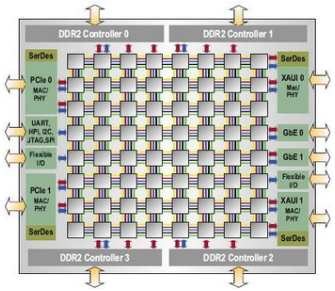
# Advanced Computer Architecture

## Advanced Coherence

**Fall 2016**




**Pejman Lotfi-Kamran**

Adapted from slides originally developed by Profs. Hill, Hoe, Falsafi and Wenisch of CMU, EPFL, Michigan, Wisconsin



## Where Are We?

| Fr | Sa | Su | Mo | Tu |
|---|---|---|---|---|
| | 27-Shahrivar | | 29-Shahrivar | |
| | 3-Mehr | | 5-Mehr | |
| | 10-Mehr | | 12-Mehr | |
| | 17-Mehr | | 19-Mehr | |
| | 24-Mehr | | 26-Mehr | |
| | 1-Aban | | 3-Aban | |
| | 8-Aban | | 10-Aban | |
| | 15-Aban | | 17-Aban | |
| | 22-Aban | | 24-Aban | |
| | 29-Aban | | 1-Azar | |
| | 6-Azar | | 8-Azar | |
| | 13-Azar | | 15-Azar | |
| | 20-Azar | | 22-Azar | |
| | 27-Azar | | 29-Azar | |
| | 4-Dey | | 6-Dey | |

- This Lecture
  - Advanced Coherence
    - Efficient snooping
    - Directories
    - Optimization
- Next Lecture:
  - Consistency



## A Note on L2 Inclusion

- If L2 inclusive
  - On a WB, update copy in L2
  - Upon subsequent reads, data comes from L2
- If L2 non-inclusive
  - A read might find other L1 copies
  - Can get data from L1
  - Or go to the next level



## No snoop, but centralize around directory

- No snoop because of the network is not a point of serialization
- But, make directory the point of serialization
  - Requests for block A are serialized at directory
  - Requests across blocks (e.g., for block B) can overlap
- For example
  - Read miss for A blocks until read completes
  - Read miss for A from another CPU blocks while the first is pending
  - Read miss for B proceeds while A is pending

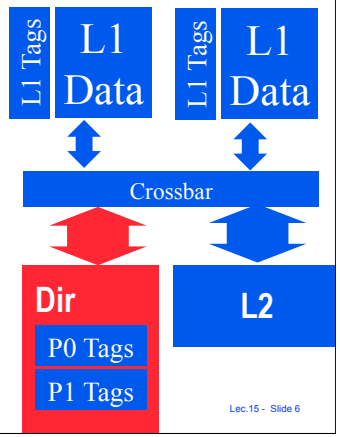
## CMP Directories

- Duplicate Tag
  - Keep a copy of all of L1's Tag Arrays
- Sparse
  - Keep a limited number of Tags
  - FULL: Per-tag keep bit vector
  - Partial: Keep a limited list of sharers



## Duplicate Directories

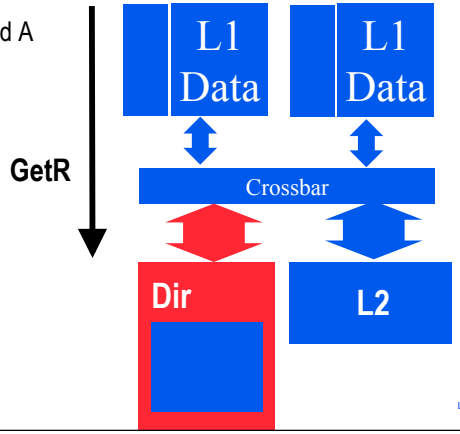
- Keeps a copy of all L1 Tags
  - Compaq Piranha, ARM Elbea Chip (ARM A9)
- Upon request (GetR, GetX, GetUpgr)
  - Searches all L1 Tags
  - if found, MSI transitions
  - If not, go to L2






