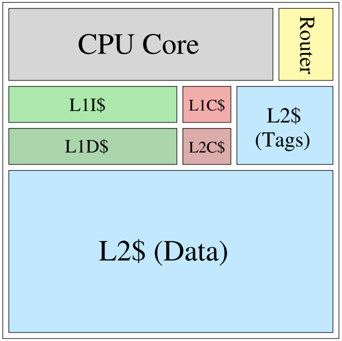
Direct Coherence
Changes in the Structure of Tiles

- T mplies c

L1D$: Adds Sharing Information

- Every owner cache must keep track of the sharers to keep coherence.
- This field replaces the directory structure.

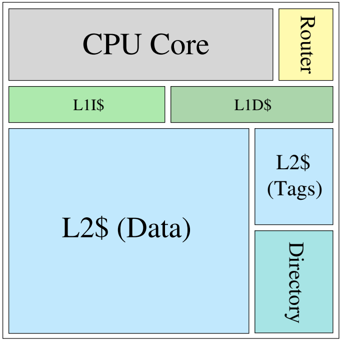


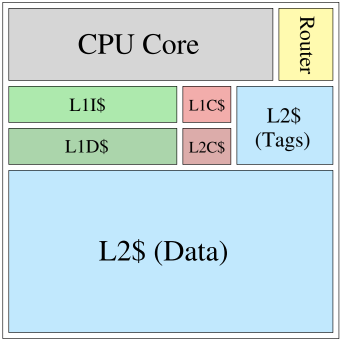


Direct Coherence
Changes in the Structure of Tiles

- This ies
  cha

L1C$: L1 Coherence Cache
- Each requesting cache stores the identity of the owner for some memory blocks.
- This information is used to directly send the requests to the corresponding owner cache.



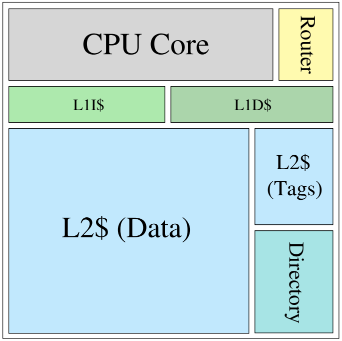


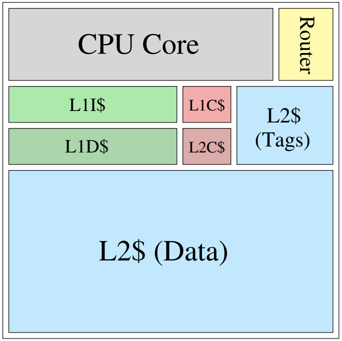
Direct Coherence
Changes in the Structure of Tiles

- This di
  chang

L2C$: L2 Coherence Cache

- Each home tile needs to store the identity of the owner cache of each one of its blocks.
- This information is accessed when the requestor is not able to locate the owner cache.



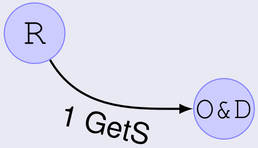


Direct Coherence
Behavior: Cache-to-Cache Read Miss

Direct Coherence

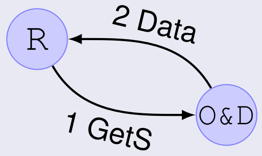
Direct Coherence
Behavior: Cache-to-Cache Read Miss

Direct Coherence



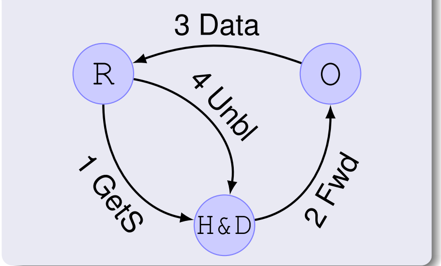
Direct Coherence
Behavior: Cache-to-Cache Read Miss

Direct Coherence

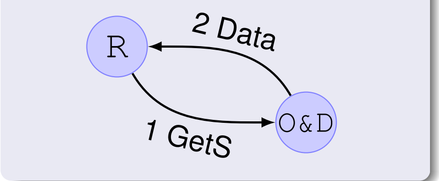


Direct Coherence
Behavior: Cache-to-Cache Read Miss

Directory



Direct Coherence



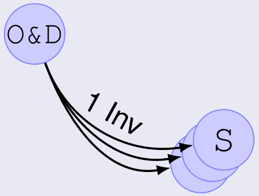
- The critical path of the miss is reduced from three to two hops.
- The number of coherence messages is halved.
- The waiting time at the home tile is removed.

Direct Coherence
Behavior: Upgrade in Owner

Direct Coherence

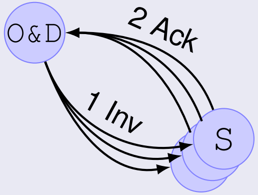
Direct Coherence
Behavior: Upgrade in Owner

Direct Coherence



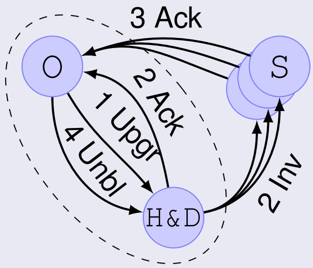
Direct Coherence
Behavior: Upgrade in Owner

Direct Coherence

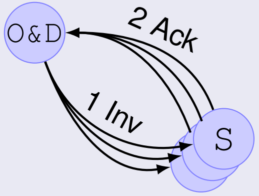


Direct Coherence
Behavior: Upgrade in Owner

Directory



Direct Coherence



- The critical path of the miss is reduced from three to two hops.
- The number of coherence messages is also reduced.

Direct Coherence
Updating the L2 coherence cache

- The L2C$ must keep the identity of the current owner cache for each block allocated in any L1 data cache.
  - This information is accessed when the requestor is not able to locate the owner cache.
- The L2C$ is notified on every owner change through control messages.

Direct Coherence
Updating the L2 coherence cache

- The L2C$ must keep the identity of the current owner cache for each block allocated in any L1 data cache.
  - This information is accessed when the requestor is not able to locate the owner cache.
- The L2C$ is notified on every owner change through control messages.

