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
1 INTRODUCTION
   - Challenges in many-core computing

2 CACHE COHERENCE PROTOCOLS
   - Direct coherence (DiCo)
   - Coherence deactivation
   - Synchronous coherence

3 MEMORY HIERARCHY ORGANIZATION
   - Replacement policies for shared caches
   - Indexing policies for shared caches
   - Impact of NUCA mapping policies on directory scalability

4 CONCLUSIONS
INTRODUCTION

- Challenges in many-core computing

CACHE COHERENCE PROTOCOLS

- Direct coherence (DiCo)
- Coherence deactivation
- Synchronous coherence

MEMORY HIERARCHY ORGANIZATION

- Replacement policies for shared caches
- Indexing policies for shared caches
- Impact of NUCA mapping policies on directory scalability

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

**Exploiting ILP**
- Very complex core
- Small improvements
The increasing number of transistors per chip can be used to obtain more performance.

**Exploiting ILP**
- Very complex core
- Small improvements

**Exploiting TLP**
- Many simple cores
- Programming effort
The increasing number of transistors per chip can be used to obtain more performance.

**Exploiting ILP**
- Very complex core
- Small improvements

**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.
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 ⇒ Keep cache coherence in hardware.

Problem ⇒ 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.
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.

Alberto Ros
Research lines
Uppsala, May 27, 2011
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.
1. **INTRODUCTION**
   - Challenges in many-core computing

2. **CACHE COHERENCE PROTOCOLS**
   - Direct coherence (DiCo)
   - Coherence deactivation
   - Synchronous coherence

3. **MEMORY HIERARCHY ORGANIZATION**
   - Replacement policies for shared caches
   - Indexing policies for shared caches
   - Impact of NUCA mapping policies on directory scalability

4. **CONCLUSIONS**
**DIRECT COHERENCE MOTIVATION**

**REMEMBER**

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

- This indirection impacts on applications’ performance.
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**

- **Cache miss**
- **R**
**DIRECT COHERENCE**

**INDIRECTION PROBLEM**

---

**CACHE-TO-CACHE TRANSFER IN DIRECTORY-BASED PROTOCOLS**

1. **Cache miss**

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

2. **R**

3. **GetS**

4. **H&D**
Cache Coherence 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

- R
  - 1 GetS
- H & D
  - 2 Fwd
- O
  - Why?
    - To get a *fresh* copy of the block
**Direct Coherence Indirection Problem**

**Cache-to-Cache Transfer in Directory-Based Protocols**

1. **GetS** (Get Directory Information)
   - Why? To order requests
   - Why? To get directory information
   - Why? To provide main memory storage

2. **Fwd** (Forwarded)
   - Why? To get a fresh copy of the block

3. **Data**

4. **Unbl** (Unblock)

**R** (Request) 

**H&D** (Hit & Directory)

**O** (Owner)
DIRECT COHERENCE
THE ROLES

DIRECTORY

HOME

- Order requests
- Keep sharers
- Keep owner

OWNER

- Provide MM storage
- Provide block
DIRECT COHERENCE
THE ROLES

DIRECTORY

HOME

OWNER

DIRECT COHERENCE

Order requests
Keep sharers
Keep owner
Provide MM storage
Provide block
Direct Coherence

The Roles

**DIRECTORY**
- Order requests
- Keep sharers
- Keep owner
- Provide MM storage
- Provide block

**DIRECT COHERENCE**
- HOME
- OWNER
**DIRECT COHERENCE**

**THE ROLES**

**DIRECTORY**

- Order requests
- Keep sharers
- Keep owner
- Provide MM storage
- Provide block

**DIRECT COHERENCE**

- Requestor
- Home
- Owner
DIRECT COHERENCE
THE ROLES

ORDER REQUESTS
- Keep sharers
- Keep owner
- Provide MM storage
- Provide block
This distribution of the roles in direct coherence implies changes in the structure of each tile.