## Example

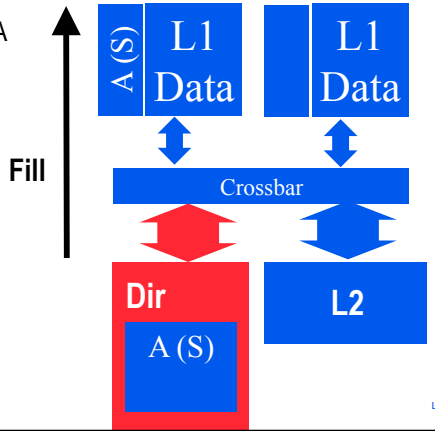
- P0 Read A





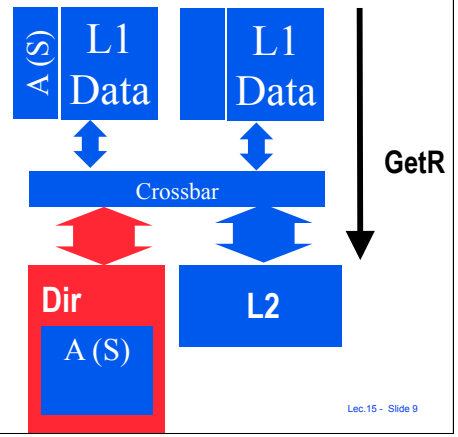

## Example

- P0 Read A



## Example

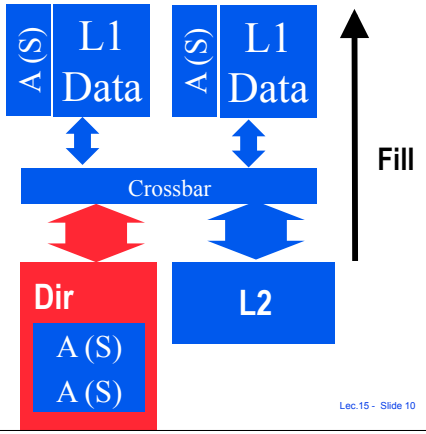
- P0 Read A
- P1 Read A






## Example

- P0 Read A
- P1 Read A

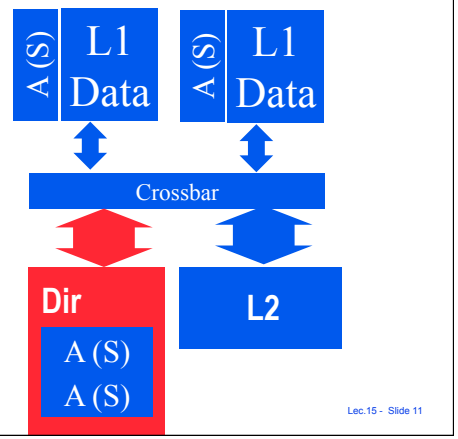





## Example

- P0 Read A
- P1 Read A
- P0 Write A

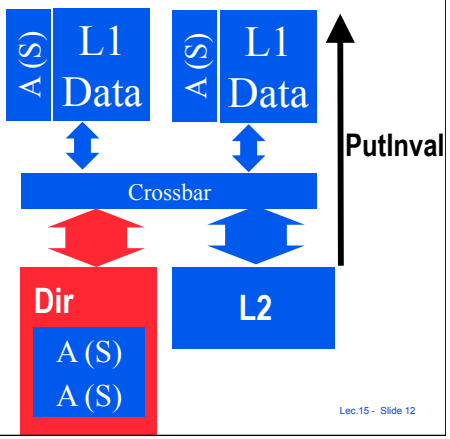
GetUpgr





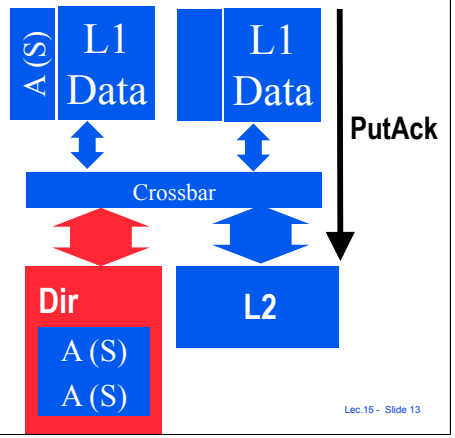

## Example

- P0 Read A
- P1 Read A
- P0 Write A



## Example

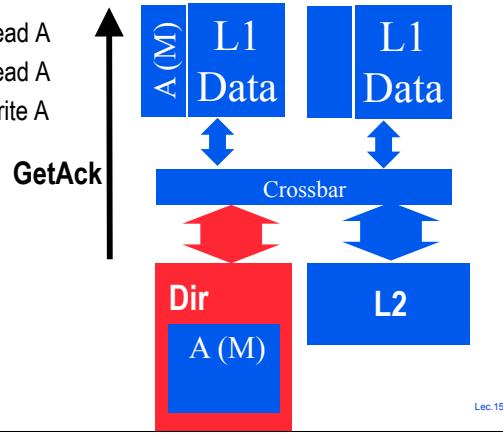
- P0 Read A
- P1 Read A
- P0 Write A






## Example

- P0 Read A
- P1 Read A
- P0 Write A






## A Note on Deadlock

- Transactions are:
  - Non-atomic and interleaved
- In the case of a general-purpose NoC
  - To avoid deadlock
  - Use request/response virtual channels
  - As long as requests are serviced while waiting, deadlock can be avoided

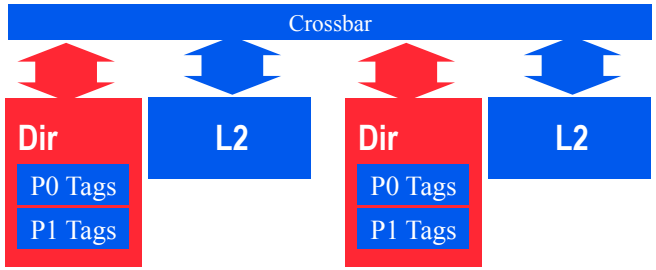


## Duplicate Tags Problem (1)

- Centralized
  - A centralized structure is not much better than the bus
  - Directory becomes a bottleneck
  - But, large L2's are banked (multi-banked, address-interleaved)
  - Solution: distribute directories with L2 banks

## Distributed Directories

- Bank based on block address
- Example: Odd address (1), even addresses (2)






## Distributed directories

- Distribute the traffic across banks
- But suffers from the same problems
  - Bank conflict if burst of accesses to one bank
- As with caches, in practice works well
  - E.g., Compaq Piranha chip, 8 L2 + directory banks



