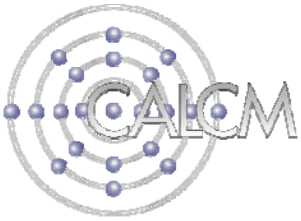
Reactive NUCA: Near-Optimal Block Placement and Replication in Distributed Caches

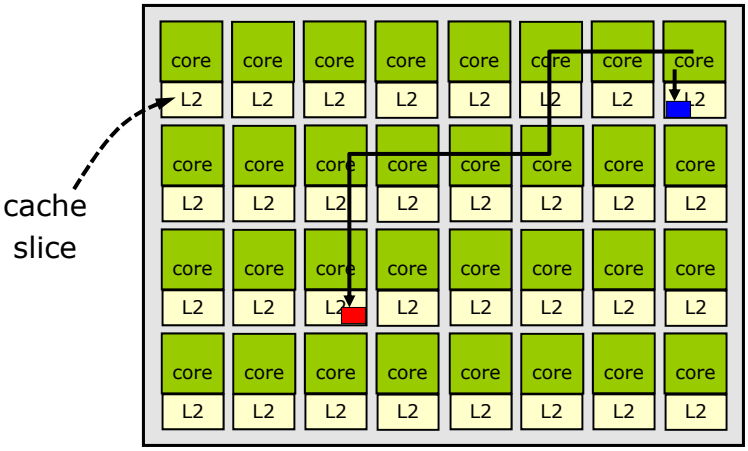
Nikos Hardavellas

Michael Ferdman, Babak Falsafi, Anastasia Ailamaki
Carnegie Mellon and EPFL



Databases @ Carnegie Mellon

Data Placement in Distributed Caches



- Data placement determines performance
- Goal: place data on chip close to where they are used

Prior Work

- Several proposals for CMP cache management
  - ASR, cooperative caching, victim replication, CMP-NuRapid, D-NUCA
- ...but suffer from shortcomings
  - complex, high-latency lookup/coherence
  - don't scale
  - lower effective cache capacity
  - optimize only for subset of accesses

We need:

➡ Simple, scalable mechanism for fast access to all data

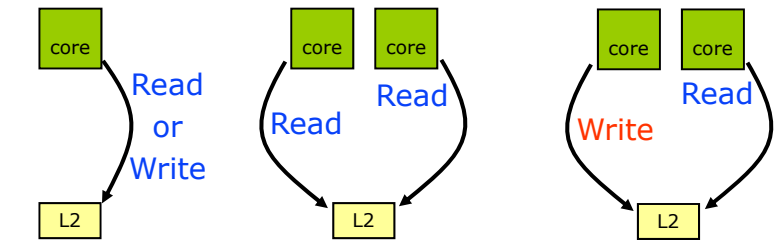
Our Proposal: Reactive NUCA

- Cache accesses can be classified at run-time
  - Each class amenable to different placement
- Per-class block placement
  - Simple, scalable, transparent
  - No need for HW coherence mechanisms at LLC
  - Avg. speedup of 6% & 14% over shared & private
    - Up to 32% speedup
    - -5% on avg. from ideal cache organization
- Rotational Interleaving
  - Data replication and fast single-probe lookup

Outline

- Introduction
- Access Classification and Block Placement
- Reactive NUCA Mechanisms
- Evaluation
- Conclusion

Terminology: Data Types



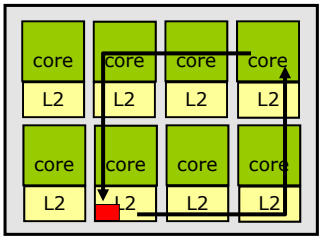
Private

Shared Read-Only

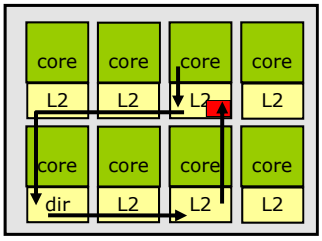
Shared Read-Write

Conventional Multicore Caches

Shared



Private



Addr-interleave blocks

+ High effective capacity
– Slow access

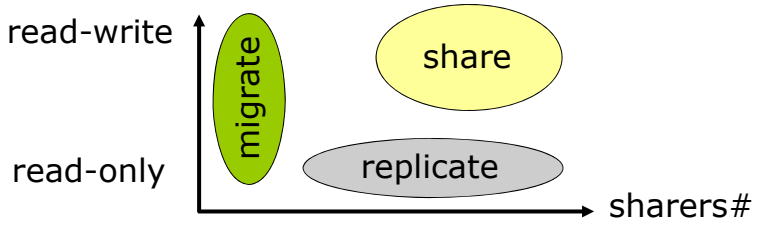
Each block cached locally

+ Fast access (local)
– Low capacity (replicas)
– Coherence: via indirection (distributed directory)

➧ We want: high capacity (shared) + fast access (priv.)

