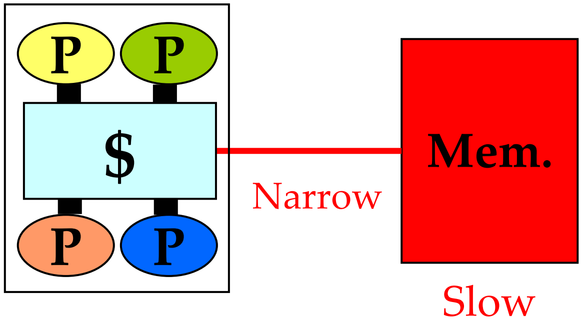
Cooperative Cache Partitioning for Chip Multiprocessors

Jichuan Chang
Guri Sohi
University of Wisconsin—Madison

ICS-21, 6/20/2007

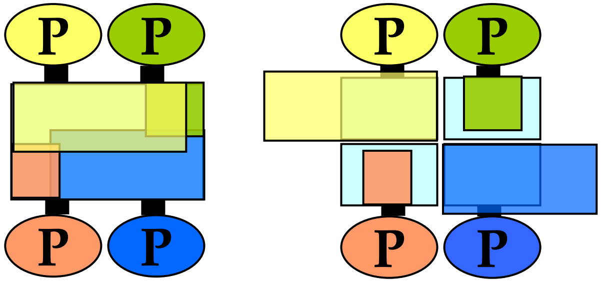
CMP Caching Overview

- **Critical for CMPs**
  - Processor/memory gap
  - Limited pin-bandwidth



4-core CMP

- **Current designs**
  - Shared cache: sharing can lead to contention
  - Private caches: isolation can waste resources



Capacity contention

Wasted capacity

Challenges and Our Approach

- **Key challenges**
  - Growing on-chip wire delay
  - Expensive off-chip accesses
  - Destructive inter-thread interference
  - Diverse workload characteristics

Cooperative Caching Partitioning

- Adapting to a wide range of workloads

CMP Cooperative Caching [Chang/Sohi ISCA06]

- Locality (private caches)
- Capacity (LRU-based sharing)

\+

Time-sharing Based Cache Partitioning

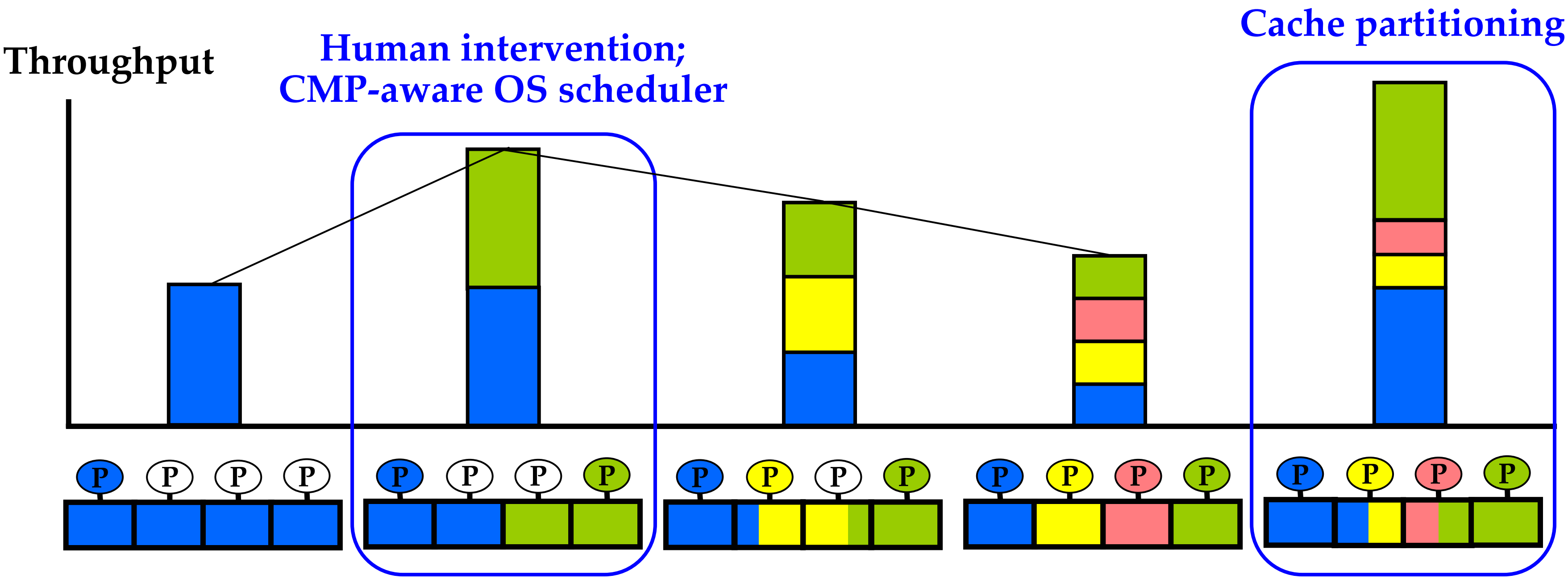
- Throughput
- Fairness
- QoS guarantee

Outline

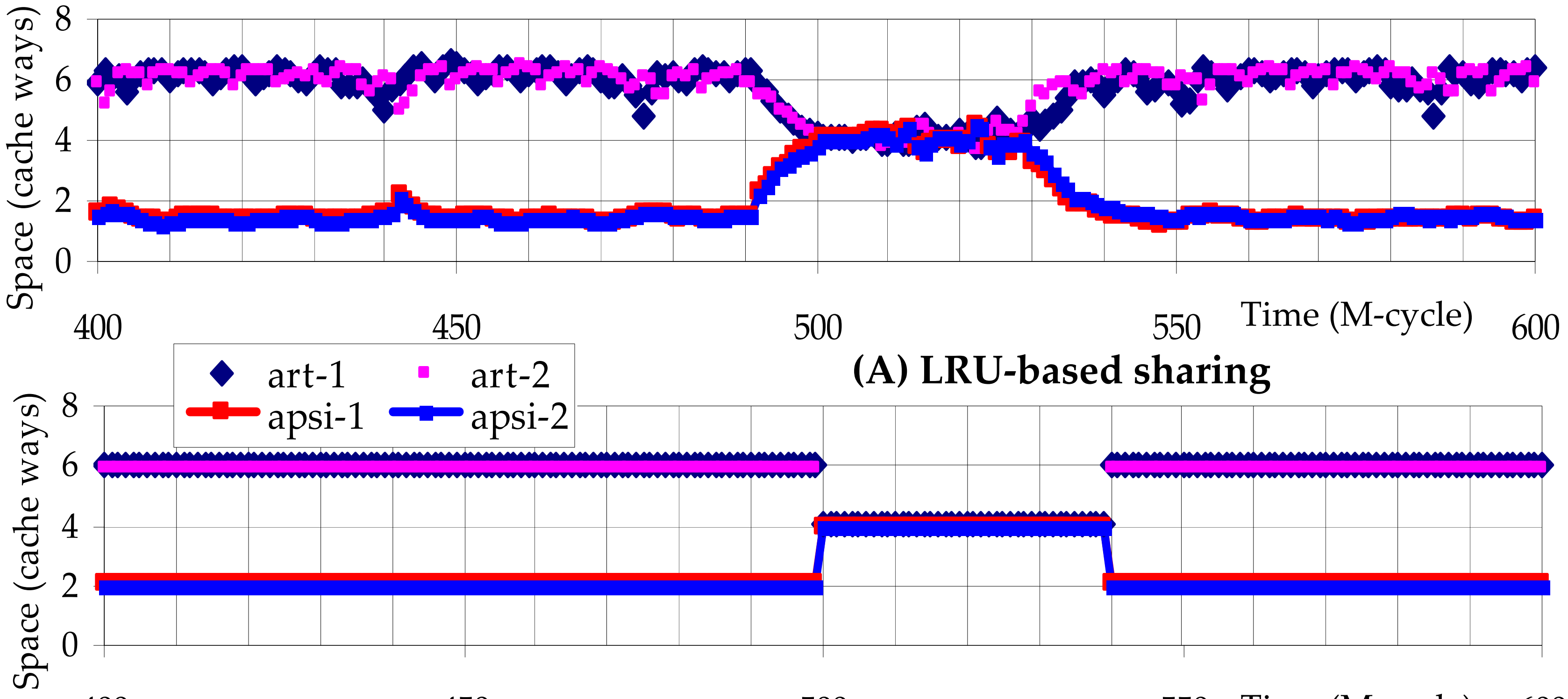
- Overview
- **Problems of destructive interference**
  - Motivating examples
  - Objectives and metrics
  - Limitations of prior proposals
- **Cooperative caching partitioning**
- **Evaluation results**