## Duplicate Tags Problem (2)

- Don't scale!
- Assume 4-way L1's
- Each directory search is N times 4 ways
- For two L1's (two cores) ➔8 ways to search
- For 32 L1's (32 cores) ➔128 ways to search
  - Bottleneck in latency
  - Bottleneck in energy



## Sparse Directories

- Do not keep all tags
- Keep a cache of tags
- Limited/fixed associativity
  - Invalidate blocks in L1 which you can't track in directory
- Represent duplicate tags with a bit vector
  - Copies of one tag have bits set in vector

## Example: Duplicate Tag

- P0 has cached blocks A (shared) and B (modified)
- P1 has cached blocks A (shared) and C (shared)
- A, B and C all fall in the same set in L1's


- All entries for one L1 set map to the same set in directory
- Search associatively



## Example: Sparse (3-way associative)

P0 Read A




## Example: Sparse (3-way associative)

P0 Read A
P1 Read C




## Example: Sparse (3-way associative)

P0 Read A
P1 Read C
P1 Read A

## Example: Sparse (3-way associative)

P0 Read A
P1 Read C
P1 Read A
P0 Write B

**Sparse Tag (3-way)**

| | | |
|---|---|---|
| A(S)[11] | C(S)[10] | B(M)[01] |



## Example: Sparse (2-way associative)

P0 Read A

**Sparse Tag (2-way)**

| | |
|---|---|
| A(S)[01] | |



## Example: Sparse (2-way associative)

P0 Read A
P1 Read C

**Sparse Tag (2-way)**

| | |
|---|---|
| A(S)[01] | C(S)[10] |



## Example: Sparse (2-way associative)

P0 Read A
P1 Read C
P1 Read A

**Sparse Tag (2-way)**

| | |
|---|---|
| C(S)[10] | A(S)[11] |

## Example: Sparse (2-way associative)

P0 Read A
P1 Read C
P1 Read A
P0 Write B (invalidate C from P1)