Where to Place the Data?

- Close to where they are used!
- Accessed by single core: migrate locally
- Accessed by many cores: replicate (?)
  - If read-only, replication is OK
  - If read-write, coherence a problem
    - Low reuse: evenly distribute across sharers



Methodology

Flexus: Full-system cycle-accurate timing simulation

Workloads

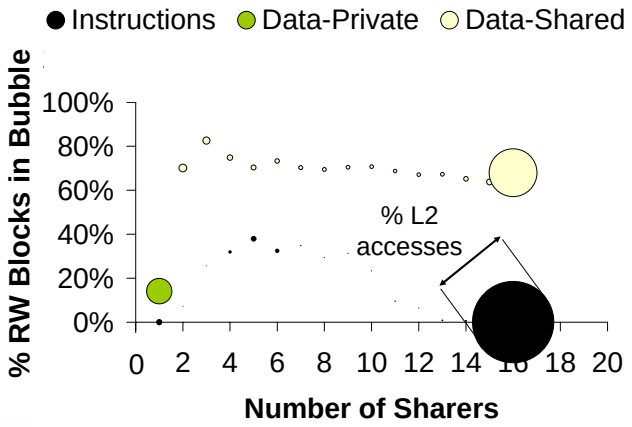
- OLTP: TPC-C 3.0 100 WH
  - IBM DB2 v8
  - Oracle 10g
- DSS: TPC-H Qry 6, 8, 13
  - IBM DB2 v8
- SPECweb99 on Apache 2.0
- Multiprogammed: Spec2K
- Scientific: em3d

Model Parameters

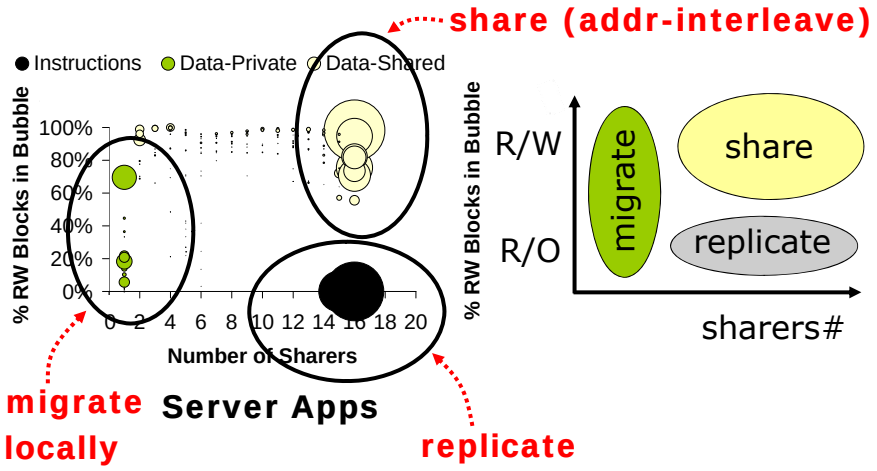
- Tiled, LLC = L2
- Server/Scientific wrkld.
  - 16-cores, 1MB/core
- Multi-programmed wrkld.
  - 8-cores, 3MB/core
- OoO, 2GHz, 96-entry ROB
- Folded 2D-torus
  - 2-cycle router
  - 1-cycle link
- 45ns memory

Cache Access Classification Example

- Each bubble: cache blocks shared by x cores
- Size of bubble proportional to % L2 accesses
- y axis: % blocks in bubble that are read-write



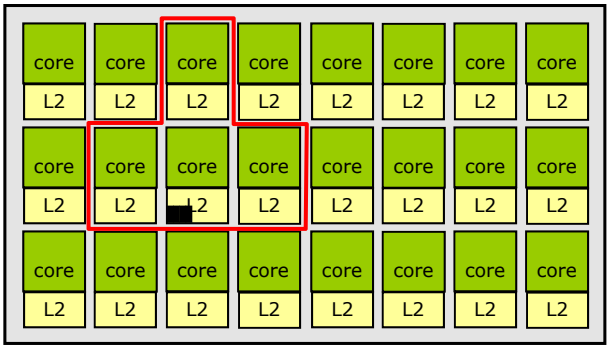
Cache Access Clustering



- Accesses naturally form 3 clusters

Instruction Replication

- Instruction working set too large for one cache slice



- Distribute in cluster of neighbors, replicate across

Outline

- Introduction
- Access Classification and Block Placement
- Reactive NUCA Mechanisms
- Evaluation
- Conclusion