Thrashing

- **Different ways to run 4 copies of `art`**
  - on a 4-core CMP with 4MB total L2 cache



CMP Cache Partitioning



(A) LRU-based sharing

(B) Cache Partitioning

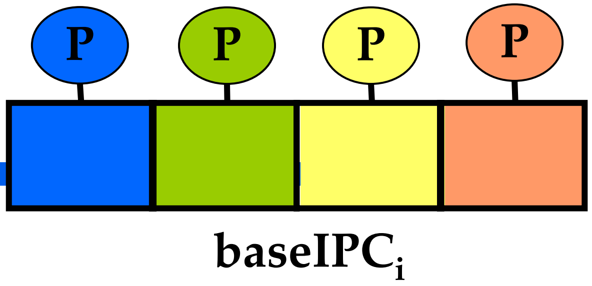
- **Two limitations of prior partitioning schemes**
  1. Coarse-grained partitions: often worse than LRU
  2. Single spatial partition (SSP): hard to resolve conflicts

Optimization Goals

- **Important resource sharing objectives**
  - Maximize overall throughput
  - Improve fairness
  - Guarantee QoS
  - Support priority

- **… can sometimes be conflicting**
  - "Some" threads have to suffer to mitigate thrashing
  - QoS guarantee can restrict throughput optimization
  - Priority support further complicates the problem

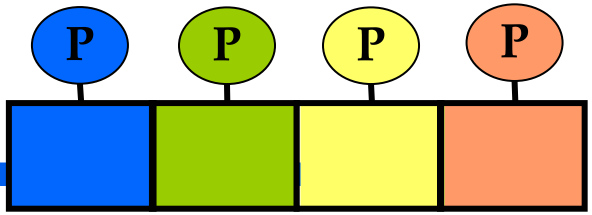
Performance Baseline



- **Proportional partitioning**
  - Resource allocation proportional to priorities/weights
  - Special case: equal priority among concurrent threads
    [Kim et al. PACT ’04] [Yeh/Reinman CASES ’05] [Hsu et al. PACT ’06]

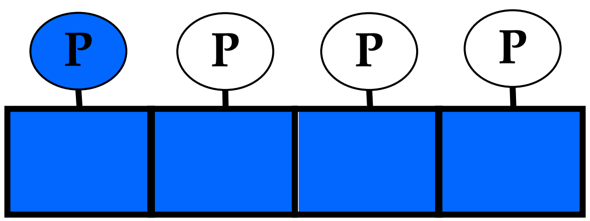
- **Equal-share partition as our default baseline**
  - Correspond to private cache based CMPs and SMPs
  - Achieve the “baseline” performance without effort
  - Our proposal can support proportional partitioning
    - Different speedup curves, same partitioning policy/algorithm

Metrics Definition



$baseIPC_i$

- **Our metrics**
  - QoS := $\sum(slowdown_i)$ = $\sum \min(0, IPC_i/baseIPC_i - 1)$
    - QoS guaranteed if this value ≥ threshold (e.g., -5%)
    - [Yeh/Reinman CASES '05]
  - Fair speedup (FS) := Hmean ($IPC_i/baseIPC_i$)
    - Reduce execution time; penalize unequal speedups
    - Hmean used in [Luo et al. IPDPS '01] (SMT baseline)
    - Hmean of IPCs used in [Dybdahl/Stenstrom HPCA '07]



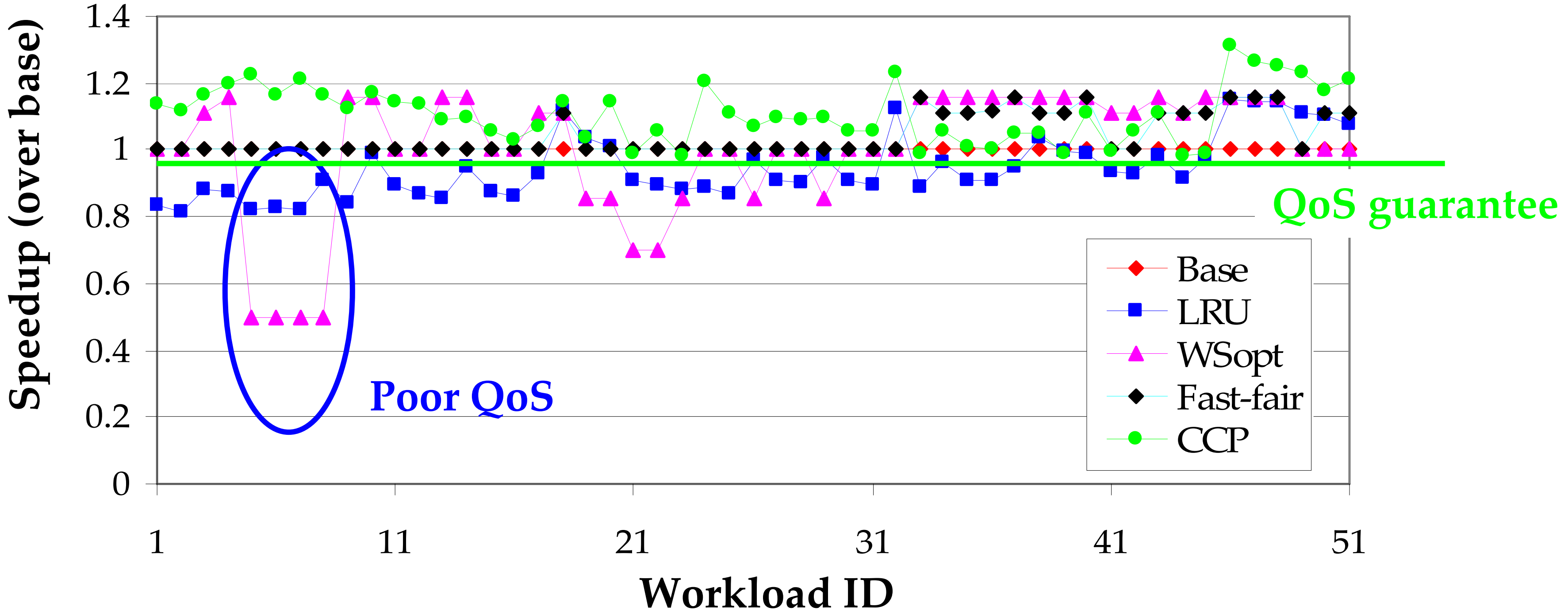
$scIPC_i$

- **Other metrics**
  - Weighted speedup (WS) := sum ($IPC_i/scIPC_i$)
  - Throughput := sum (IPC_i)

Prior Cache Partitioning Schemes

- **Use one partition repeatedly in a stable phase**
  - Hard to satisfy conflicting optimization goals

Speedups of vpr when co-scheduled with large applications

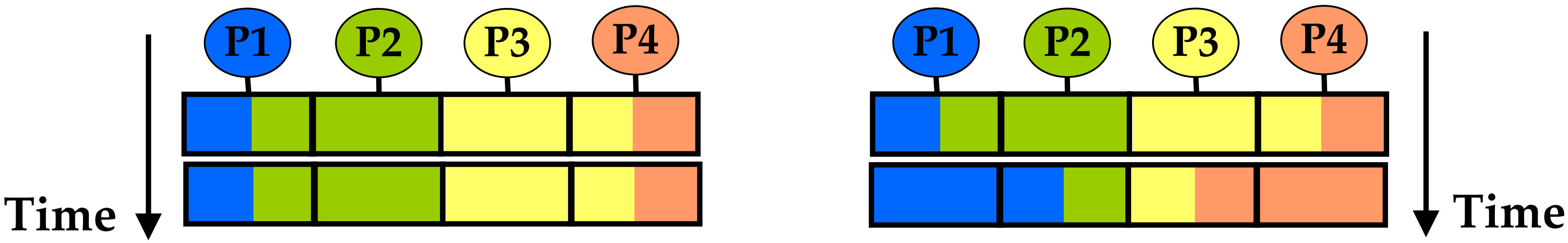


Outline

- Overview
- Problems of destructive interference
- **Cooperative caching partitioning**
  - Time-sharing based cache partitioning
  - Integration with CMP cooperative caching
- **Evaluation results**