## Summary

- Buses don't scale
  - Use directories
  - Point-to-point messages
  - Serialize per address at directory
- Centralized directories are a bottleneck
  - Distribute
- Can't keep all tags (duplicates)
  - Sparse directories



## Roadmap

- Snoopy optimization
- Directory protocols
- Protocol optimization
- → Directory optimization

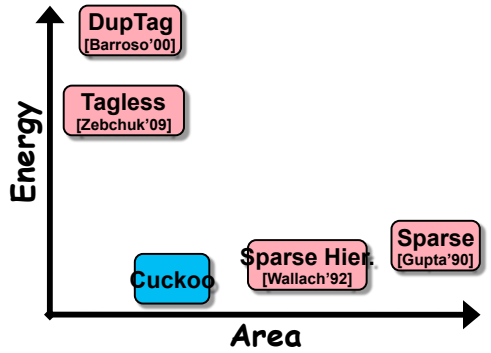


## Scaling Coherence Directories

- CMPs heading to high core counts
  - High performance requires private caches
  - Private caches require coherence
  - Coherence limited by directory conflicts
- Traditional ways to reduce conflicts
  - Increase associativity (Dup-Tag) → pay Power
  - Over-provision sets (Sparse) → pay Area

***Want low-power and area-efficient organization***

## Directory Efficiency




***Cuckoo Directory: low power, small area***

## Cuckoo Directory Organization

- Inspired by Cuckoo Hash [Pagh'01, Fotakis'03]
- Re-locates conflicting entries
  - Cuckoos put eggs in others' nests
  - Move "native" eggs to make space
- Eliminates practically all conflicts
  - Low associativity (3-way)
  - Small area



## Contributions

- Leverage Cuckoo Hash organization
  - Algorithm to re-locate conflicting entries
  - 3-way, but behaves like fully-associative
- Cuckoo Directory
  - More power-efficient than Dup-Tag
  - More area-efficient than Sparse
  - Scalable to 1000+ cores

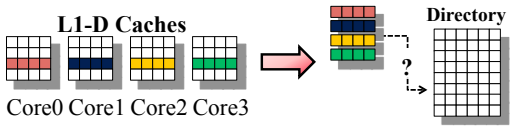
***Cuckoo Directory: a truly scalable directory***

## Outline

- Introduction
- Directory Conflicts and Solutions
- Cuckoo Directory
- Evaluation
- Conclusions

## Origin of the Conflict Problem

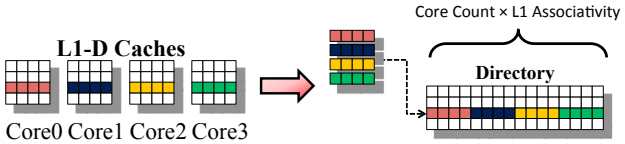
- Directory maps all blocks in private caches
- Low-order bits in “same” set identical
- Entries map to same directory indexes
  - either… increase associativity with each core
  - or… over-provision to spread across sets






## Duplicate-Tag Directory

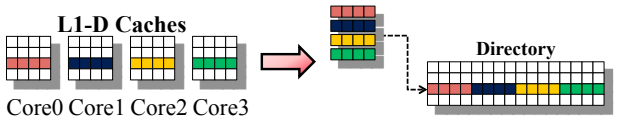
- Ensure sufficient space for worst-case
  - Allocate space for each way of each cache
- High core counts → huge associativity




***Unmanageable energy use***

## Sparse Directory

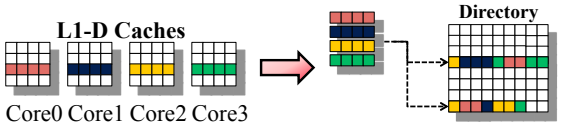
- Split L1 sets into directory sets
  - Grow number of bits in index