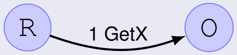
Write Miss in DiCo

Direct Coherence

Updating the L2 coherence cache

- The L2C$ must keep the identity of the current owner cache for each block allocated in any L1 data cache.
  - This information is accessed when the requestor is not able to locate the owner cache.
- The L2C$ is notified on every owner change through control messages.

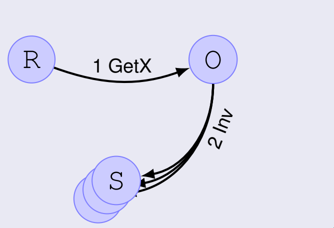
Write Miss in DiCo



Direct Coherence
Updating the L2 coherence cache

- The L2C$ must keep the identity of the current owner cache for each block allocated in any L1 data cache.
  - This information is accessed when the requestor is not able to locate the owner cache.
- The L2C$ is notified on every owner change through control messages.

Write Miss in DiCo

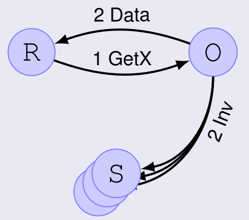


Direct Coherence

Updating the L2 coherence cache

- The L2C$ must keep the identity of the current owner cache for each block allocated in any L1 data cache.
  - This information is accessed when the requestor is not able to locate the owner cache.
- The L2C$ is notified on every owner change through control messages.

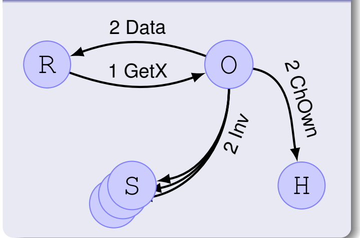
Write Miss in DiCo



Direct Coherence
Updating the L2 coherence cache

- The L2C$ must keep the identity of the current owner cache for each block allocated in any L1 data cache.
  - This information is accessed when the requestor is not able to locate the owner cache.
- The L2C$ is notified on every owner change through control messages.
- These messages should be processed by the L2C$ in the very same order in which they were generated.
  - To ensure this, the L2C$ sends an ACK message to the new owner when it receives a change owner message.

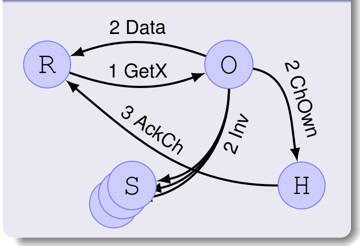
Write Miss in DiCo



Direct Coherence
Updating the L2 coherence cache

- The L2C$ must keep the identity of the current owner cache for each block allocated in any L1 data cache.
  - This information is accessed when the requestor is not able to locate the owner cache.
- The L2C$ is notified on every owner change through control messages.
- These messages should be processed by the L2C$ in the very same order in which they were generated.
  - To ensure this, the L2C$ sends an ACK message to the new owner when it receives a change owner message.
  - Until this message is not received by the owner node, it could use the block but cannot give the onwership to another cache.

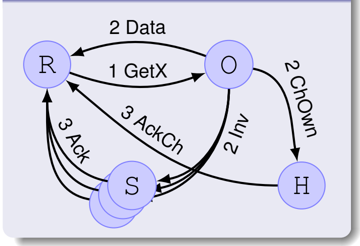
Write Miss in DiCo



Direct Coherence
Updating the L2 coherence cache

- The L2C$ must keep the identity of the current owner cache for each block allocated in any L1 data cache.
  - This information is accessed when the requestor is not able to locate the owner cache.
- The L2C$ is notified on every owner change through control messages.
- These messages should be processed by the L2C$ in the very same order in which they were generated.
  - To ensure this, the L2C$ sends an ACK message to the new owner when it receives a change owner message.
  - Until this message is not received by the owner node, it could use the block but cannot give the onwership to another cache.

Write Miss in DiCo



Direct Coherence
Updating the L1 coherence cache

- Base: information about the last core that invalidated or provided each block is kept in the L1C$.
  - Extra messages are not needed.
  - In some cases this information is not enough to obtain accurate predictions.

DIRECT COHERENCE
UPDATING THE L1 COHERENCE CACHE

- Base: information about the last core that invalidated or provided each block is kept in the L1C$.
  - Extra messages are not needed.
  - In some cases this information is not enough to obtain accurate predictions.
- Hints: control messages update the L1C$.
  - More accurate predictions.
  - Area and network traffic overhead.

FREQUENT SHARERS (FS)

- Area: Duplicated sharing information.
- Network: Hints sent on each owner change.

ADDRESS SIGNATURES (AS)

- Area: Two address signatures.
- Network: Hints filtering.

DIRECT COHERENCE

UPDATING THE L1 COHERENCE CACHE

- Base: information about the last core that invalidated or provided each block is kept in the L1C$.
  - Extra messages are not needed.
  - In some cases this information is not enough to obtain accurate predictions.
- Hints: control messages update the L1C$.
  - More accurate predictions.
  - Area and network traffic overhead.

FREQUENT SHARERS (FS)

- Area: Duplicated sharing information.
- Network: Hints sent on each owner change.

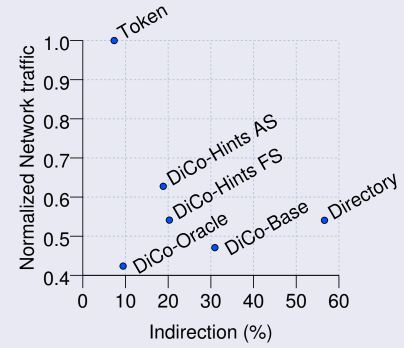
ADDRESS SIGNATURES (AS)

- Area: Two address signatures.
- Network: Hints filtering.

- Oracle: the requestor always knows the identity of the current owner.