- **L1I$**
- **L1D$**
- **L2$ (Data)**
- **L2$ (Tags)**
- **Directory**

- **L1I$**
- **L1C$**
- **L1D$**
- **L2C$**
- **L2$ (Tags)**
DIRECT COHERENCE
CHANGES IN THE STRUCTURE OF TILES

L1D$: Adds Sharing Information
- Every owner cache must keep track of the sharers to keep coherence.
- This field replaces the directory structure.

Alberto Ros
Research lines
Uppsala, May 27, 2011 14 / 63
This distribution of the roles in direct coherence implies changes in the structure of each tile.

- **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.

- **L1I$: L1 Instruction Cache**
  - This cache stores the instructions that are not present in the L1 Data Cache.

- **L1D$: L1 Data Cache**
  - This cache stores the data that is not present in the L1 Instruction Cache.

- **L2$: L2 Cache**
  - This cache stores the data that is not present in the L1 Cache.
  - It also stores the tags for the data in the L1 Cache.

- **Directory**
  - This component manages the coherence between caches.

- **Router**
  - This component routes requests between the caches and the cores.

These changes in the structure of tiles enable more efficient and direct communication between the caches and the cores.
**DIRECT COHERENCE**
**CHANGES IN THE STRUCTURE OF TILES**

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

- **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.

![Diagram showing changes in the structure of tiles with L1I$, L1D$, L2$, L1C$, and L2C$ labels.](image-url)
DIRECT COHERENCE

Behavior: Cache-to-Cache Read Miss

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: Cache-to-Cache Read Miss**

- 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: CACHE-TO-CACHE READ MISS

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.
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

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

**BEHAVIOR: UPGRADE IN OWNER**

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

**Behavior: Upgrade in Owner**

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

1. GetX

Diagram showing the flow of messages in the write miss process in Direct Coherence (DiCo) protocol.
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

Alberto Ros
Research lines
Uppsala, May 27, 2011
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.

- 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

Diagram showing the process of 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 ownership to another cache.
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 ownership 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.
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**

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

<table>
<thead>
<tr>
<th>Traffic-intensive</th>
<th>Traditional</th>
<th>Indirection-aware</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hammer</td>
<td>Token</td>
</tr>
<tr>
<td>Area-demanding</td>
<td>Directory</td>
<td>DiCo</td>
</tr>
</tbody>
</table>
**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**

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

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

- **Execution time (normalized)**
- **Network traffic (normalized)**
- **Area required (mm²)**

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

DiCo-FM obtains a very good compromise among execution time, network traffic, and area requirements.
Overall Trade-Off

Execution time (normalized)

Network traffic (normalized)

Area required (mm$^2$)

- Hammer
- Directory
- Token

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

**OVERALL TRADE-OFF**

- **Execution time (normalized)**
  - Hammer
  - Directory
  - Token
  - DiCo-FM

- **Network traffic (normalized)**
  - 1.00
  - 0.75
  - 0.50
  - 0.25

- **Area required (mm²)**
  - 9.0
  - 8.5
  - 8.0
  - 7.5
  - 7.0

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


INTERNATIONAL JOURNALS


BOOK CHAPTERS

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.
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.

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.
1. **INTRODUCTION**
   - Challenges in many-core computing

2. **CACHE COHERENCE PROTOCOLS**
   - Direct coherence (DiCo)
   - Coherence deactivation
   - Synchronous coherence

3. **MEMORY HIERARCHY ORGANIZATION**
   - Replacement policies for shared caches
   - Indexing policies for shared caches
   - Impact of NUCA mapping policies on directory scalability

4. **CONCLUSIONS**
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 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.
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.
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**
  - Memory reference to block A
  - A is in private page
  - Cache miss on private block A
  - Non-coherent request issue
  - Cache miss resolved

- **P1**
  - Memory reference to block A
  - A is in private page
  - Cache miss on shared block A
  - Coherent request issue
  - Cache miss resolved

- **MC**
  - Resolve non-coherent miss
  - Override coherence protocol
  - Do not keep track in directory cache