Rotational Interleaving

RID → +1

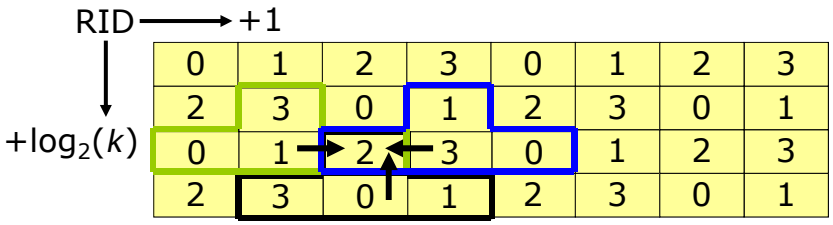
$+\log_2(k)$ ↓

0	1	2	3	0	1	2	3
2	3	0	1	2	3	0	1
0	1	2	3	0	1	2	3
2	3	0	1	2	3	0	1

size-4 clusters:
local slice + 3 neighbors

- Fast access (nearest-neighbor, simple lookup)
- Balance access latency with capacity constraints
- Equal capacity pressure at overlapped slices

Rotational Interleaving

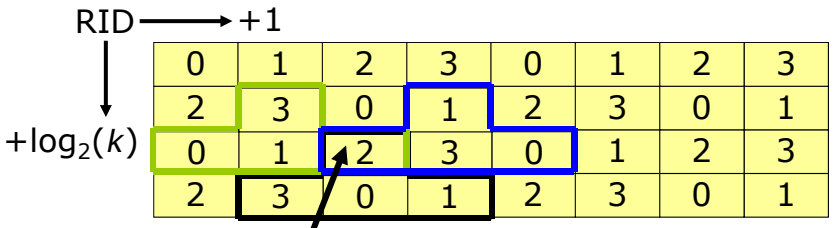


PC: 0xfa480

$$Destination = \left(Addr + \overline{RID} + 1\right) \& (n-1)$$

- Fast access (nearest-neighbor, simple lookup)
- Balance access latency with capacity constraints
- Equal capacity pressure at overlapped slices

Rotational Interleaving

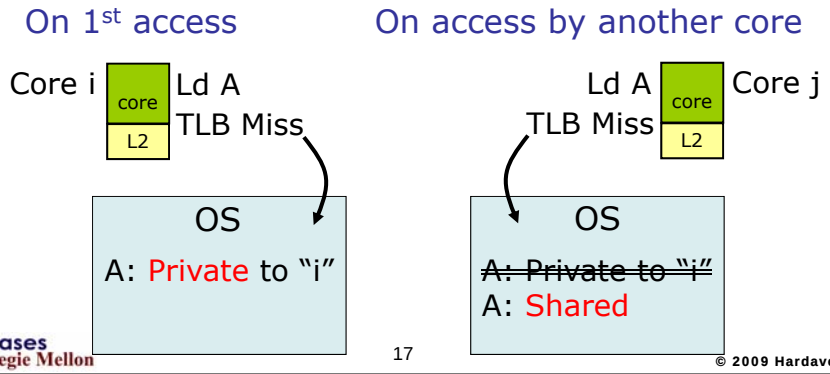


each slice caches the same blocks
on behalf of any cluster

- Fast access (nearest-neighbor, simple lookup)
- Balance access latency with capacity constraints
- Equal capacity pressure at overlapped slices

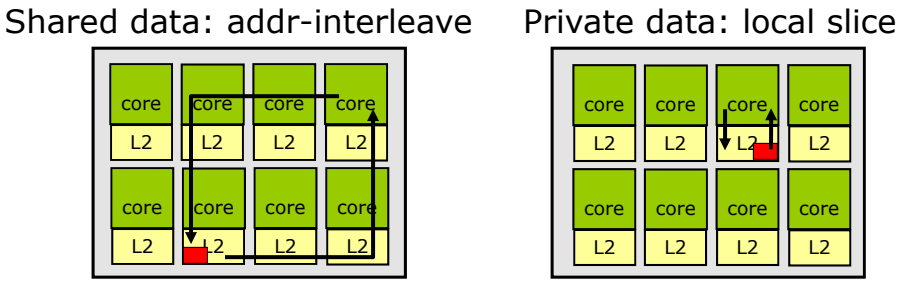
Classification Mechanisms

- Instructions classification: all accesses from L1-I
- Per-page classification for data: at TLB miss
  - Utilize OS page table & TLB for book-keeping info
  - Page classification is accurate (<0.5% error)