DIRECT COHERENCE
EVALUATION

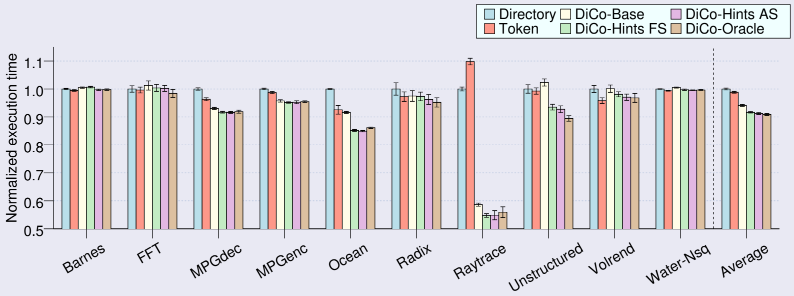
TRAFFIC-INDIRECTION TRADE-OFF



- *Directory* introduces indirection in the critical path of cache misses.
- *Token* generates high levels of network traffic.
- *DiCo-Base* reduces traffic even compared to *Directory*, but the indirection avoidance is limited.
- *DiCo-Hints* policies slightly increase traffic compared to *DiCo-Base* and successfully avoid indirection.

Direct Coherence
Evaluation

Applications' execution time



- *DiCo-Hints AS* reduces execution time compared to *Directory* (9%) and *Token* (8%).

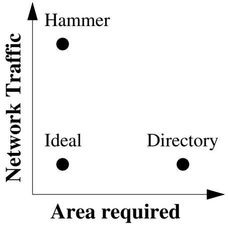
Direct Coherence

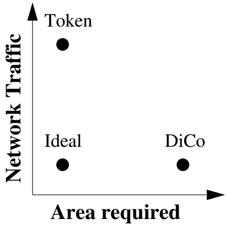
Traffic-Area Trade-Off in DiCo

- We have obtained a good trade-off between execution time and network traffic.
- However, the area requirements of *DiCo* do not scale with the number of cores.
- There are other protocols that scale better in terms of area.

Classification of protocols

	Traditional	Indirection-aware
Traffic-intensive	Hammer	Token
Area-demanding	Directory	DiCo





Direct Coherence

Traffic-area trade-off in DiCo

- Extra structures for keeping coherence:
  - L1C$: One pointer to the predicted owner $\Rightarrow O(log_2 n)$
  - L2C$: One pointer to the current owner $\Rightarrow O(log_2 n)$
  - Sharing information (L1 and L2): One bit per tile $\Rightarrow O(n)$
    - This structure compromises scalability.
- Solution: To use compressed sharing codes.
- Advantage of DiCo: The owner tile keeps cache coherence, so the first sharer (i.e., the owner) is always known.
  - Read misses do not need to check the sharing code field, so the compressed sharing code employed do not affect them.
  - Reduces network traffic compared to broadcast-based protocols even when the sharing information field is removed.

Direct Coherence
Compressed sharing codes

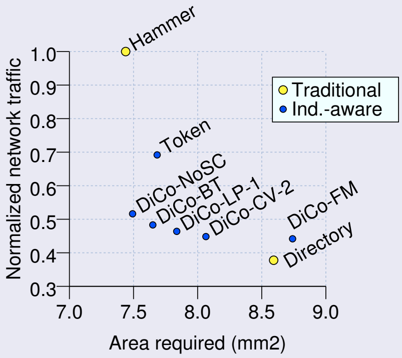
Sharing codes evaluated

Protocol	Sharing Code	Bits L1 cache and L2 cache	Bits L1C$ and L2C$	Order
DiCo-FM	Full-map	n	$log_2 n$	$O(n)$
DiCo-CV-K	Coarse vector	$\frac{n}{K}$	$log_2 n$	$O(n)$
DiCo-LP-P	Limited pointers	$1 + P \times log_2 n$	$log_2 n$	$O(log_2 n)$
DiCo-BT	Binary Tree	$\lceil log_2(1 + log_2 n) \rceil$	$log_2 n$	$O(log_2 n)$
DiCo-NoSC	None	0	$log_2 n$	$O(log_2 n)$

- We evaluate the *DiCo-Hints AS* policy.
- *DiCo-FM* is the previously evaluated *DiCo-Hints AS* policy.

Direct Coherence
Evaluation

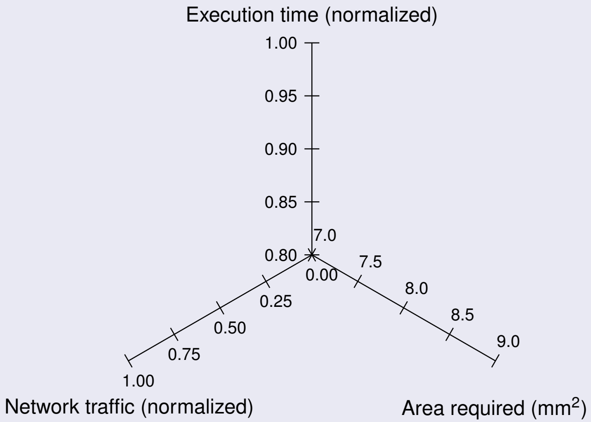
Traffic-area trade-off



- *Hammer* and *Token* are traffic-intensive.
- *Directory* and *DiCo-FM* are area-demanding.
- *DiCo-BT* achieves a good compromise.
- *DiCo-NoSC* also achieves a good compromise without modifying the data caches.