- **OS**
  - A is in private page, but it should be shared
  - Trigger the coherence recovery mechanism
  - A is now in shared page
  - 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 $\Rightarrow$ 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

1. Trigger coherence recovery
2. Page P keeps
3. Keeper P0
4. Wait for pending operations
5. Evict cached blocks of P
6. Unlock page P in TLB
7. Set TLB entry to shared
8. Write data to memory
9. Evictions
10. Recovery request
11. Recovery done
12. End coherence recovery
13. Set page table entry to shared

Alberto Ros
Research lines
Uppsala, May 27, 2011
Coherence Deactivation
Updating-based recovery mechanism

Example of the updating-based recovery

- Lock page P in TLB
- Look for cached blocks of P
- Wait for pending operations
- Set TLB entry to shared
- Unlock page P in TLB
- Recovery request
- Coherence recovery triggered
- Recovery response
- Directory cache updating
- Recovery target done
- Recovery done
- End coherence recovery
- Set page table entry to shared
- Set page P to keeper P0
- Recovery response
- Trigger coherence recovery
- Page P
- Keeper P0
- Initiator P1
- Home MC
- OS

Alberto Ros
Research lines
Uppsala, May 27, 2011
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.
Private blocks (75%) ⇒ Detected private blocks (57%).

57% memory blocks are not tracked.
Coherence Deactivation

**RESULTS: CACHE MISSES**

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

**NORMALIZED NUMBER OF PROCESSOR CACHE MISSES**

![Normalized cache miss rate chart](chart.png)

- 1. Base
- 2. Flushing
- 3. Updating

Alberto Ros
Research lines
Uppsala, May 27, 2011
Similar runtime for flushing and updating recovery mechanisms.

With same directory size $\Rightarrow$ performance improvement: 15%.

With same performance $\Rightarrow$ directory cache 8 times smaller.
Reduced by **40%** when keeping directory cache size.
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


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 programming language?
OUTLINE

1 INTRODUCTION
   - Challenges in many-core computing

2 CACHE COHERENCE PROTOCOLS
   - Direct coherence (DiCo)
   - Coherence deactivation
   - Synchronous coherence

3 MEMORY HIERARCHY ORGANIZATION
   - Replacement policies for shared caches
   - Indexing policies for shared caches
   - Impact of NUCA mapping policies on directory scalability

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

Some authors reduce protocol races by relying on atomic transitions [1].

Another approach: simple request-response protocols.

REFERENCES

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

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? ⇒ Not very efficient.
  - Solution: time-based cache coherence protocols (synchronous coherence).
    - A global clock is needed ⇒ 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

SYNCHRONOUS COHERENCE EXAMPLE

REQUEST-RESPONSE PROTOCOL WITH EXPIRATION DATE FOR CACHED BLOCKS

Cache 0  Cache 1  Cache n

Directory
**Synchronous Coherence Example**

**Request-response protocol with expiration date for cached blocks**

- **Cycle 0**
  - Cache 0
  - Cache 1
  - Cache n
- **Directory**

Request-response protocol with expiration date for cached blocks.
SYNCHRONOUS COHERENCE EXAMPLE

- Directory does not keep list of sharers but expiration date.
**Synchronous Coherence Example**

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

**Request-Response Protocol with Expiration Date for Cached Blocks**

- Cycle 6: Cache 0 (6), Cache 1, Cache n, GetS
- Directory: 6

Alberto Ros

Research lines

Uppsala, May 27, 2011 46 / 63
SYNCHRONOUS COHERENCE EXAMPLE

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

REQUEST-RESPONSE PROTOCOL WITH EXPIRATION DATE FOR CACHED BLOCKS

Cycle 8

- Cache 0
  - Data 4

- Cache 1
  - Data 10

- Cache n

Directory

- Data 10

Alberto Ros
Research lines
Upssala, May 27, 2011 46 / 63
**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**

- **Cycle 0**
  - Cache 0: 2
  - Cache 1: 0
  - Cache n: ...