Time-share Based Partitioning

- **Throughput-fairness dilemma**
  - Cooperation: Taking turns to speed up
  - Multiple time-sharing partitions (MTP)
- **QoS guarantee**
  - Cooperatively shrink/expand across MTPs
  - Bound average slowdown over the long term



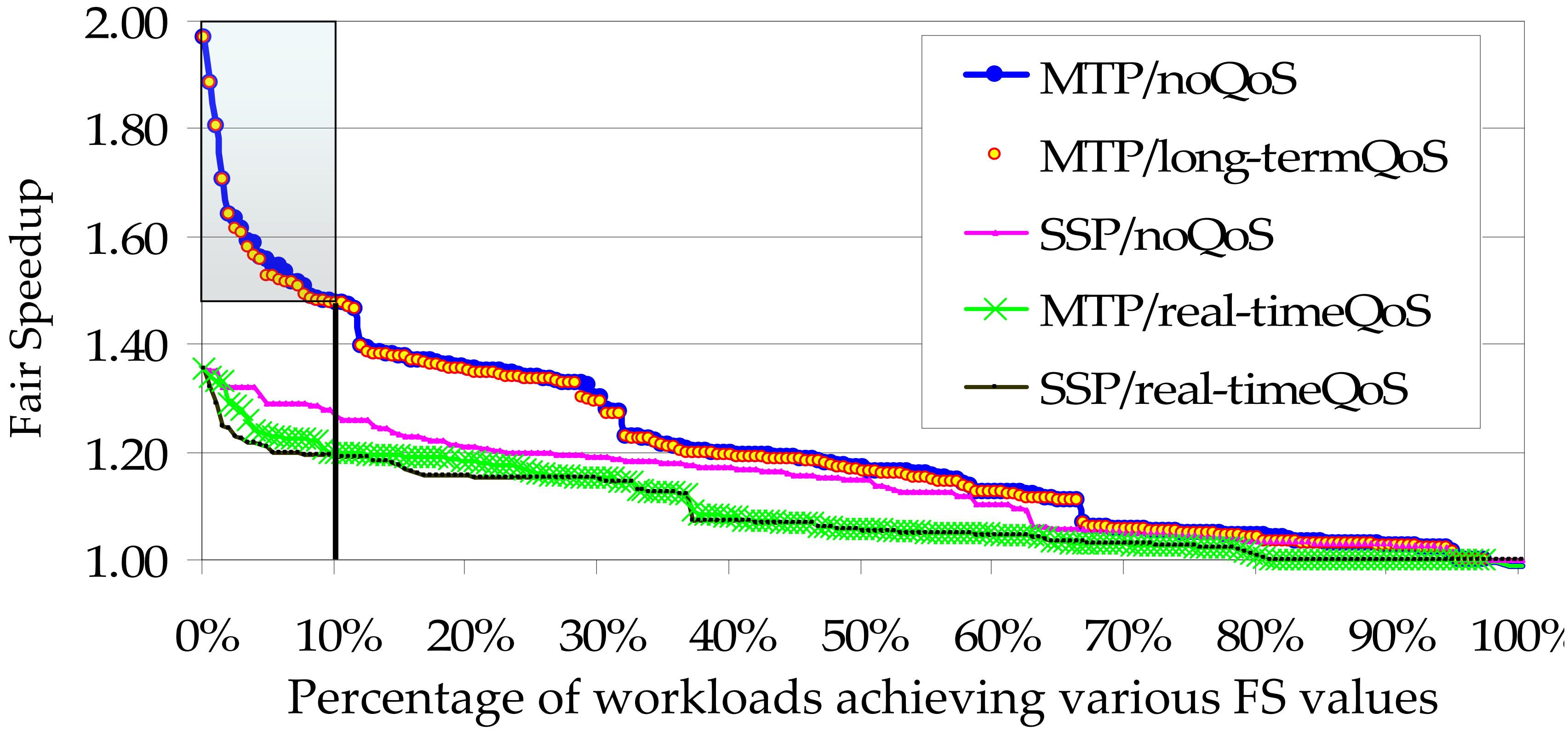
IPC = 0.52 WS=2.42 == IPC = 0.52 WS = 2.42

QoS = -0.52 FS = 1.22 << QoS = 0 FS = 1.97

Fairness improvement and QoS guarantee reflected by higher FS and bounded QoS values

MTP Benefits

- **Better than single spatial partition (SSP)**
  - MTP/long-termQoS almost the same as MTP/noQoS



Offline analysis based on profile info, 210 workloads (job mixes)

Better than MTP

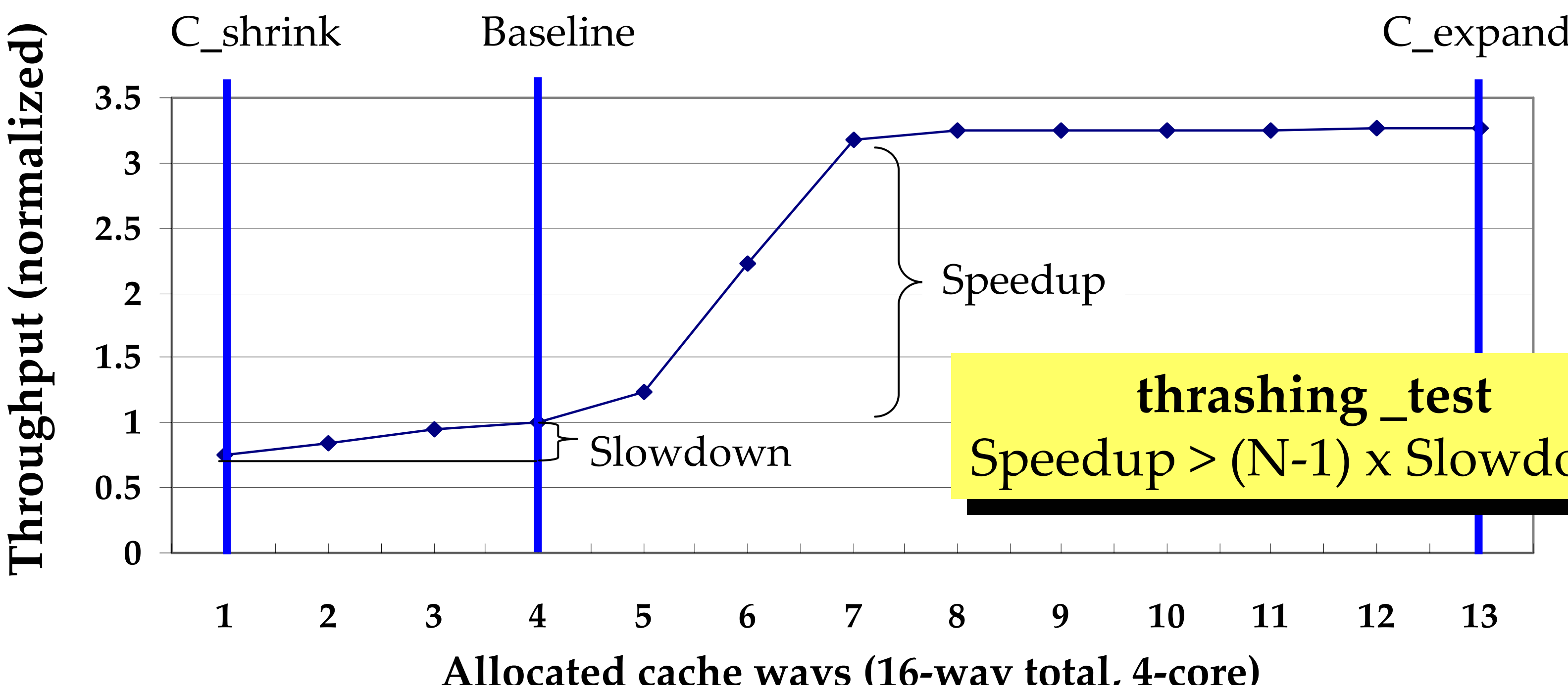
- **MTP issues**
  - Not needed if LRU performs better (LRU often near-optimal [Stone et al. IEEE TOC '92])
  - Partitioning is more complex than SSP
- **Cooperative Cache Partitioning (CCP)**
  - Integration with Cooperative Caching (CC)
  - Exploit CC's latency and LRU-based sharing benefits
  - Simplify the partitioning algorithm
  - Total execution time = Epochs(CC) + Epochs(MTP)
    - Weighted by # of threads benefiting from CC vs. MTP

