



# NTNU

Det skapende universitet

## **Cooperative Caching for Chip Multiprocessors (CC)**

2. Motivations
3. Concepts
4. Implementation
5. Evaluation

# Motivations

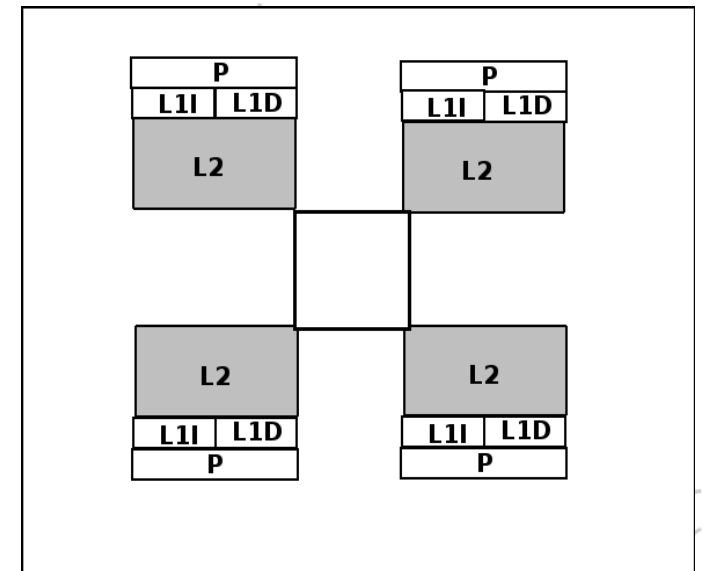
- Higher demands
- Increasing cost of off-chip misses
- On-Chip wire delay
- Shared L2 minimizes the overall miss rate.
- Private L2 reduces the access latency and complexity
- Reuse multi-CPU paradigms

# Why CC

- More self-contained.
  - Easier to manage as an single unit in case of resource management.
- Easier to implement performance isolation, priority and QoS
- Reduces the cross-chip interconnect, and decreases the complexity and power consumption.

# What is CC

- Each core's L2 close in locality
- Privately owned
- In case of L2 misses, possible to transfer from other L2 cache
- Remotely accessible data
- Reduce off-chip access



# Policies for reduction of off-chip access

- Cache-to-cache Transfer
  - In case of miss transfer “clean” blocks from another L2 cache
  - Clean data because one are more likely to share “read-only” data ( Instructions )

- Replication-aware Data Replacement

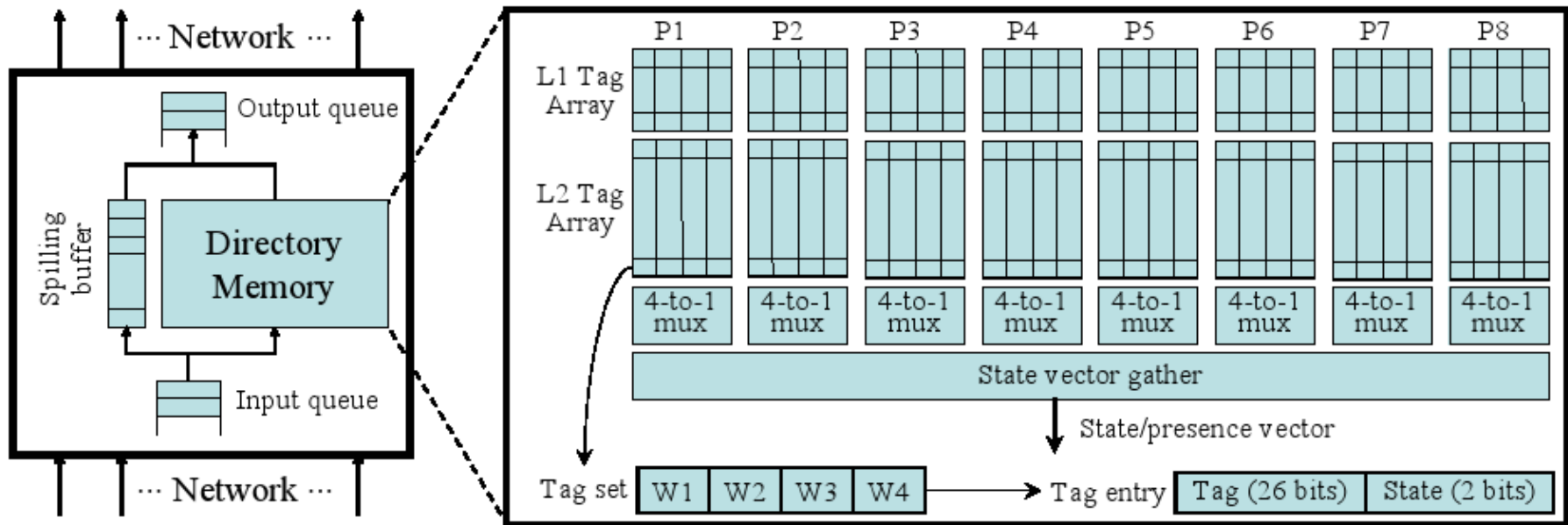
- Singlet – One chip only
- Replicate – Replication exists
- Evict singlets only when no replicates exists. Use LRU or other cache replacement strategy of choice.
- Singlets can be “spilled” to other cache banks