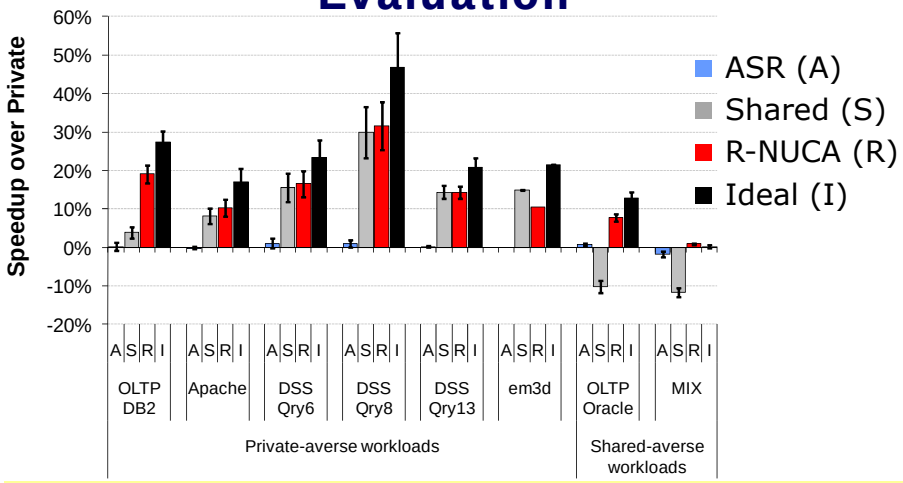
Coherence: No Need for HW Mechanisms at LLC

- Reactive NUCA placement guarantee
  - Each R/W datum in unique & known location



➧ Fast access, eliminates HW overhead

Evaluation



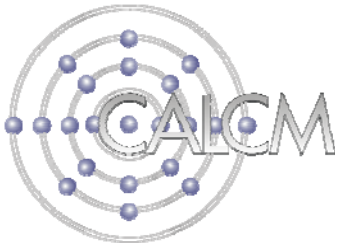
- **Delivers robust performance across workloads**
  - Shared: same for Web, DSS; **17%** for OLTP, MIX
  - Private: **17%** for OLTP, Web, DSS; same for MIX

Conclusions

Reactive NUCA: near-optimal block placement and replication in distributed caches

- Cache accesses can be classified at run-time
  - Each class amenable to different placement
- Reactive NUCA: placement of each class
  - Simple, scalable, low-overhead, transparent
  - Obviates HW coherence mechanisms for LLC
- Rotational Interleaving
  - Replication + fast lookup (neighbors, single probe)
- Robust performance across server workloads
  - Near-optimal placement (-5% avg. from ideal)

Questions ?



Impetus Group
Computer Architecture Lab (CALCM)
Carnegie Mellon University
www.ece.cmu.edu/CALCM/

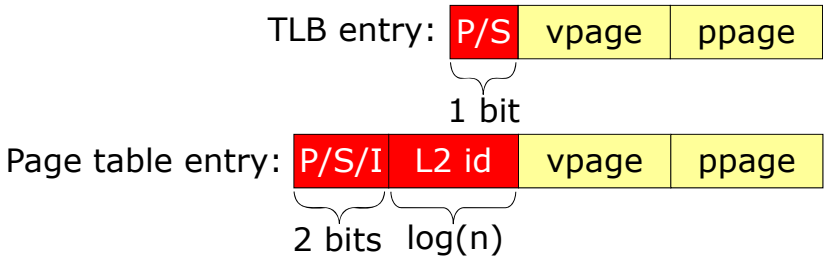
Flexus full-system simulator available at
www.ece.cmu.edu/~simflex/

Backup Slides

Classification and Lookup

Classification Granularity: OS Page

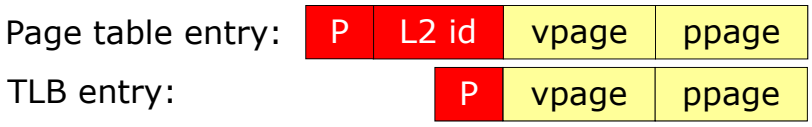
- Per-block classification: high area/latency overhead
- Per-page classification (utilize OS page table)
  - Core accesses it for every access anyway (TLB)
  - Page classification is accurate (<0.5% error)



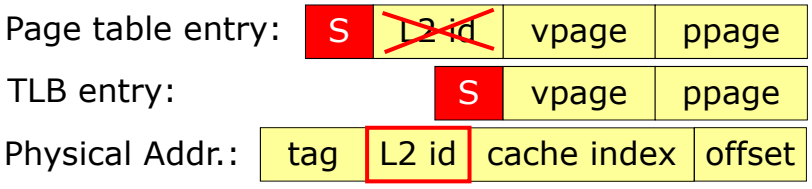
➧ Page granularity allows simple + practical HW

Data Class Bookkeeping

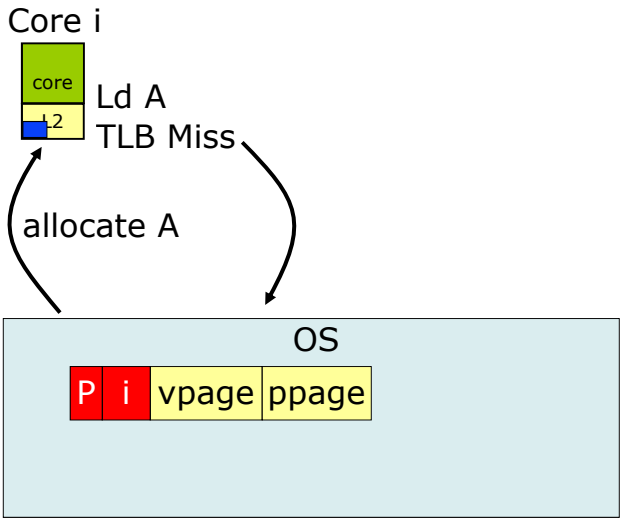
- **private data**: place in local L2 slice