CC Background

- **CC = private caches + capacity sharing**
- **Sharing mechanism – spill**
  - Placing locally evicted blocks in other on-chip caches
  - Randomly selected host caches, no ripple spilling
- **Sharing policy – aging-based global LRU**
  - Spill brings global data to local caches
  - Global LRU ≅ Local LRU + global spill/reuse
    - Age := 0 when being used (⟶MRU)
    - Age ++ when being spilled (MRU⟶LRU)
    - Age ≥ N triggers global eviction (N=1 is sufficient)
- **Benefits: better latency + LRU-sharing**

Partitioning Heuristic

- **When is MTP better than CC**
  - QoS: ∑speedup > ∑slowdown (over N partitions)
  - Speedup should be large
    - ❑ CC already good at fine-grained tuning



thrashing _test
Speedup > (N-1) x Slowdown

Partitioning Algorithm

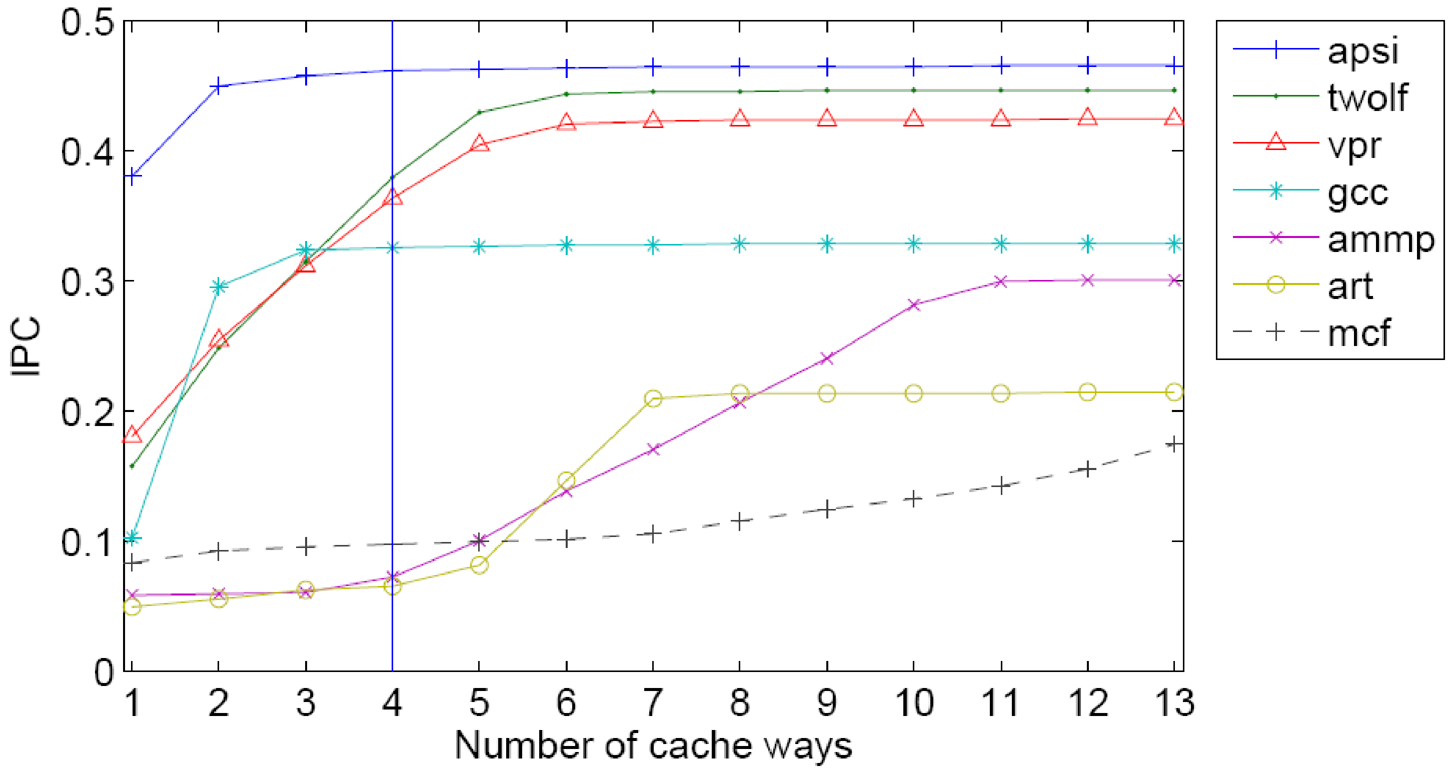
1. **S = All threads - supplier threads (e.g., gcc, swim)**
   - Allocate them with gPar (guaranteed partition, or min. capacity needed for QoS) [Yeh/Reinman CASES '05]
   - For threads in S, init their C_expand and C_shrink
2. **Do thrashing_test iteratively for each thread in S**
   - If thread *t* fails, allocate *t* with gPar, remove *t* from S
   - Update C_expand and C_shrink for other threads in S
3. **Repeat until <u>S is empty</u> or <u>all threads in S pass the test</u>**

Outline

- Overview
- Problems of destructive interference
- Cooperative caching partitioning
- **Evaluation results**

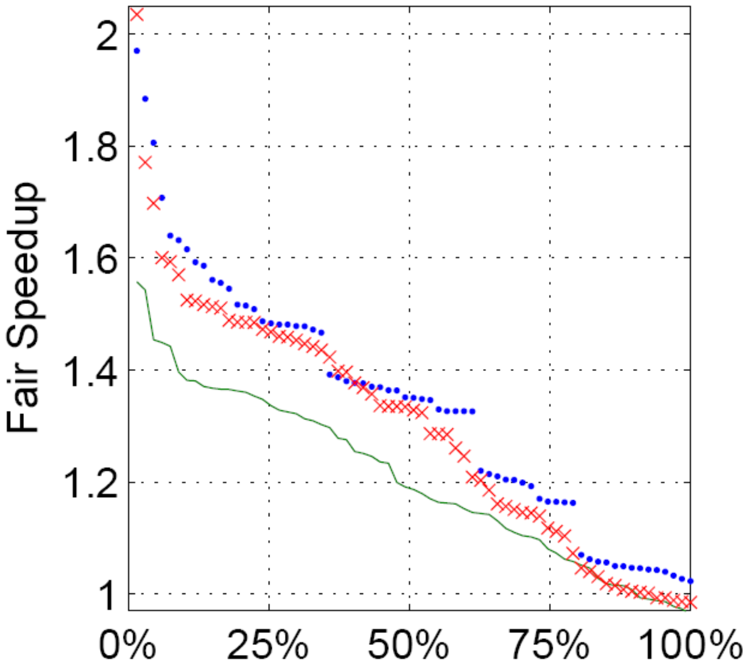
Evaluation

- **Workloads**
  - 7 benchmarks (diverse IPCs and speedup curves)
  - All 4-thread combinations (210 combinations)
  - In-order cores, simulation for fine-grained schemes