- Global Replacement of Inactive data
  - “Spilling” requires global management
  - N-Chance Forwarding
    - When discarding singlet block, spill to another random peer cache.
    - Set recirculation count to N when victimized
    - Decrease N by 1 when spilled again, unless N becomes 0.
    - If reused reset N to 0

# Implementation

- N-Forwarding and singlet/replicate can be solved by adding two bits to tag. ( 1-Forwarding and singlet )
- L2 can implement a snooping protocol to keep concurrent singlet bit.
- Spilling can be implemented via “Push” or “Pull”
- Chang and Sohi have implemented a central Directory to speed up the communication

# Central Coherence Engine



source: Cooperative Caching for Chip Multiprocessors

# Evaluation – Miss Rate

**Table 5. Multithreaded Workload Miss Rate and L1 Miss Breakdown**

	Thousand misses per transaction Off-chip (Private / Shared / CC)	L1 Misses breakdown (Private / Shared / CC)		
		Local L2	Remote L2	Off-chip
OLTP	9.75 / 3.10 / 3.80	90% / 15% / 86%	7% / 84% / 13%	3% / 1% / 1%
Apache	1.60 / 0.90 / 0.94	65% / 9% / 51%	15% / 77% / 36%	20% / 14% / 13%
JBB	0.13 / 0.08 / 0.10	72% / 10% / 57%	14% / 80% / 32%	14% / 10% / 11%
Zeus	0.71 / 0.46 / 0.49	67% / 9% / 45%	15% / 78% / 41%	19% / 12% / 13%

# Miss Rate Spec2000

**Table 6. Multiprogrammed Workload Miss Rate and L1 Miss Breakdown**

	Misses per thousand instructions Off-chip (Private / Shared / CC)	L1 Misses breakdown (Private / Shared / CC)		
		Local L2	Remote L2	Off-chip
Mix1	3.1 / 2.0 / 2.4	78% / 19% / 67%	3% / 73% / 22%	19% / 9% / 11%
Mix2	3.0 / 1.6 / 1.8	64% / 35% / 75%	4% / 55% / 14%	32% / 9% / 11%
Mix3	1.2 / 0.7 / 0.8	91% / 20% / 87%	1% / 77% / 9%	7% / 3% / 4%
Mix4	0.6 / 0.3 / 0.3	95% / 12% / 90%	0% / 86% / 8%	4% / 2% / 2%
Rate1	0.8 / 0.6 / 0.8	90% / 20% / 80%	3% / 76% / 13%	7% / 4% / 6%
Rate2	53 / 51 / 41	31% / 7% / 24%	11% / 47% / 34%	58% / 46% / 42%

Differs in:

- Little data shared between threads
- Most L1 misses are satisfied by local L2

# Overall

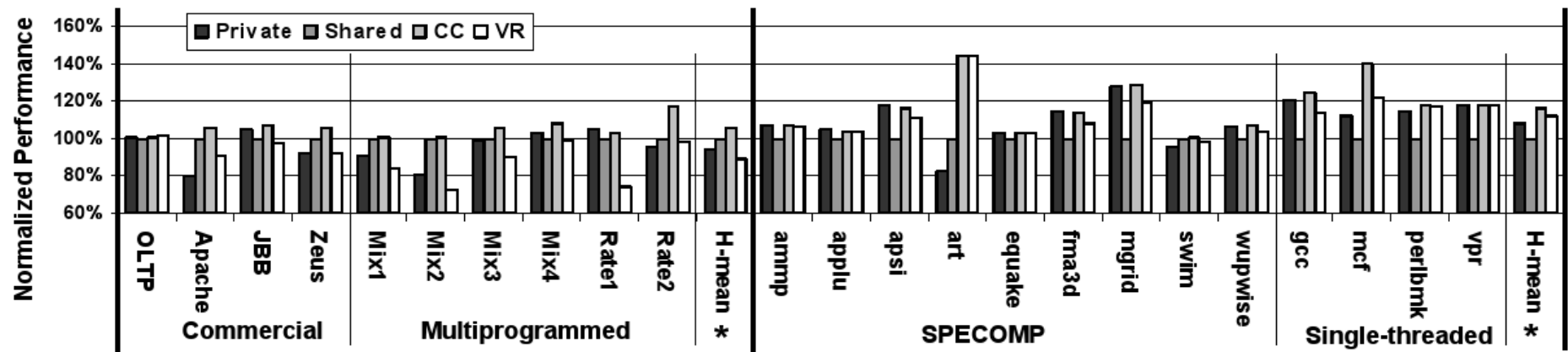


Figure 11. Comparison with Victim Replication

# Endnote

Our simulation shows that using cooperative caching achieves the best performance for different CMP configurations and workloads