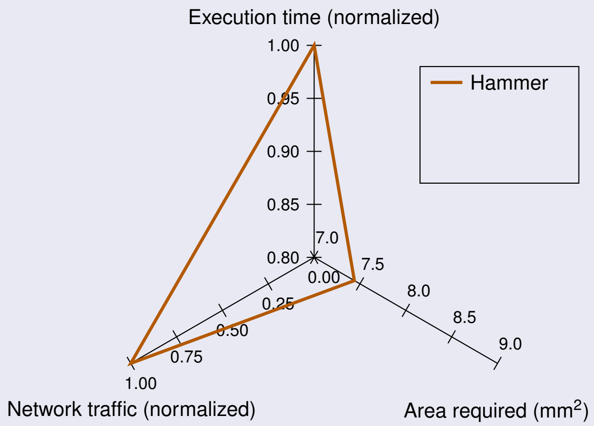
Direct Coherence
Evaluation

Overall Trade-Off



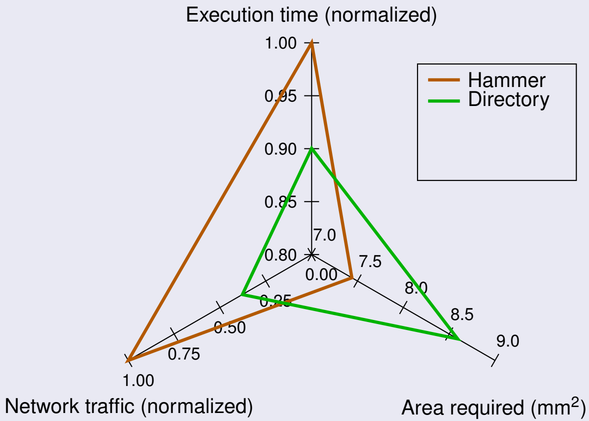
Direct Coherence
Evaluation

Overall Trade-Off



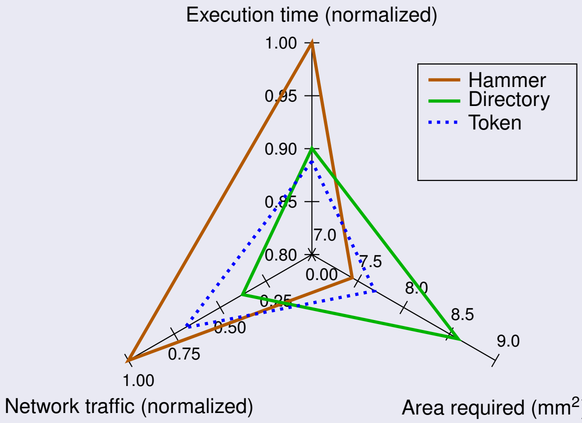
Direct Coherence
Evaluation

Overall Trade-Off



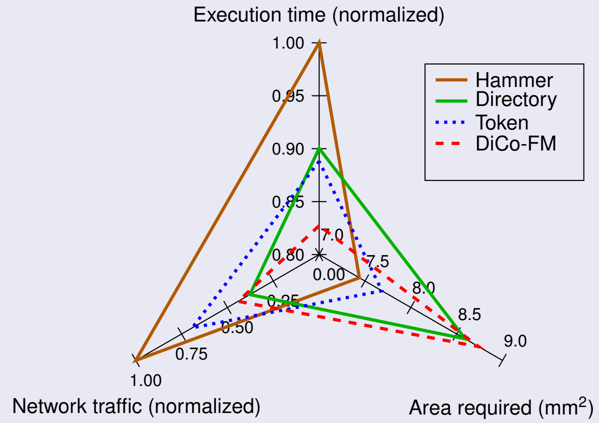
Direct Coherence
Evaluation

Overall Trade-Off



Direct Coherence
Evaluation

Overall Trade-Off

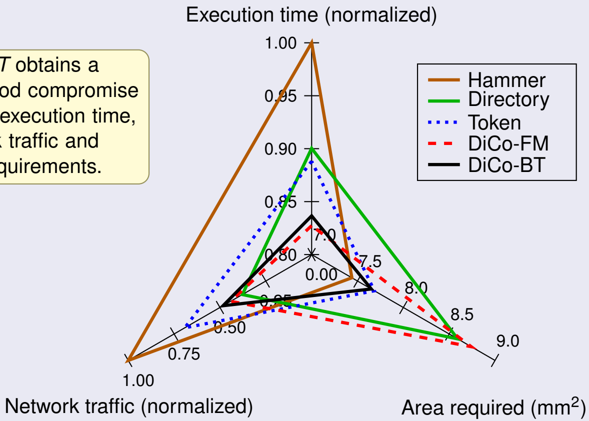


Direct Coherence

Evaluation

Overall Trade-Off

DiCo-BT obtains a very good compromise among execution time, network traffic and area requirements.



Direct Coherence
Conclusions

- Direct coherence protocols:
  - Do not rely on broadcasting requests.
  - Avoid the indirection for most cache misses.
  - Work well with compressed sharing codes.
- The following improvements have been obtained by DiCo-FM (Hints AS):
  - Execution time: 9% compared to *Directory* and 8% compared to *Token*.
  - Network traffic: 37% compared to *Token* and a slightly increase compared to *Directory*.
- *DiCo-BT* and *DiCo-NoSC* obtain a good trade-off among execution time, network traffic and area requirements.

Direct Coherence
Publications

International conferences

- **A. Ros**, M. E. Acacio and J. M. García, *"Direct Coherence: Bringing Together Performance and Scalability in Shared-Memory Multiprocessors"*. **HiPC'07**.
- **A. Ros**, M. E. Acacio and J. M. García, *"DiCo-CMP: Efficient Cache Coherency in Tiled CMP Architectures"*. **IPDPS'08**.
- **A. Ros**, M. E. Acacio and J. M. García, *"Dealing with Traffic-Area Trade-Off in Direct Coherence Protocols for Many-Core CMPs"*. **APPT'09**.

International journals

- **A. Ros**, M. E. Acacio and J. M. García, *"A Direct Coherence Protocol for Many-Core Chip Multiprocessors"*. **TPDS**, Dec 2010.

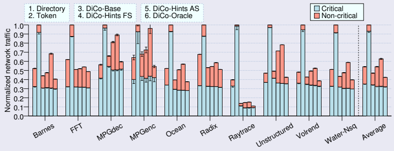
Book Chapters

- **A. Ros**, M. E. Acacio and J. M. García, *"Cache Coherence Protocols for Many-Core CMPs"*. Parallel and Distributed Computing.

Direct Coherence
Current work

- Heterogeneous networks:
  - Network provided with fast and low-power links.
  - Non-critical messages can be sent by low-power links.
  - DiCo increases the number of non-critical messages: hints.

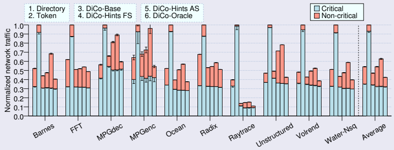
Non-critical traffic



Direct Coherence
Current work

- Heterogeneous networks:
  - Network provided with fast and low-power links.
  - Non-critical messages can be sent by low-power links.
  - DiCo increases the number of non-critical messages: hints.

Non-critical traffic



- Server consolidation or multiprogrammed workloads:
  - Several virtual machines (VM) in a CMP.
  - Home nodes can map anywhere.
  - Owner nodes will likely be in the same VM.

Outline

Coherence Deactivation
Motivation

Remember

Directory protocols are the most scalable alternative for keeping cache coherence.

- But the area requirements of the directory structure could become prohibitive for large-scale multiprocessors.