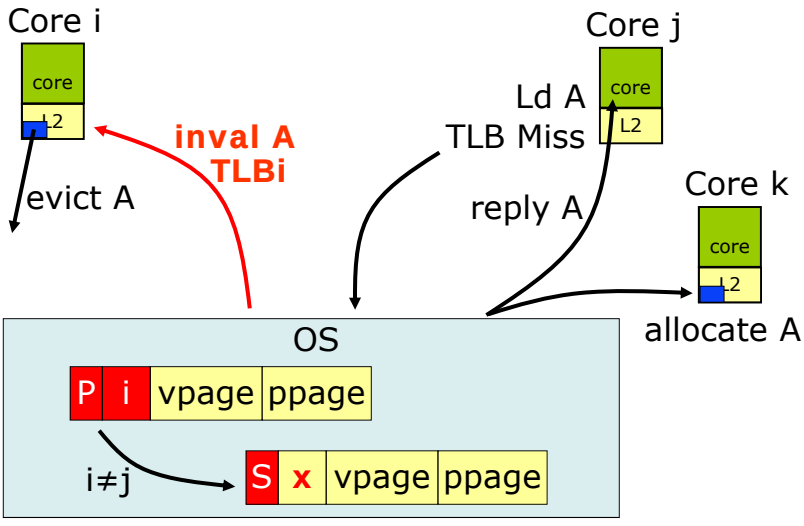
- **shared data**: place in aggregate L2 (addr interleave)



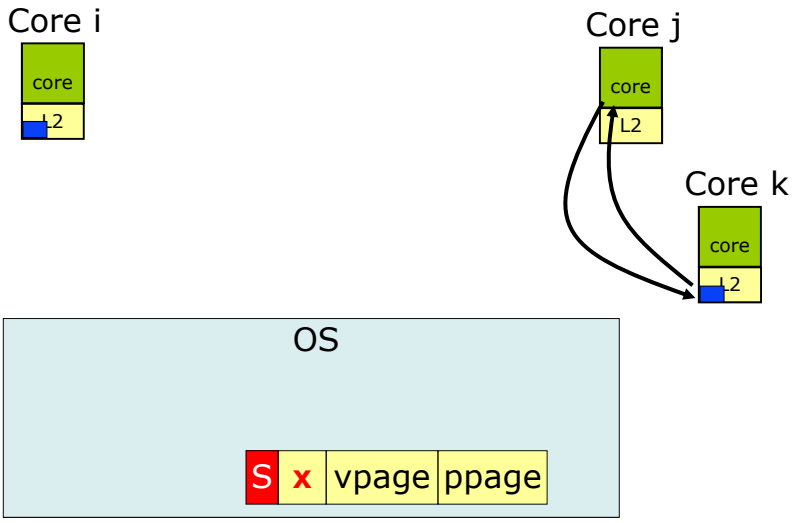
Data Classification and Lookup



Data Classification and Lookup

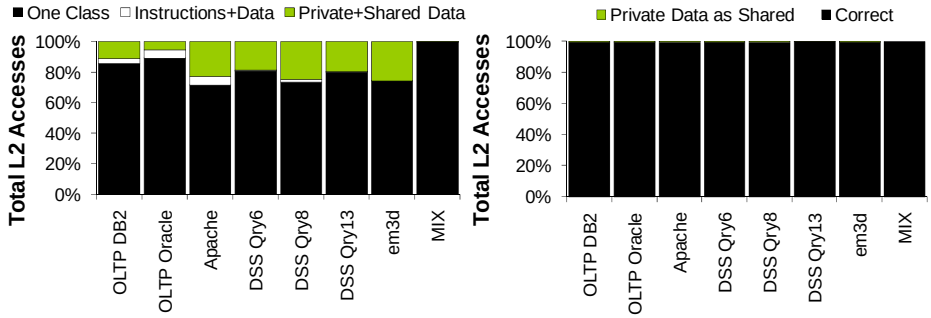


Data Classification and Lookup



➧ Fast & simple lookup for data

Misclassifications at Page Granularity



Accesses from pages with multiple access types

Access misclassifications

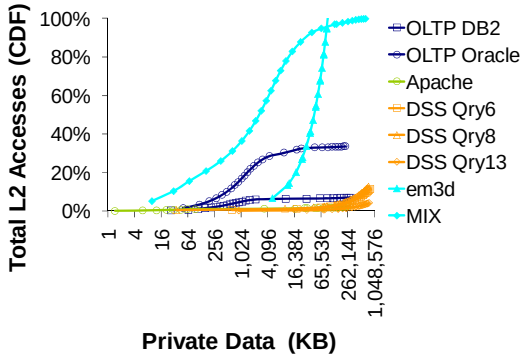
- A page may service multiple access types
- But, one type always dominates accesses