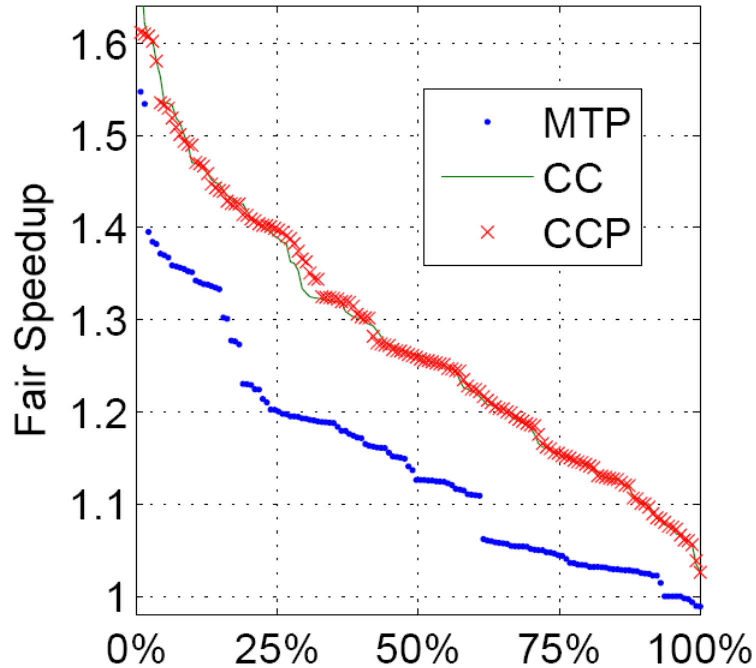


Fair Speedup Results

- **Two groups of workloads**
  - PAR: MTP better than CC (partitioning helps)
  - LRU: CC better than MTP (partitioning hurts)

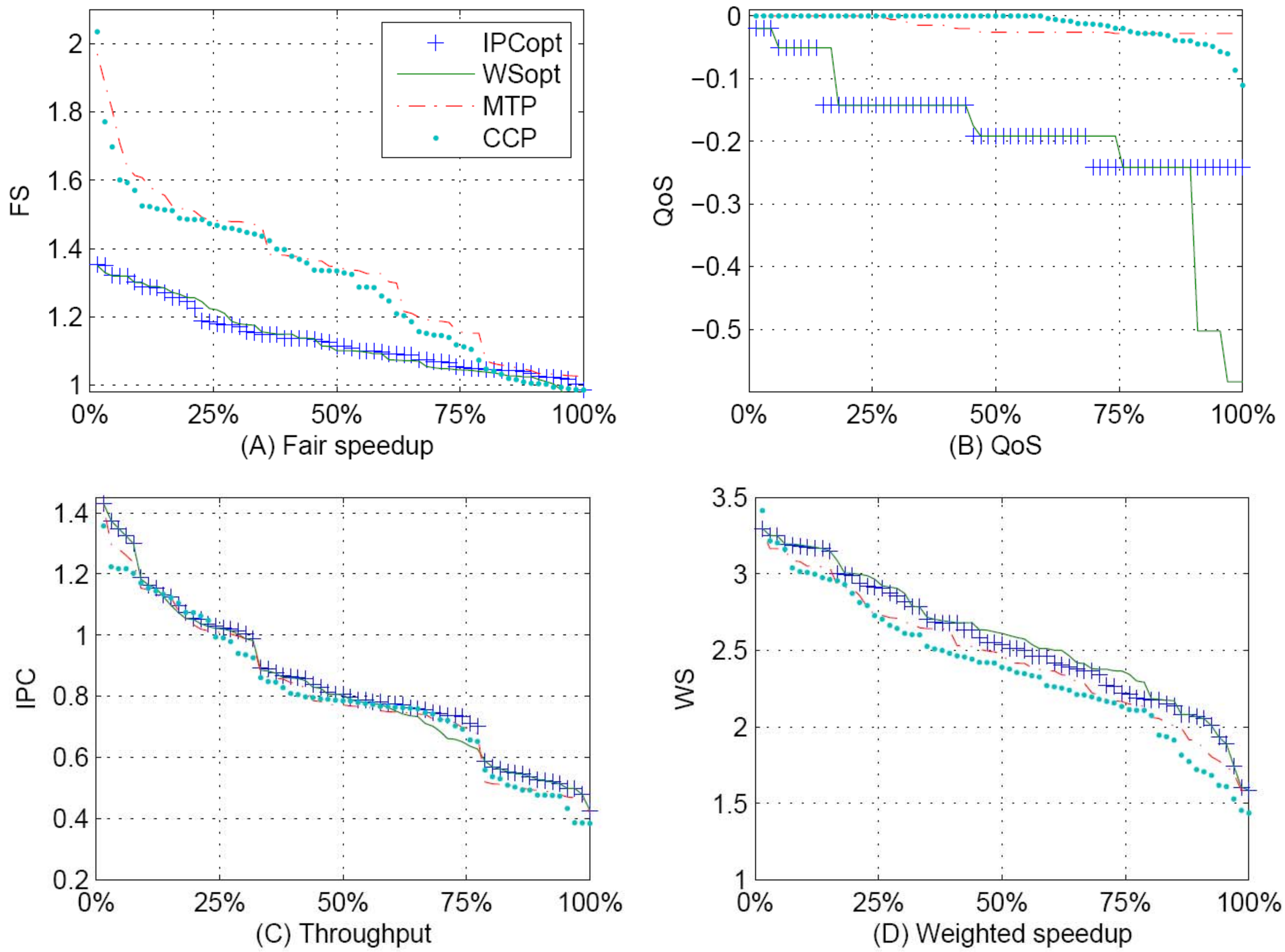


PAR (67 out of 210 workloads)

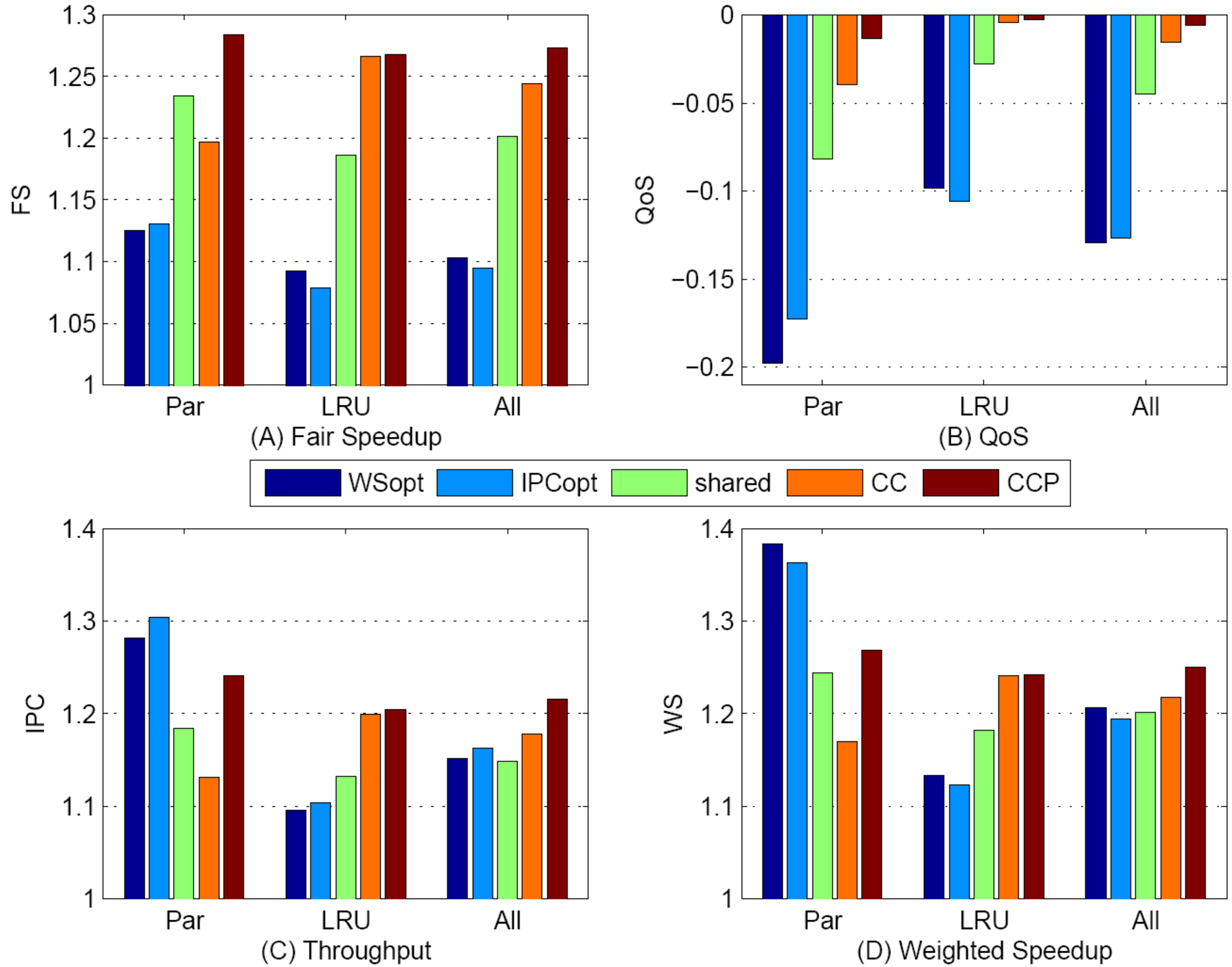


LRU (143 out of 210 workloads)