Coherence Deactivation
Motivation

Remember

Directory protocols are the most scalable alternative for keeping cache coherence.

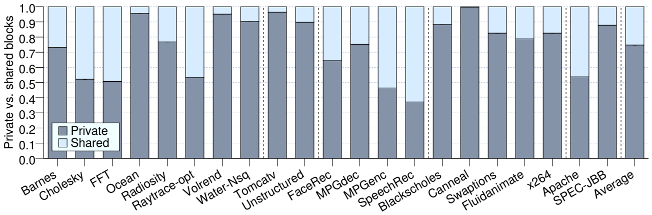
- But the area requirements of the directory structure could become prohibitive for large-scale multiprocessors.

- Directory caches accelerate the access to the coherence information and reduce directory overhead with respect to a memory directory but...
    - ...directory cache evictions cause the invalidation of cached data, resulting in performance degradation.

Coherence Deactivation

Motivation

- Is it necessary to keep cache coherence for all referenced blocks?
  - Private blocks will never be incoherent!
    - 75% of referenced blocks (on average).
- If we do not maintain directory information for these blocks we can save a lot of directory storage.

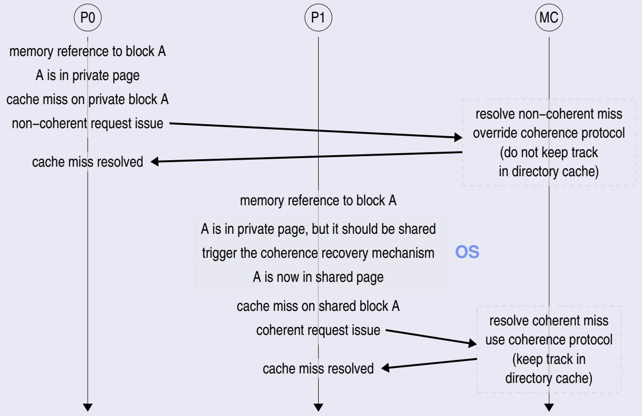


Coherence Deactivation
Proposal

- We propose a mechanism that:
  - Classifies memory blocks into private and shared (How?).
  - Deactivates the coherence protocol for private blocks
    - i.e., do not keep track of them.
- A block-grain classification would require significant storage resources.
  - Blocks are classified at page granularity.
  - The operating system detects when a page (initially considered private) becomes shared (minimal OS overhead).
    - Performed upon TLB misses: state stored in the page table.
    - A coherence recovery mechanism is necessary to restore block's coherence status.
  - Collaboration between hardware and operating system.

Coherence Deactivation
Example

Requests for private and shared blocks

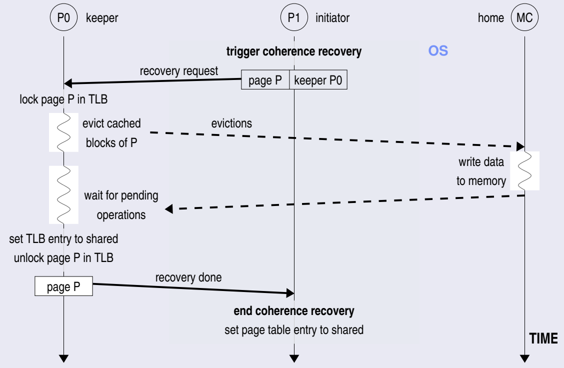

P0
P1
MC
memory reference to block A
A is in private page
cache miss on private block A
non–coherent request issue
cache miss resolved
resolve non–coherent miss
override coherence protocol
(do not keep track
in directory cache)
memory reference to block A
A is in private page, but it should be shared
trigger the coherence recovery mechanism
A is now in shared page
OS
cache miss on shared block A
coherent request issue
cache miss resolved
resolve coherent miss
use coherence protocol
(keep track in
directory cache)

Coherence Deactivation
The recovery mechanism

- When a private page becomes shared, it is necessary:
  1. Make coherent directory caches with cached blocks ⇒ recovery mechanism.
     - Triggered by OS during TLB miss resolution (critical section)
  2. From this point on, keep track of the blocks in this page.
- To main options:
  - Flushing-based recovery: evicts blocks in the page being recovery from processor caches.
  - Updated-based recovery: updates directory caches with the information about cached blocks.

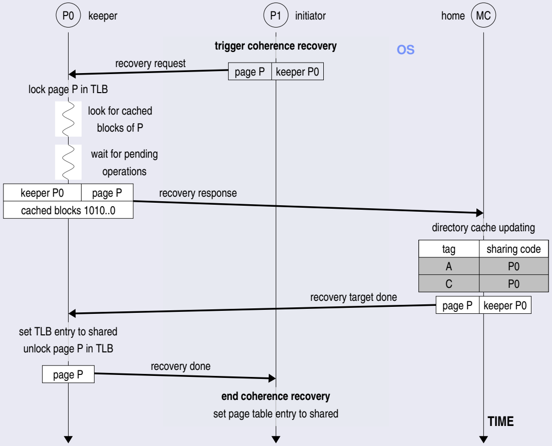
Coherence Deactivation
Flushing-based recovery mechanism

Example of the flushing-based recovery



Coherence Deactivation
Updating-based recovery mechanism

Example of the updating-based recovery



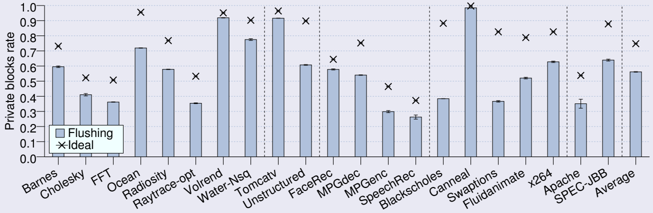
Coherence Deactivation
Advantages

- The amount of directory information required to maintain coherence is reduced.
    - Reduce directory cache evictions to improve performance.
    - Reduce directory cache size while keeping performance.
- Request for private blocks do not need to access the directory structure.
    - Savings in both cache miss latency and power consumption.