➧ Classification at page granularity is accurate

Backup Slides

Placement

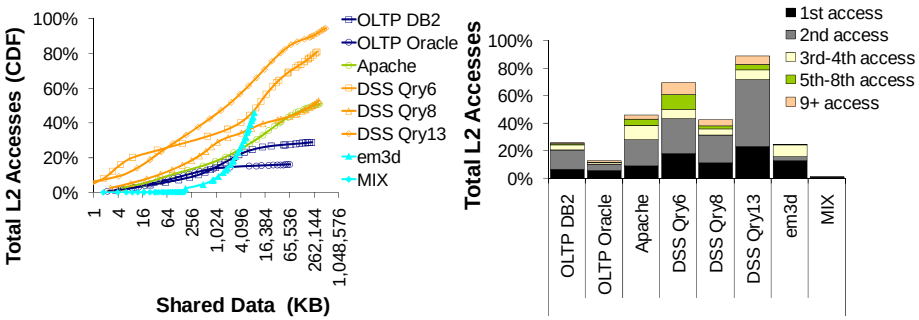
Private Data Placement



- Spill to neighbors if working set too large?
  - NO!!! Each core runs similar threads

➧ Store in local L2 slice (like in private cache)

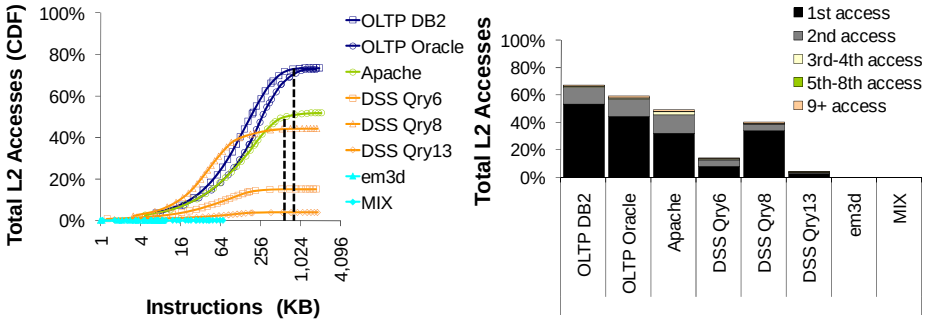
Shared Data Placement



- Read-write + large working set + low reuse
  - Unlikely to be in local slice for reuse
- Also, next sharer is random [WMPI'04]

➡ Address-interleave in aggregate L2 (like shared cache)

Instruction Placement



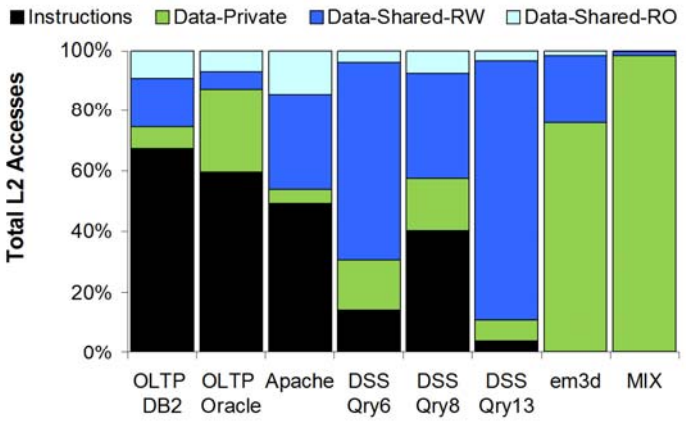
- Working set too large for one slice
  - Slices store private & shared data too!
  - Sufficient capacity with 4 L2 slices