Results of Other Metrics (for PAR)



Average Improvement



Summary

- **Cooperative Cache Partitioning**
  - Cooperation to resolve conflicts
  - Integration to exploit CC benefits
  - Adaptation to accommodate diversity

Cooperative Caching Partitioning

- Adapting to a wide range of workloads

CMP Cooperative Caching [Chang & Sohi ISCA06]

- Locality (private caches)
- Capacity (LRU-based sharing)

\+

Time-sharing Based Cache Partitioning

- Throughput
- Fairness
- QoS guarantee

Backup Slides

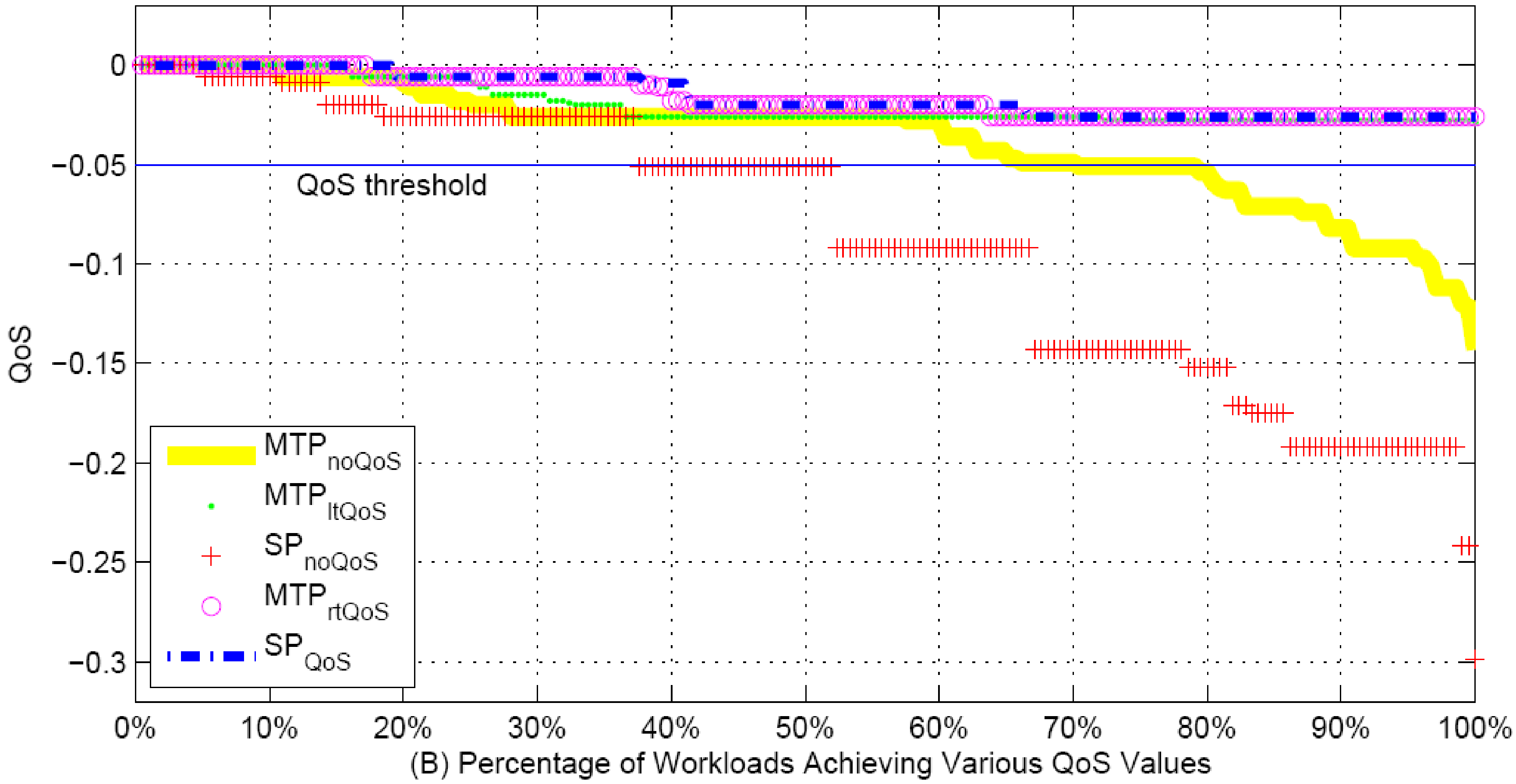
More on Baseline

- **Desirable attributes of a baseline**
  - Provide schedule-independent performance
  - Directly guarantee QoS (no resource overcommitment)
  - Correspond to real implementation (intuitive results)
- **Candidate baselines**
  - LRU sharing among threads
  - Single thread using all caches (SMT)
  - Proportional sharing [Waldspurger thesis 1995]
    - Private caches (equal-share allocation)
- **Policy decoupled from baseline definition**
  - MTP works for proportional sharing partitions
  - We use equal-share allocation for our study

Offline Analysis

- **Idealized setting**
  - Profile available for all (benchmark, capacity) pairs
  - Each workload combination forms a partition space
  - Offline search in the space for optimal results
  - Suitable for cache partitioning (coarse-grained)
- **Used for limit study**
  - Estimate the performance upper limit
  - Discover the limitations of existing schemes
  - Avoid comparison with real implementations