Coherence Deactivation
Results: Detected blocks

- Private blocks (75%) ⇒ Detected private blocks (57%).
- 57% memory blocks are not tracked

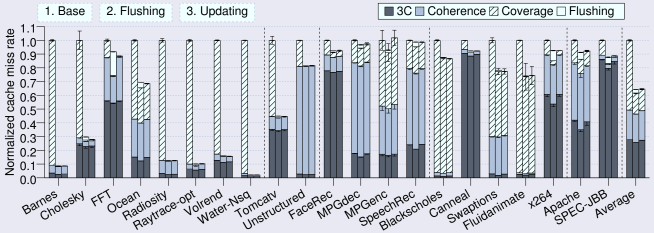
Private blocks vs. detected blocks



Coherence Deactivation
Results: Cache misses

- 5C cache miss classification.
- 75% coverage misses avoided (35% overall).

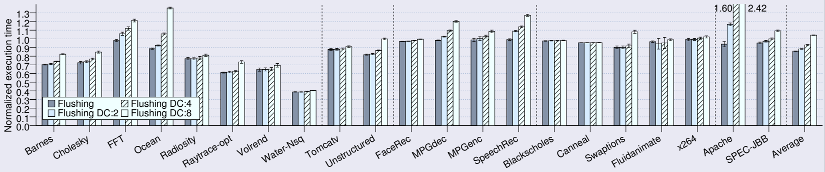
Normalized number of processor cache misses



Coherence Deactivation
Results: Execution time

- Similar runtime for flushing and updating recovery mechanisms.
- With same directory size ⇒ performance improvement: 15%.
- With same performance ⇒ directory cache 8 times smaller.

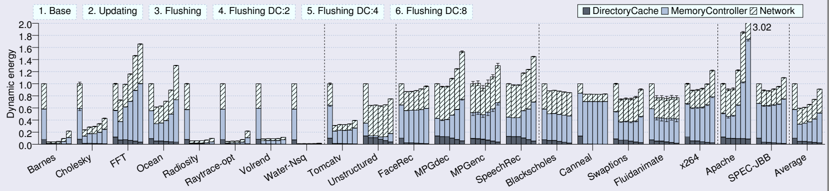
Normalized runtime



Coherence Deactivation
Results: Energy consumption

- Reduced by 40% when keeping directory cache size.

Normalized dynamic energy consumption



Coherence Deactivation

Future work and publications

- Publications:

International conferences

- B. Cuesta, **A. Ros**, M. E. Gómez, A. Robles, and J. Duato, *“Increasing the Effectiveness of Directory Caches by Deactivating Coherence for Private Memory Blocks”*. **ISCA’11**.

International journals

- B. Cuesta, **A. Ros**, M. E. Gómez, A. Robles, and J. Duato, *“Increasing the Effectiveness of Directory Caches by Avoiding the Tracking of Non-Coherent Memory Blocks”*. Submitted to **TC**.

COHERENCE DEACTIVATION
FUTURE WORK AND PUBLICATIONS

- Publications:

INTERNATIONAL CONFERENCES

- B. Cuesta, **A. Ros**, M. E. Gómez, A. Robles, and J. Duato, *"Increasing the Effectiveness of Directory Caches by Deactivating Coherence for Private Memory Blocks"*. **ISCA'11**.

INTERNATIONAL JOURNALS

- B. Cuesta, **A. Ros**, M. E. Gómez, A. Robles, and J. Duato, *"Increasing the Effectiveness of Directory Caches by Avoiding the Tracking of Non-Coherent Memory Blocks"*. Submitted to **TC**.

- Future work:
  - Thread migration can reduce the number of non-coherent blocks.
  - A page-grained classification misclassifies about 18% of blocks.
    - Blocks detected as coherent are actually non-coherent.
  - Possible solutions: Hardware mechanisms or modifications in the programing language?

Outline

Synchronous Coherence
Motivation

Remember

- The verification of a cache coherence protocol is very time-consuming and tedious.

Synchronous Coherence
Motivation

Remember

- The verification of a cache coherence protocol is very time-consuming and tedious.

- The more complex the coherence protocol is, the more verification time is required.
- The appearance of race conditions makes even harder the protocol verification.
- Some authors reduce protocol races by relying on atomic transitions [1].
- Another approach: simple request-response protocols.

References

[1] D. Vantrease, M. H. Lipasti, and N. Binkert, *"Atomic Coherence: Leveraging Nanophotonics to Build Race-Free Cache Coherence Protocols"*. **HPCA'10**.

Synchronous Coherence

Request-response protocols

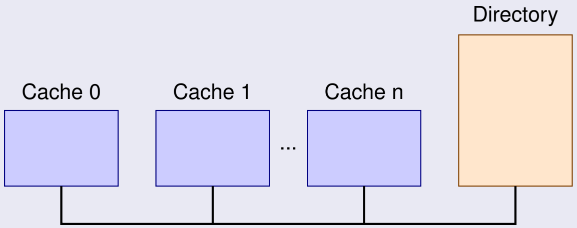
- A request-response protocol does not forwards requests to other nodes (2-hop protocol).
  - The requester issues a message to the home node.
  - The home node directly responds with a copy of the request block.
- What happens with dirty cached copies?
  - Write-through caches? $\Rightarrow$ Not very efficient.
  - Solution: time-based cache coherence protocols (synchronous coherence).
    - A global clock is needed $\Rightarrow$ use of global lines [2].
    - Block stored in cache will have expiration date!
    - When a cached block expires it will be invalidated, performing a writeback in case the block is dirty.

References

[2] R. T. Chang, N. Talwalkar, C. P. Yue, and S. S. Wong, *"Near Speed-of-Light Signaling Over On-Chip Electrical Interconnects"*. IEEE Journal of Solid-State Circuits, 2003.

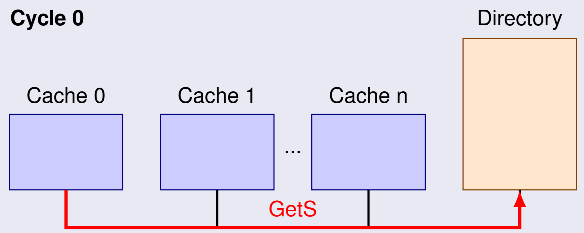
Synchronous Coherence
Example

Request-response protocol with expiration date for cached blocks



Synchronous Coherence
Example

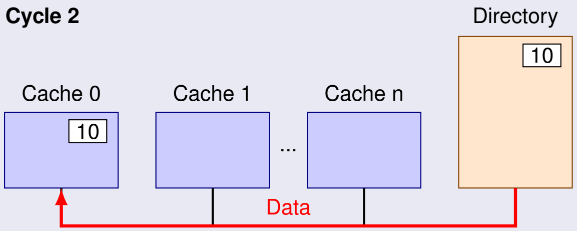
Request-response protocol with expiration date for cached blocks



Synchronous Coherence
Example

- Directory does not keep list of sharers but expiration date.