➡ Share in clusters of neighbors, replicate across

Backup Slides

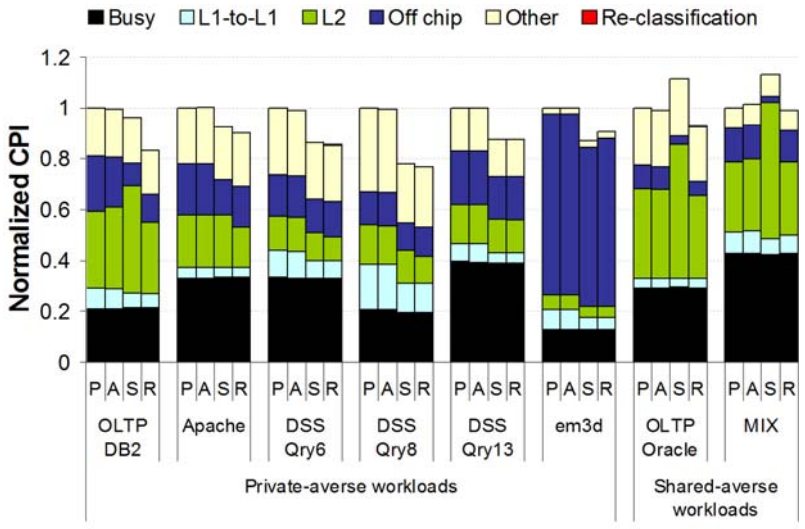
Detailed Evaluation

Cache Accesses Breakdown

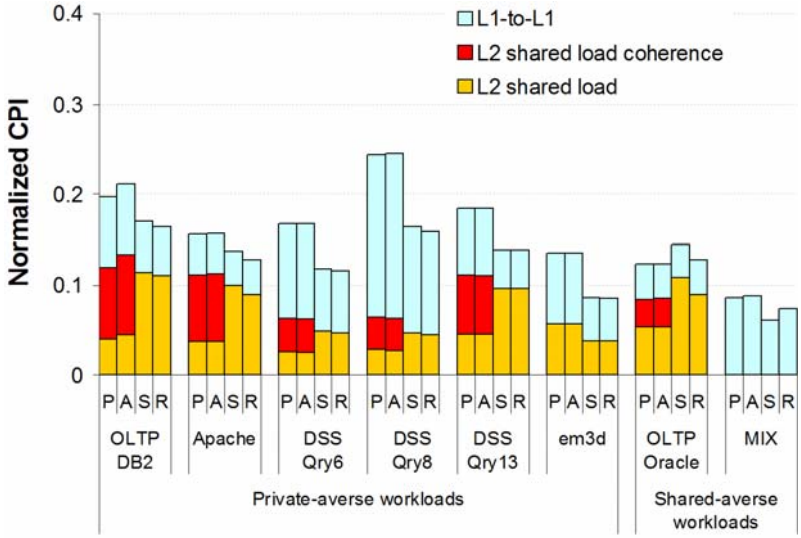


- Instr. + shared-RW dominate server workloads
- Private dominate scientific/MIX

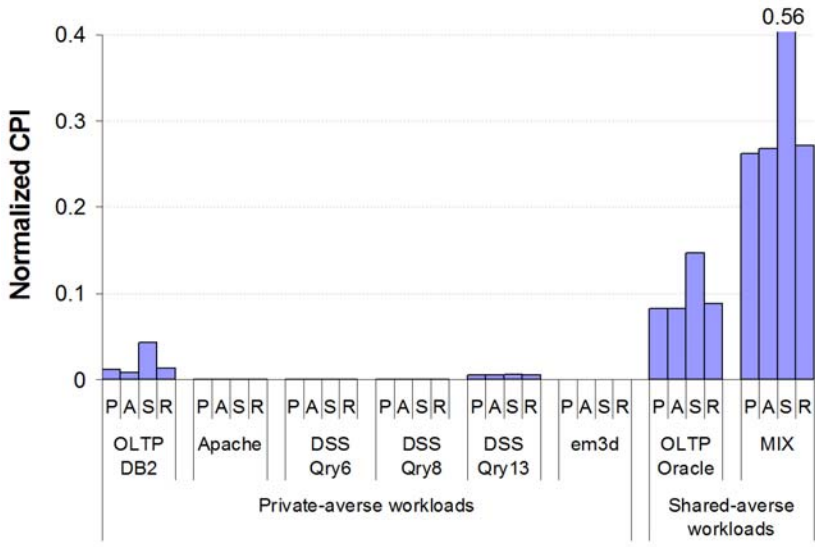
CPI Breakdown



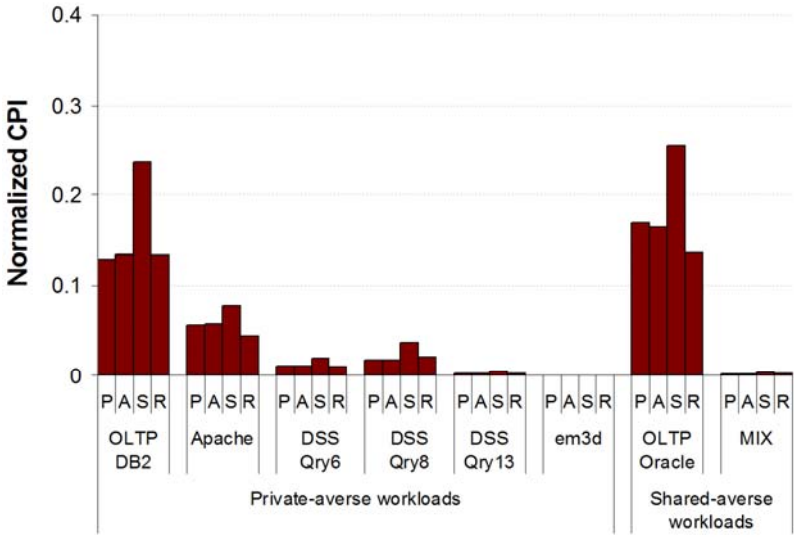