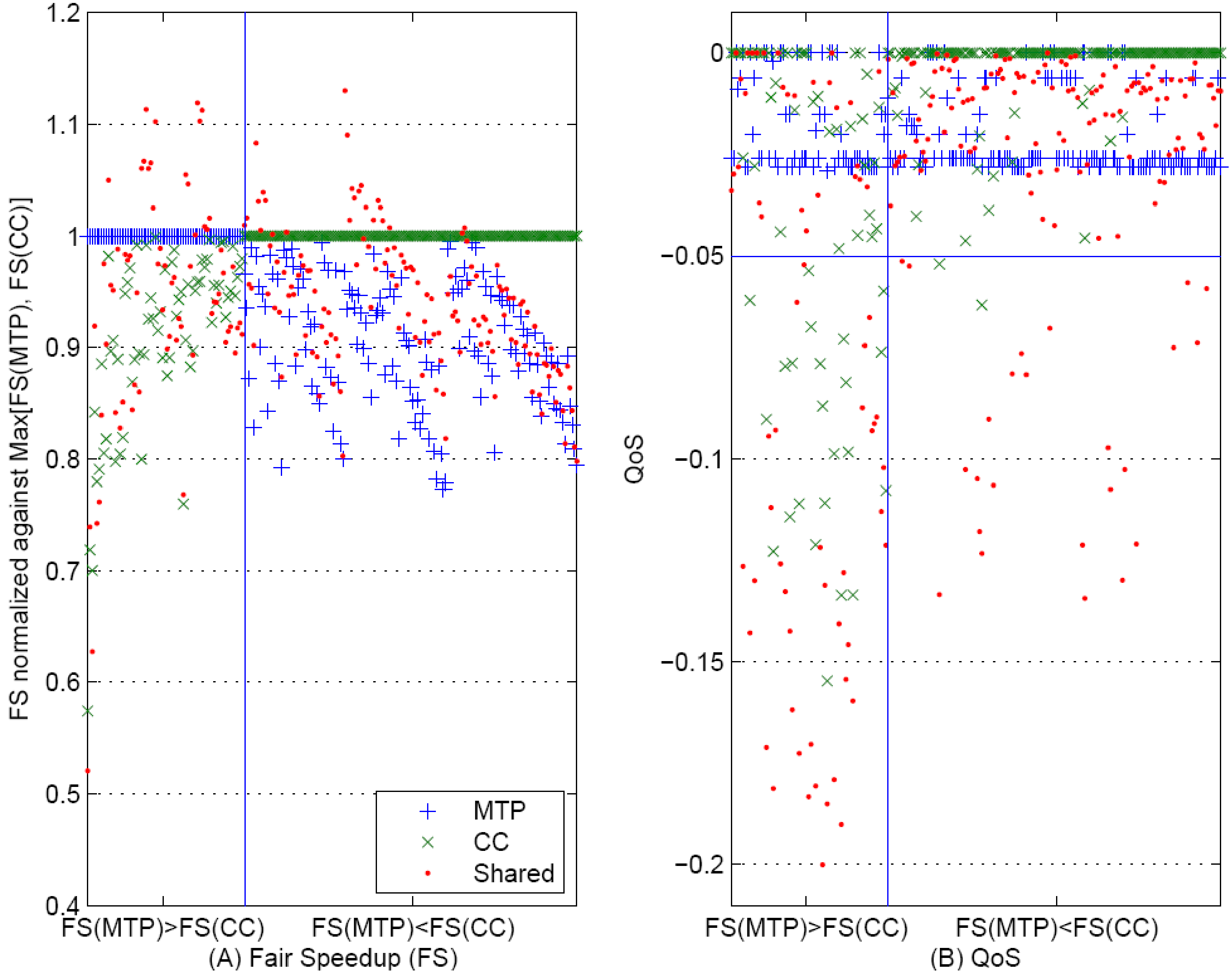
MTP Benefits – QoS Results



Other MTP Issues

- **Support of priority**
  - MTP supports other proportional sharing baseline
  - Also support prioritized time-sharing of MTPs
  - Currently study equal priority (equal-share baseline)
  - Future work need to study software implementation
- **Real-time QoS**
  - Guaranteed partition for threads w/ real-time QoS
  - Apply MTP to other threads
- **Better adaptation to phase/schedule changes**
  - Phase change detection/prediction
  - Cooperate with software to handle schedule changes

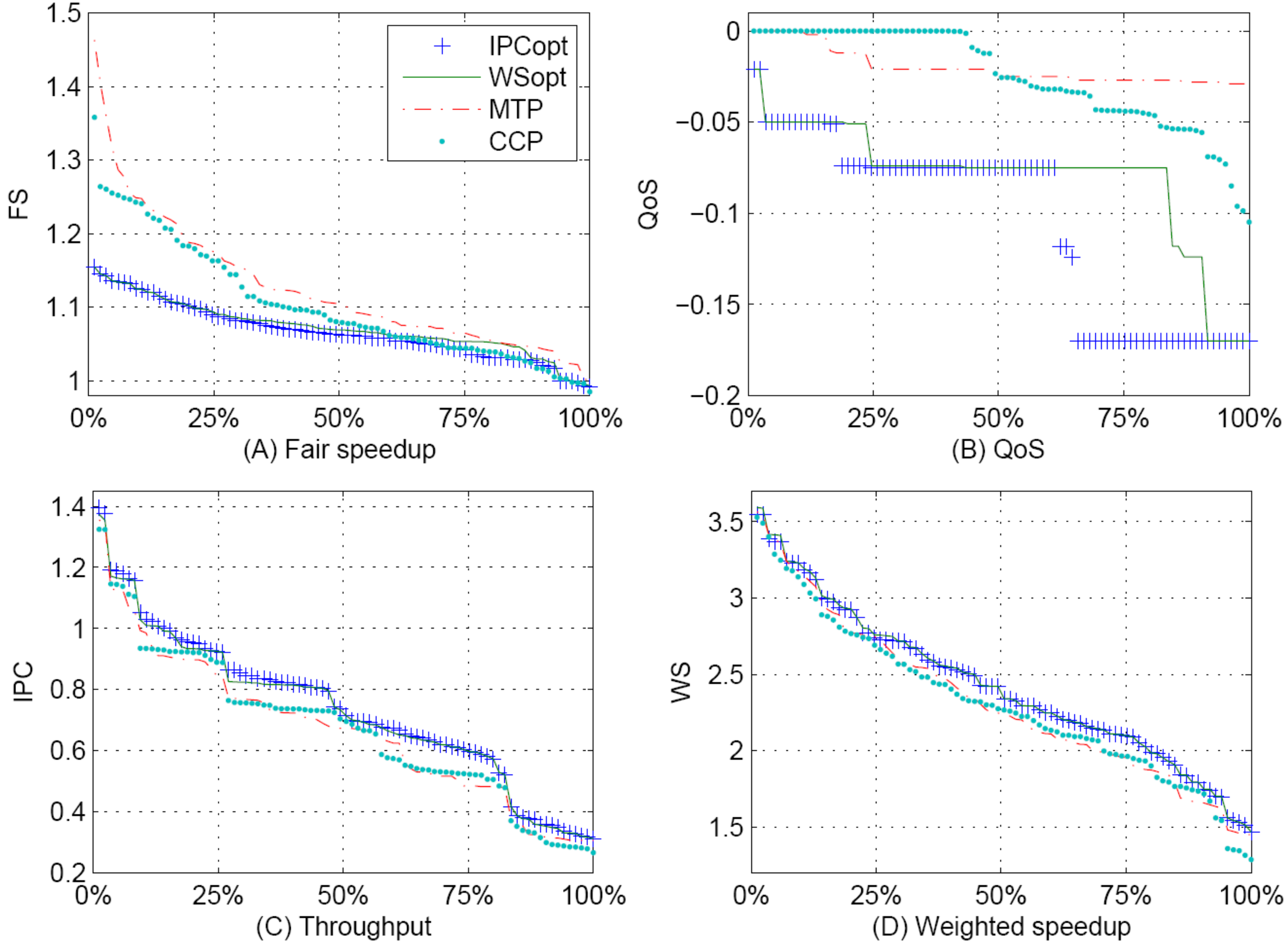
Why MTP + CC? Why not shared?