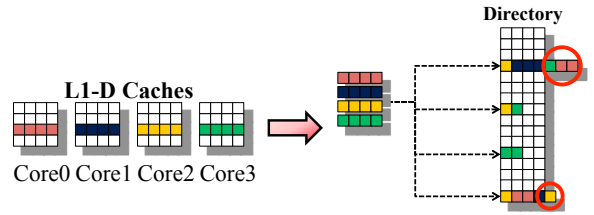

## Sparse Directory

- Split L1 sets into directory sets
  - Grow number of bits in index



## Sparse Directory

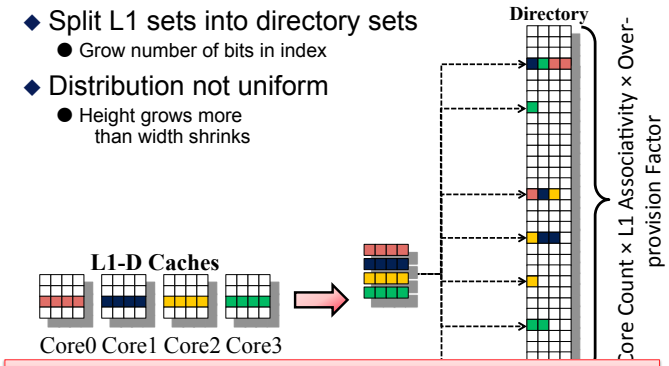
- Split L1 sets into directory sets
  - Grow number of bits in index
- Distribution not uniform






## Sparse Directory

- Split L1 sets into directory sets
  - Grow number of bits in index
- Distribution not uniform
  - Height grows more than width shrinks




***Leads to large area overheads***

## Methodology

- ***Flexus*** simulation infrastructure [Wenisch'06]
- Full-system trace (OS + user) evaluation

**Benchmark Applications**

- OLTP: TPC-C
  - IBM DB2 & Oracle
- DSS: TPC-H
  - IBM DB2 Qrys 2, 16, 17
- Web: SPECweb99
  - Apache & Zeus
- Scientific
  - em3d & ocean

**Model Parameters**

- L1 (I & D) : 64KB, 2-way
- L2: 16MB (1MB per core)

**Area Model**

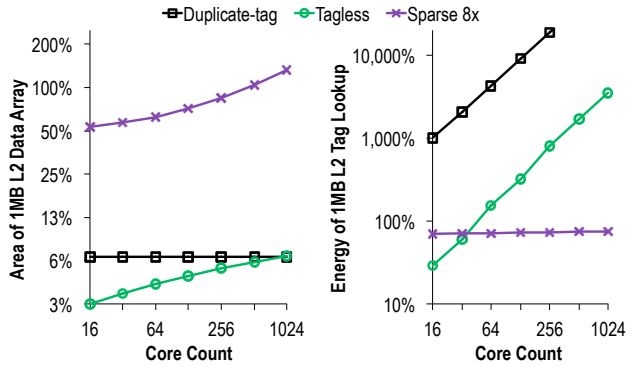
- Total bit count

**Energy Model**

- Total bit accessed
- Reads and Writes equal



## Area and Energy Comparison




***No design is both area- and power-efficient***

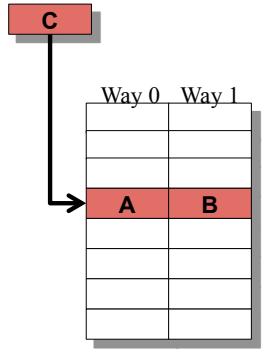
## Outline

- Introduction
- Directory Conflicts and Solutions
- Cuckoo Directory
- Evaluation
- Conclusions



## Sparse Directory

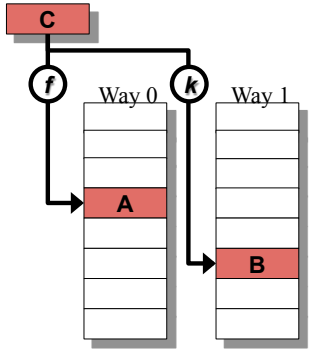
- C conflicts with A
- C conflicts with B
- Therefore…
  - A, B, C can't coexist
  - Even if space available




***To insert C, directory must evict A or B***

## Cuckoo Directory

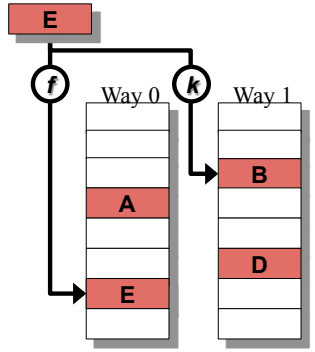
- Skewed structure [Seznec'93]
  - Hash each way's index
- ***KEY IDEA***
  - Move conflicting entries
  - Inserting C moves A