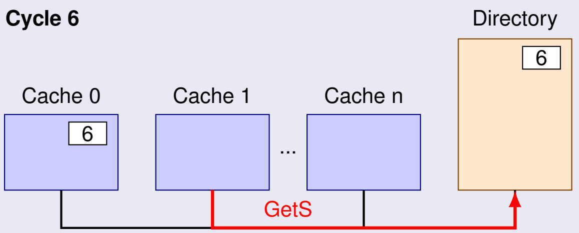
Request-response protocol with expiration date for cached blocks



Synchronous Coherence
Example

- Directory does not keep list of sharers but expiration date.

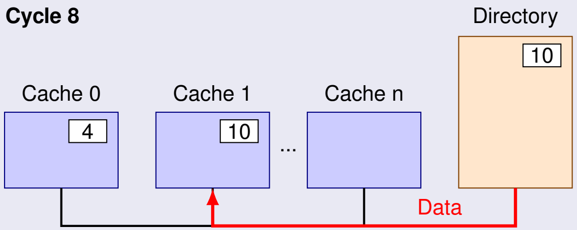
Request-response protocol with expiration date for cached blocks



Synchronous Coherence
Example

- Directory does not keep list of sharers but expiration date.

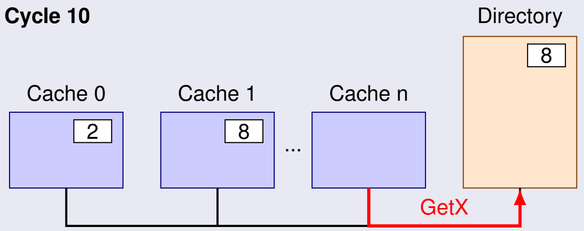
Request-response protocol with expiration date for cached blocks



Synchronous Coherence
Example

- Directory does not keep list of sharers but expiration date.
- GetX transaction waits until the block expires.

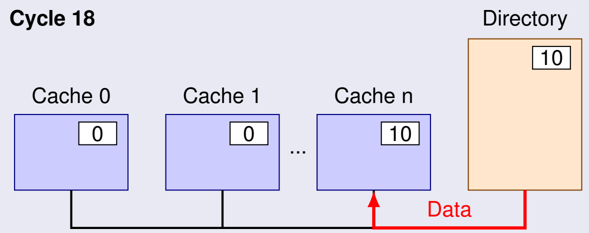
Request-response protocol with expiration date for cached blocks



Synchronous Coherence
Example

- Directory does not keep list of sharers but expiration date.
- GetX transaction waits until the block expires.
- Memory sends the block to the requester and a new expiration date is assigned.

Request-response protocol with expiration date for cached blocks



Outline

INTRODUCTION

- There are several challenges to address for the memory hierarchy organization of a CMP.
  - Thread Balancing problems.
    - Imbalance in time: Some threads arrive to a barrier before the other ones $\Rightarrow$ Can increase execution time.
    - Imbalance in storage: The working set of threads also varies $\Rightarrow$ Can increase cache misses (off-chip accesses).
  - Conflict misses.
    - Reduce last level conflict misses also can save off-chip accesses.
  - Long access latency to NUCA banks.
    - Several authors address this problem but they do not care about directory scalability.

OUTLINE

Replacement Policies for Shared Caches
Motivation

- In parallel applications, some threads arrive to a barrier before the other ones.
- The first threads arriving to a barrier start a busy waiting.
  - This consumes extra power.
- Some authors propose to save power consumption by slowing down faster threads (e.g., reducing processor frequency) [3,4].
  - This saves power but it does not improve execution time.
- Another approach: A thread-aware replacement policy.

References

[3] J. Li, J. F. Martínez, and M. C. Huang, *“The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors”*. **HPCA’04**.

[4] Q. Cai, J. González, R. Rakvic, G. Magklis, P. Chaparro, and A. González, *“Meeting Points: Using Thread Criticality to Adapt Multicore Hardware to Parallel Regions”*. **PACT’08**.

Replacement Policies for Shared Caches

A thread-aware replacement policy

- Sets in a shared cache hold blocks from different threads.
- A smart policy can be implemented:
  - Avoid evictions of blocks accessed by slower threads, or widely shared.
  - Evicts private blocks accessed by faster threads.
- Directory caches already store information about which processors hold the blocks.
- Since slower threads are accelerated, the final execution time can be reduced.
- Another option for balancing threads $\Rightarrow$ Lock priorities.
  - Give more priority for lock acquisition to slower threads.

Outline

Indexing Policies for Shared Caches
Motivation

- Memory references are not often distributed across cache sets.
  - Some sets exhibit large miss ratios, while other are underutilized.
- This causes the appearance of conflict misses.
  - Can be reduced by increasing associativity.
    - But this would increase power consumption and access latency.
- Misses in the shared last-level cache cause expensive off-chip accesses.
  - Some authors reduce conflict misses by reallocating blocks to underutilized sets [5].
  - Another approach: adaptive selection of index bits.

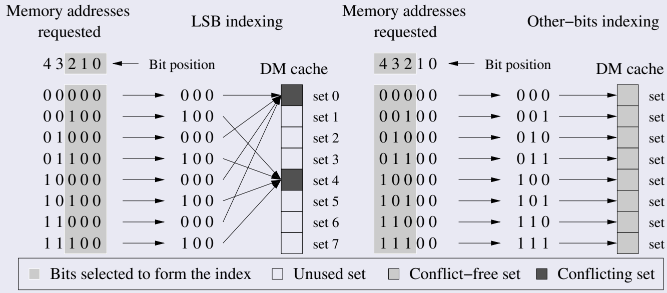
References

[5] D. Rolán, B. B. Fraguela, and R Doallo, *"Adaptive Line Placement with the Set Balancing Cache"*. **MICRO'09**.