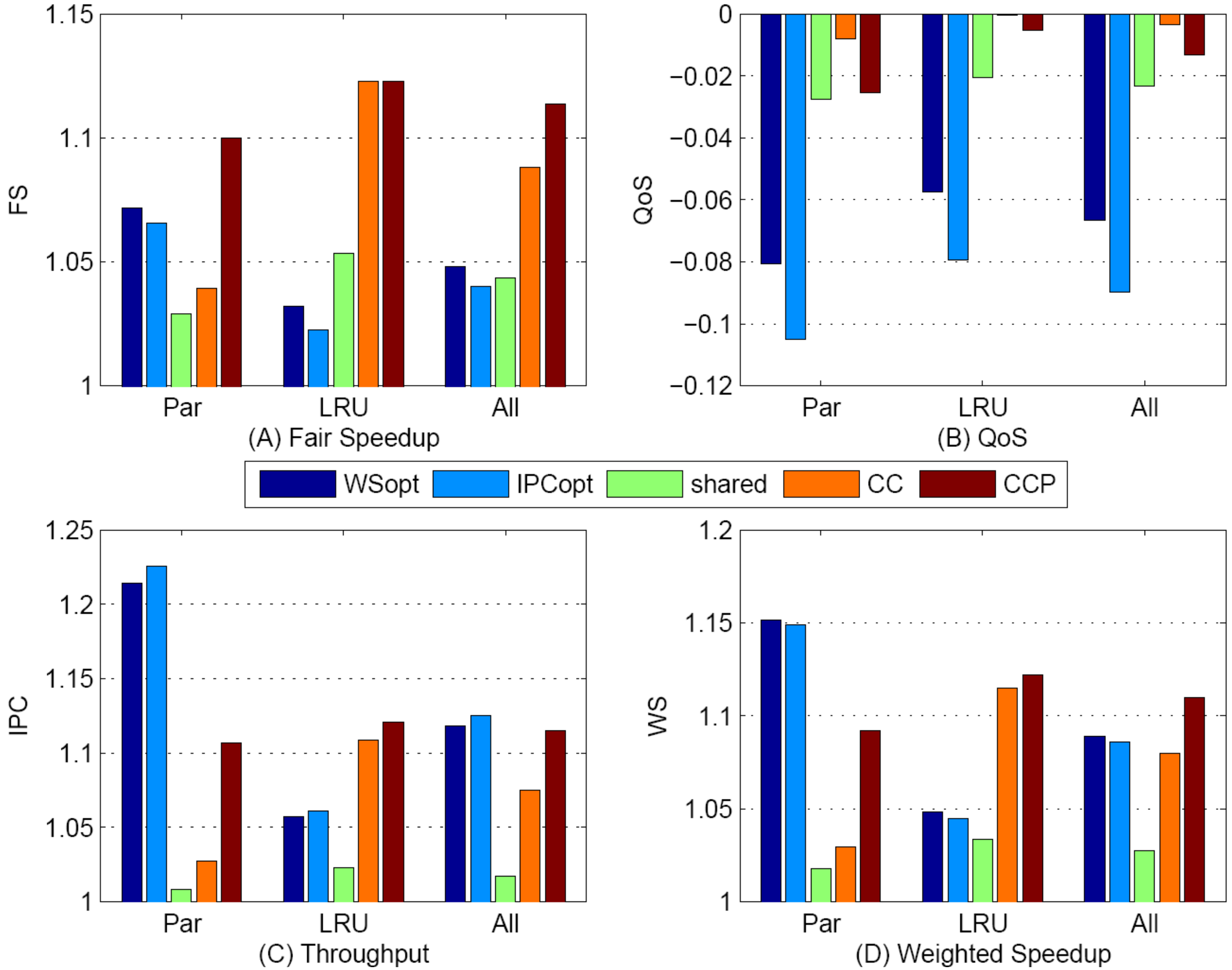
CCP Implementation

- **Epoch size 20M-cycles**
  - Shorter epochs can lead to inaccurate prediction
- **Measurement**
  - Candidate threads get C_expand in sampling epochs
  - Use LRU stack hit counters to estimate the miss rates for smaller capacities
  - Estimate speedups over the given baseline
- **Partitioning**
  - Can be implemented in either software or hardware
- **Enforcement - quota-based throttling**
  - Under-quota threads: spill, but cannot accept spill;
  - Over-quota threads: cannot spill, but accept spill

2MB Total Capacity (for PAR)

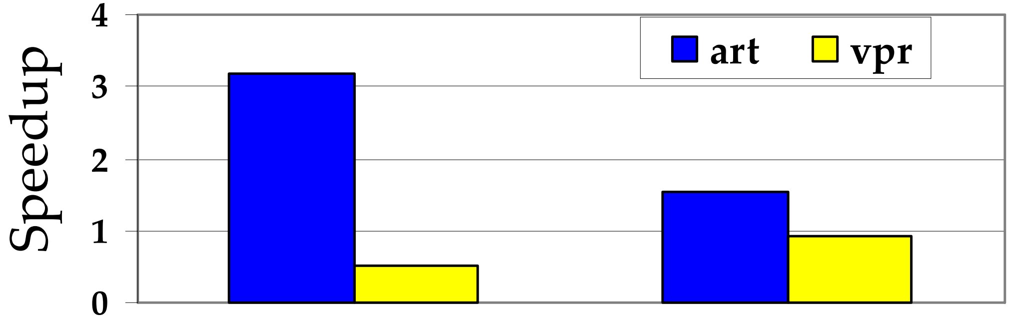


2MB Total Capacity- Summary



Metrics Examples

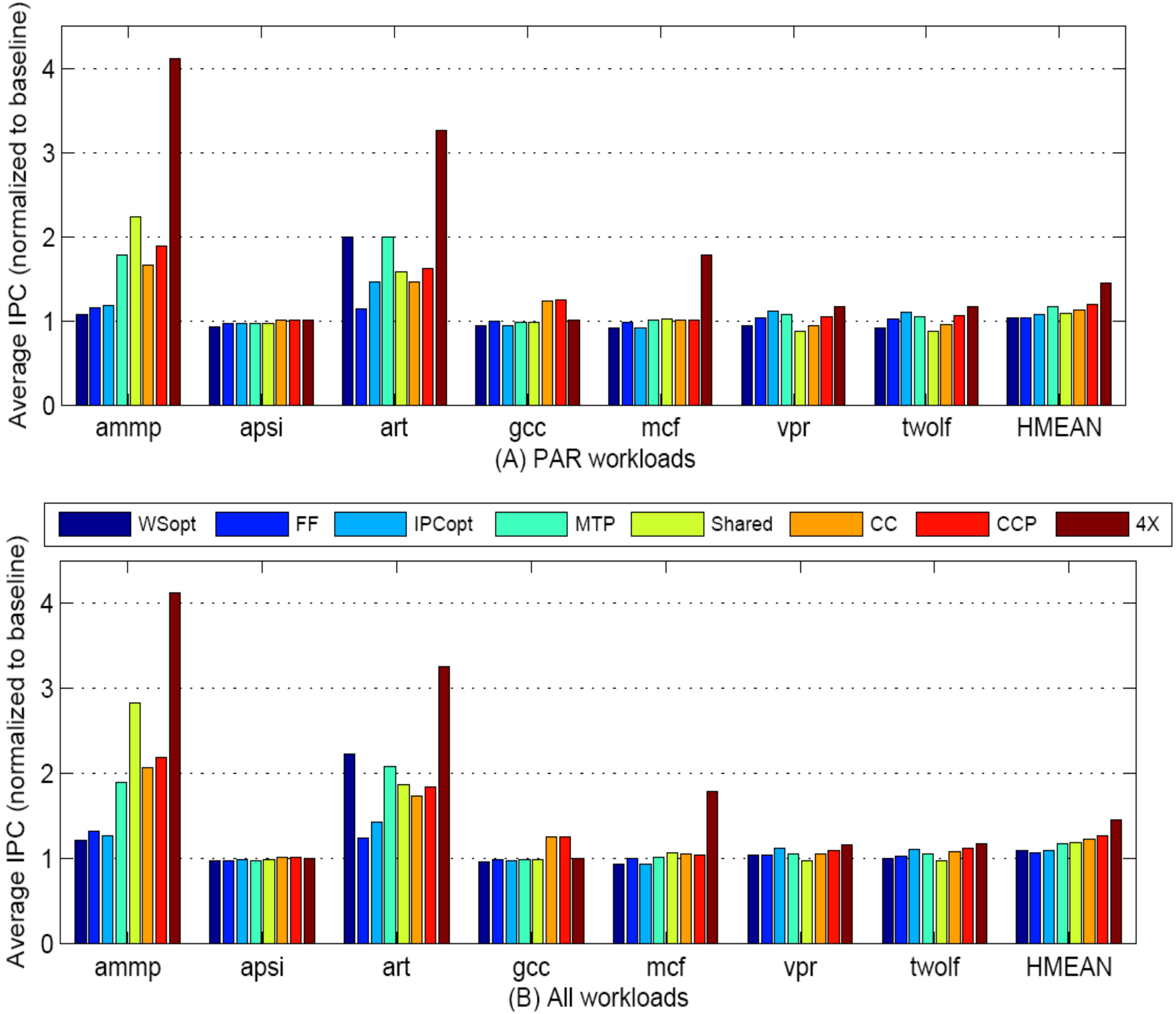
Different trade-offs needed for WS and FS optimizations
Weighted speedup improvement can be unfair



Weighted speedup = 1.40 >> Weighted speedup = 1.27

Fair speedup = 0.86 << Fair speedup = 1.16

Average IPC



Average IPC (normalized to baseline)
0
1
2
3
4
ammp
apsi
art
gcc
mcf
vpr
twolf
HMEAN
(A) PAR workloads
WSopt
FF
IPCopt
MTP
Shared
CC
CCP
4X
Average IPC (normalized to baseline)
0
1
2
3
4
ammp
apsi
art
gcc
mcf
vpr
twolf
HMEAN
(B) All workloads