Impact of Eliminating L2 Coherence



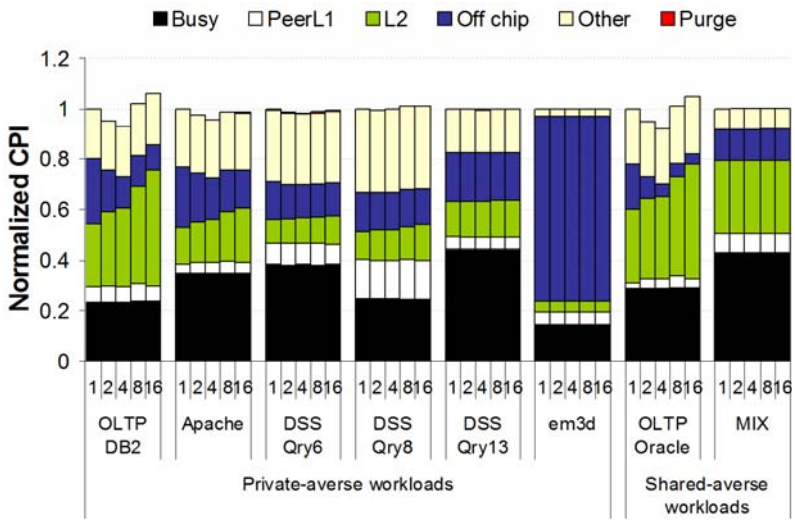
Impact of Private Allocation



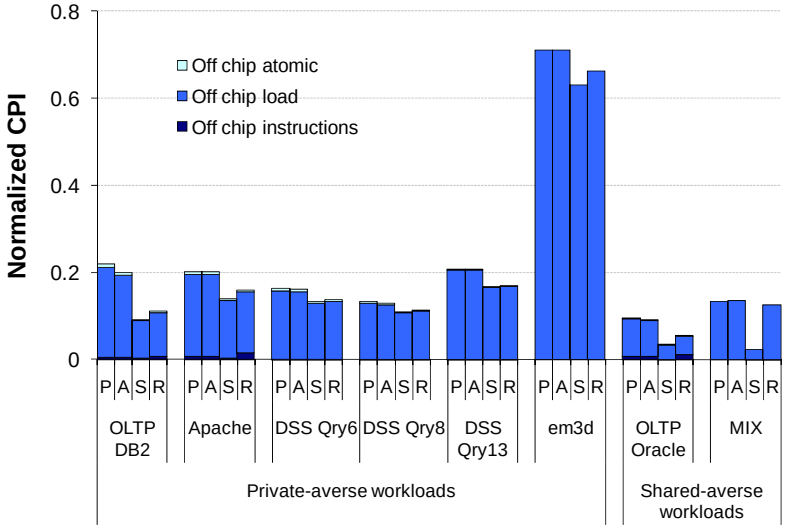
Impact of Instruction Replication



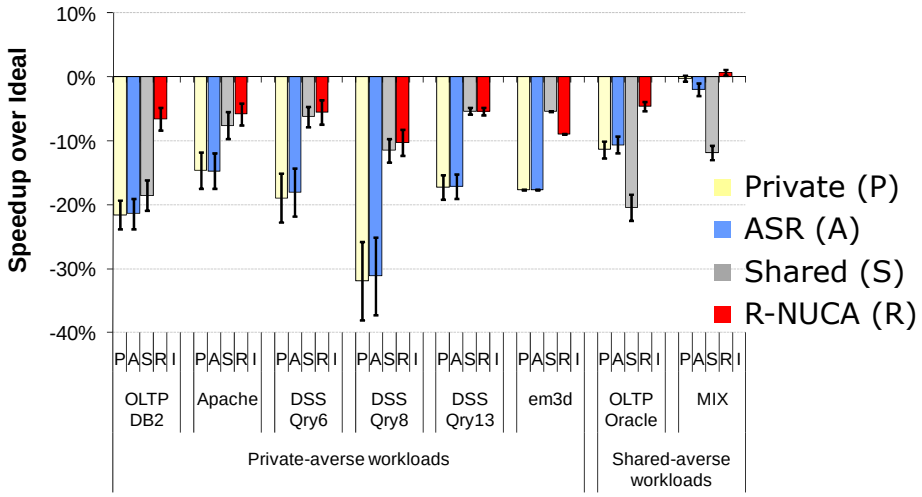
Instruction Clustering



Off-chip Misses



Evaluation: Speedup over Ideal



- Near-optimal placement: -5% on avg. from ideal