Indexing Policies for Shared Caches
Motivation example

- If we carefully chose the address bits for indexing the cache, a better set balancing can be obtained.
- Why choose other bits apart from the least significant bits (LSB)?
  - Example 1: stride memory access pattern.
  - Example 2: parallel applications (shared array partitioned).
  - Example 3: second level caches (L1 filter accesses to contiguous blocks).

LSB indexing vs. Other bits indexing



Indexing Policies for Shared Caches

Current and future work

- Current work:
  - A first approach for direct-mapped first-level caches.
  - Submitted to **PACT'11**.
- Future work:
  - Private caches in CMPs.
  - Set-associative caches.
  - Last-level cache in CMPs.
    - Where misses cause expensive off-chip accesses.
    - Thread balancing?

Outline

NUCA Mapping and Directory Scalability
Motivation

Remember

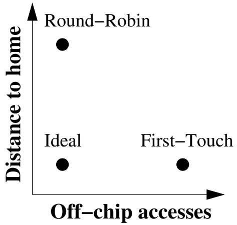
In NUCA (Non-Uniform Cache Architecture) caches, the access latency depends on where the requested block is mapped (home bank).

NUCA Mapping and Directory Scalability
Motivation

Remember

In NUCA (Non-Uniform Cache Architecture) caches, the access latency depends on where the requested block is mapped (home bank).

- This mapping is commonly performed by taking some bits from the block address leading to a Round-Robin mapping.
  - The *Round-Robin* mapping does not care about the distance between requesting cores and home banks ⇒ long access latency.
- A First-Touch mapping policy can lessen this latency.
  - But can cause imbalance among cache banks ⇒ high cache miss rate.



NUCA Mapping and Directory Scalability
Previous work

- Several authors have study the trade-off between low miss rate and low access time in NUCA caches [6,7].
  - But these works does not care about directory scalability.
  - They are based on OS allocation policies at page granularity...
    - ...which can affect directory scalability.

References

[6] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, *"Reactive NUCA: Near-optimal block placement and replication in distributed caches"*. **ISCA'09**.

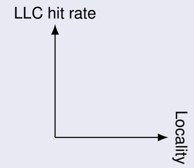
[7] **A. Ros**, M. Cintra, M. E. Acacio and J. M. García, *"Distance-Aware Round-Robin Mapping for Large NUCA Caches"*. **HiPC'09**.

NUCA Mapping and Directory Scalability

A new metric: directory scalability

- A directory cache based on duplicated tags can perfectly scale (in size) up to a certain number of nodes [8].
  - This number of nodes corresponds to the number of private cache sets.
    - Commonly, first-level caches have between 128 and 512 sets.
  - Constraint: the mapping of memory blocks to home banks must be done at fine granularity (i.e., block granularity).

Trade-off diagram



References

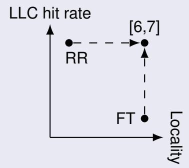
[8] **A. Ros**, M. E. Acacio and J. M. García, *"A Scalable Organization for Distributed Directories"*. **JSA**, Mar, 2010.

NUCA Mapping and Directory Scalability

A new metric: directory scalability

- A directory cache based on duplicated tags can perfectly scale (in size) up to a certain number of nodes [8].
  - This number of nodes corresponds to the number of private cache sets.
    - Commonly, first-level caches have between 128 and 512 sets.
  - Constraint: the mapping of memory blocks to home banks must be done at fine granularity (i.e., block granularity).

Trade-off diagram



References

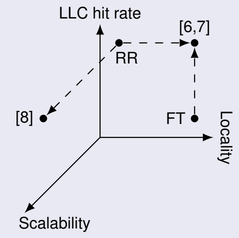
[8] **A. Ros**, M. E. Acacio and J. M. García, *“A Scalable Organization for Distributed Directories”*. **JSA**, Mar, 2010.

NUCA Mapping and Directory Scalability

A new metric: directory scalability

- A directory cache based on duplicated tags can perfectly scale (in size) up to a certain number of nodes [8].
    - This number of nodes corresponds to the number of private cache sets.
        - Commonly, first-level caches have between 128 and 512 sets.
    - Constraint: the mapping of memory blocks to home banks must be done at fine granularity (i.e., block granularity).

Trade-off diagram



References

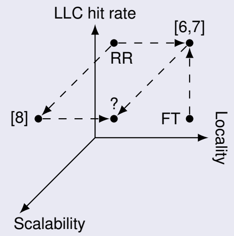
[8] **A. Ros**, M. E. Acacio and J. M. García, *"A Scalable Organization for Distributed Directories"*. **JSA**, Mar, 2010.

NUCA Mapping and Directory Scalability

A new metric: directory scalability

- A directory cache based on duplicated tags can perfectly scale (in size) up to a certain number of nodes [8].
  - This number of nodes corresponds to the number of private cache sets.
    - Commonly, first-level caches have between 128 and 512 sets.
  - Constraint: the mapping of memory blocks to home banks must be done at fine granularity (i.e., block granularity).

Trade-off diagram



References

[8] **A. Ros**, M. E. Acacio and J. M. García, *"A Scalable Organization for Distributed Directories"*. **JSA**, Mar, 2010.

NUCA Mapping and Directory Scalability
Possible solution

- Directory scalability requires block-grained interleaving.
- Low latency and miss rate approaches employ page-grained interleaving.
- Possible solution $\Rightarrow$ Decoupling directory information and data blocks.
  - Read requests may not require directory information.
    - So they can be sent to the data home bank.
  - Upgrade requests do not require data.
    - So they can be sent to the directory home bank.
  - Only few requests will require both data and directory information.

Outline

Conclusions

- Currently, working on several proposals for improve both coherence protocols and cache hierarchy.
- Publications in high-level conferences and journals.
- Future work on both fields seems promising.
- Collaboration between hardware and operating system is very effective.
- Collaboration at other levels: compiler and programming languages is also interesting.

Efficient and Scalable Cache Coherence for Many-Core Architectures

Alberto Ros

Research assistant
Computer Engineering Department
Technical University of Valencia
aros@gap.upv.es

Adjunct professor
Computer Engineering Department
University of Murcia
a.ros@ditec.um.es

Uppsala, May 27, 2011