- **Cycle 2**
  - Data: 10

- **Cycle 6**
  - GetX: 6

- **Cycle 8**
  - Data: 4

- **Cycle 10**
  - GetX: 2

- **Cycle 18**
  - Data: 0

- Directory: 8
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

Cycle 18

Cache 0

Cache 1

Cache n

Directory

0

0

10

Data

10
1 INTRODUCTION
- Challenges in many-core computing

2 CACHE COHERENCE PROTOCOLS
- Direct coherence (DiCo)
- Coherence deactivation
- Synchronous coherence

3 MEMORY HIERARCHY ORGANIZATION
- Replacement policies for shared caches
- Indexing policies for shared caches
- Impact of NUCA mapping policies on directory scalability

4 CONCLUSIONS
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.
1. **INTRODUCTION**
   - Challenges in many-core computing

2. **CACHE COHERENCE PROTOCOLS**
   - Direct coherence (DiCo)
   - Coherence deactivation
   - Synchronous coherence

3. **MEMORY HIERARCHY ORGANIZATION**
   - Replacement policies for shared caches
   - Indexing policies for shared caches
   - Impact of NUCA mapping policies on directory scalability

4. **CONCLUSIONS**
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


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 ⇒ Lock priorities.
  - Give more priority for lock acquisition to slower threads.
1 INTRODUCTION
- Challenges in many-core computing

2 CACHE COHERENCE PROTOCOLS
- Direct coherence (DiCo)
- Coherence deactivation
- Synchronous coherence

3 MEMORY HIERARCHY ORGANIZATION
- Replacement policies for shared caches
- Indexing policies for shared caches
- Impact of NUCA mapping policies on directory scalability

4 CONCLUSIONS
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

**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**

<table>
<thead>
<tr>
<th>Memory addresses requested</th>
<th>LSB indexing</th>
<th>Memory addresses requested</th>
<th>Other-bits indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 3 2 1 0</td>
<td>Bit position</td>
<td>4 3 2 1 0</td>
<td>Bit position</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>0 1 0</td>
<td>0 0 1 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
<td>0 0 0</td>
<td>0 1 0 0 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 1 1 0 0</td>
<td>1 0 0</td>
<td>0 1 1 0 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>1 0 0 0 0</td>
<td>0 0 0</td>
<td>1 0 0 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 1 0 0</td>
<td>1 0 0</td>
<td>1 0 1 0 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>1 1 0 0 0</td>
<td>0 0 0</td>
<td>1 1 0 0 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>1 0 0</td>
<td>1 1 1 0 0</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

- Bits selected to form the index
- Unused set
- Conflict-free set
- Conflicting set

Alberto Ros
Research lines
Uppsala, May 27, 2011
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

1 INTRODUCTION
   - Challenges in many-core computing

2 CACHE COHERENCE PROTOCOLS
   - Direct coherence (DiCo)
   - Coherence deactivation
   - Synchronous coherence

3 MEMORY HIERARCHY ORGANIZATION
   - Replacement policies for shared caches
   - Indexing policies for shared caches
   - Impact of NUCA mapping policies on directory scalability

4 CONCLUSIONS
**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).
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.
Several authors have studied the trade-off between low miss rate and low access time in NUCA caches [6,7].

- But these works do not care about directory scalability.
- They are based on OS allocation policies at page granularity...
- ...which can affect directory scalability.

**REFERENCES**


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).

**REFERENCES**

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).

**REFERENCES**

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).

**REFERENCES**

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).

### REFERENCES

Directory scalability requires block-grained interleaving.

Low latency and miss rate approaches employ page-grained interleaving.

Possible solution ⇒ **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.
1 INTRODUCTION
- Challenges in many-core computing

2 CACHE COHERENCE PROTOCOLS
- Direct coherence (DiCo)
- Coherence deactivation
- Synchronous coherence

3 MEMORY HIERARCHY ORGANIZATION
- Replacement policies for shared caches
- Indexing policies for shared caches
- Impact of NUCA mapping policies on directory scalability

4 CONCLUSIONS
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