***A, B, and C can coexist***

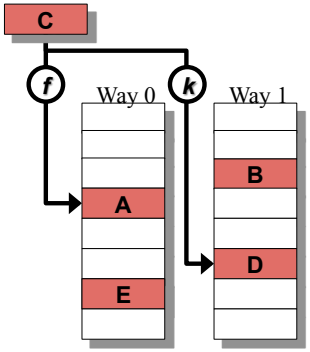
## Example 1: Common Case

- Case
  - f(E) or k(E) available
- Action
  - Insert immediately



## Example 2: Uncommon Case

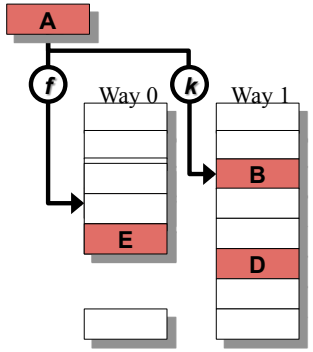
- Case
  - f(C) and k(C) occupied
- Action
  - Place C, replacing A
  - Place A, replacing B
  - Place B






## Example 2: Uncommon Case

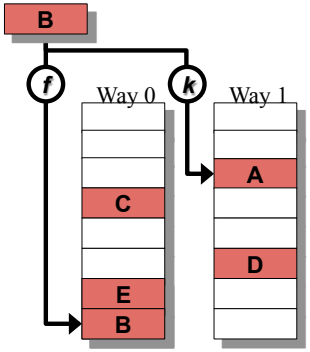
- Case
  - f(C) and k(C) occupied
- Action
  - Place C, replacing A
  - Place A, replacing B
  - Place B






## Example 2: Uncommon Case

- Case
  - f(C) and k(C) occupied
- Action
  - Place C, replacing A
  - Place A, replacing B
  - Place B

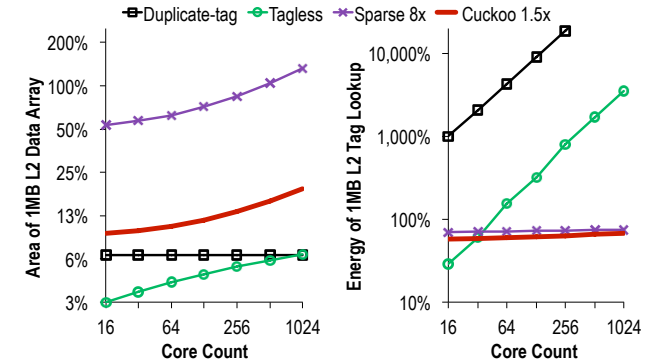



***All entries fit in directory without eviction***

## Outline

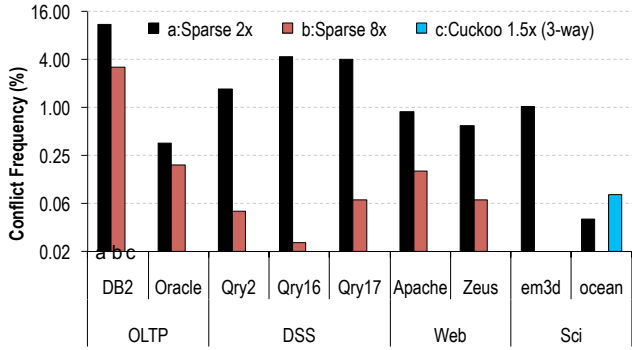
- Introduction
- Directory Conflicts and Solutions
- Cuckoo Directory
- Evaluation
- Conclusions

## Area and Energy Comparison




***Cuckoo Directory is area- AND power-efficient***

## Directory Conflict Frequency




***0.08% conflicts in worst-case benchmark***

## Conclusions

- Cuckoo Directory
  - Moves conflicting entries across ways
  - More power-efficient than Dup-Tag & Tagless
  - More area-efficient than Sparse & Coarse
- Scalable to 1000+ cores
  - Constant associativity for any core count
  - Minimal capacity over-provisioning

***Cuckoo Directory: a truly scalable directory***