Analytical Modeling of Partially Shared Caches in Embedded CMPs

Wei Zang and Ann Gordon-Ross[+]

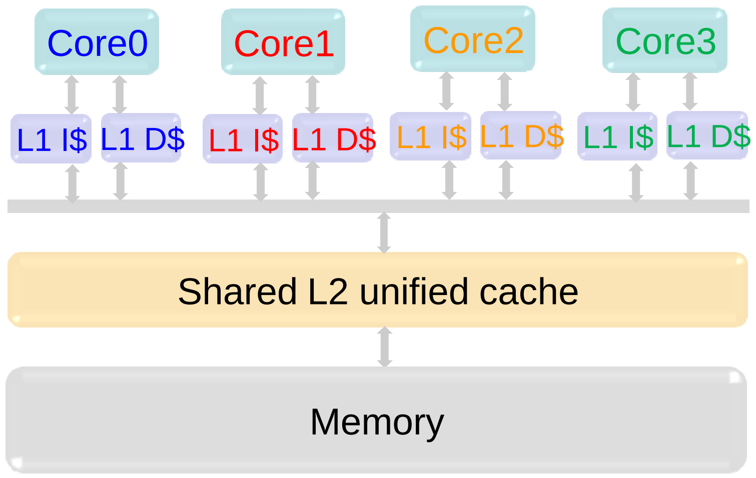
University of Florida
Department of Electrical and Computer Engineering

[+] Also affiliated with NSF Center for High-Performance Reconfigurable Computing

This work was supported by the National Science Foundation (CNS-0953447) and (ECCS-0901706)

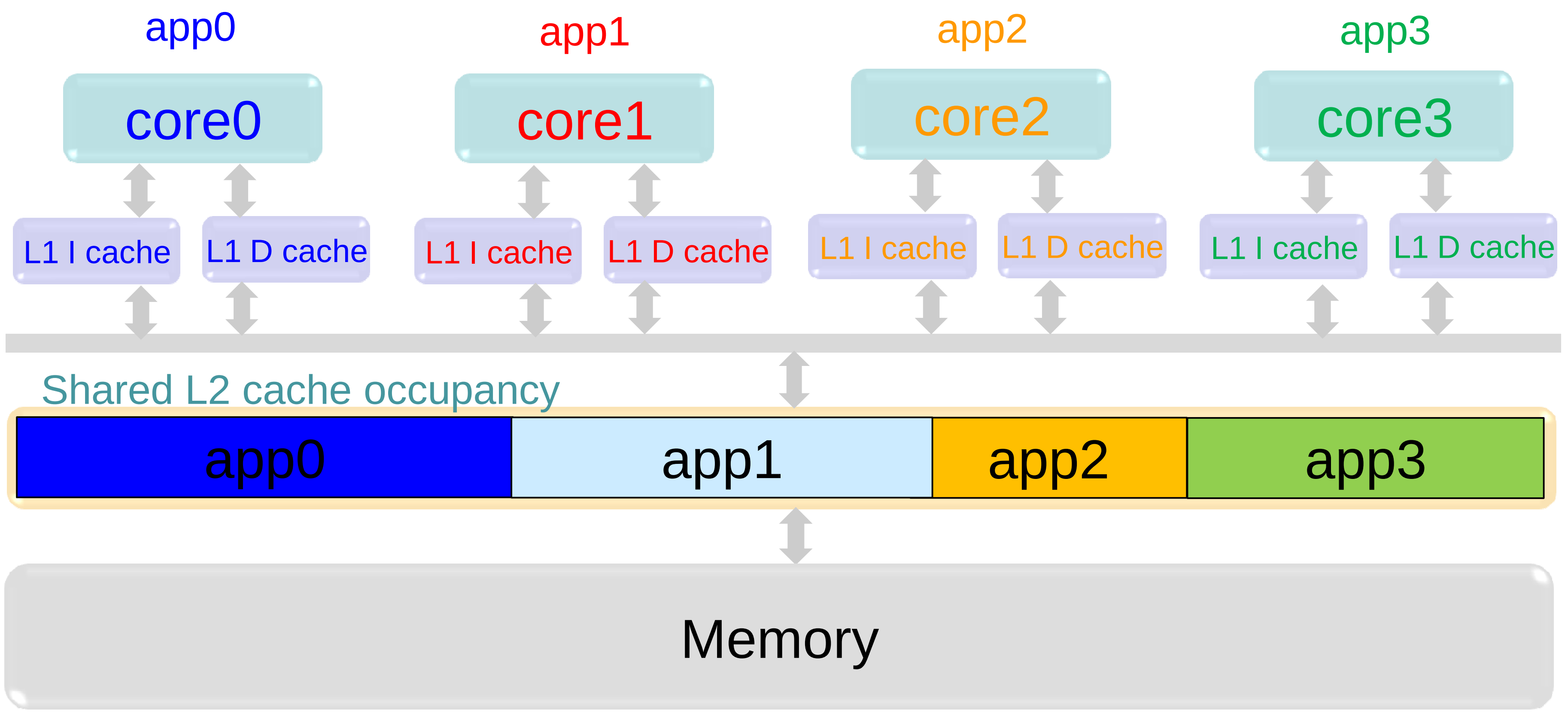
Introduction

- Shared last-level cache (LLC) (e.g., L2/L3) in chip multi-processor systems (CMPs)
  - ARM Cortex-A; Intel Xeon; Sun T2
  - Efficient capacity utilization
    - No need to replicate shared data
    - Occupancy is flexible, dictated by each application's demand



- LLC optimizations
  - Large size to accommodate all sharing cores' data
    - Introduces long access latency, high leakage power, large chip area, etc.
  - Embedded systems optimized for performance, but limited LLC area
  - Configurable cache parameters (e.g., size) similar to private caches
  - High contention results in unfair sharing

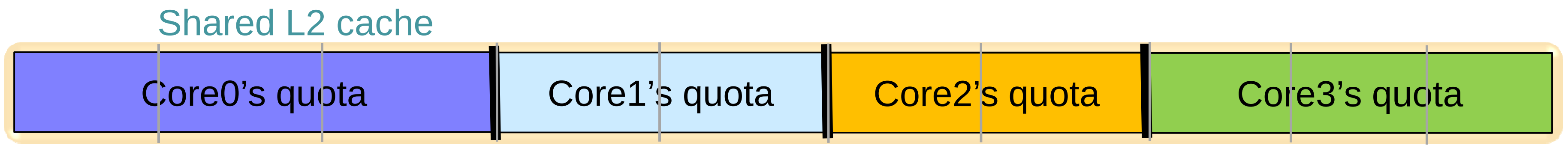
Shared Cache Contention Degrades Performance



app1: Frequent accesses and misses (e.g., streaming applications) can cause high miss rates for other applications!

Cache Partitioning

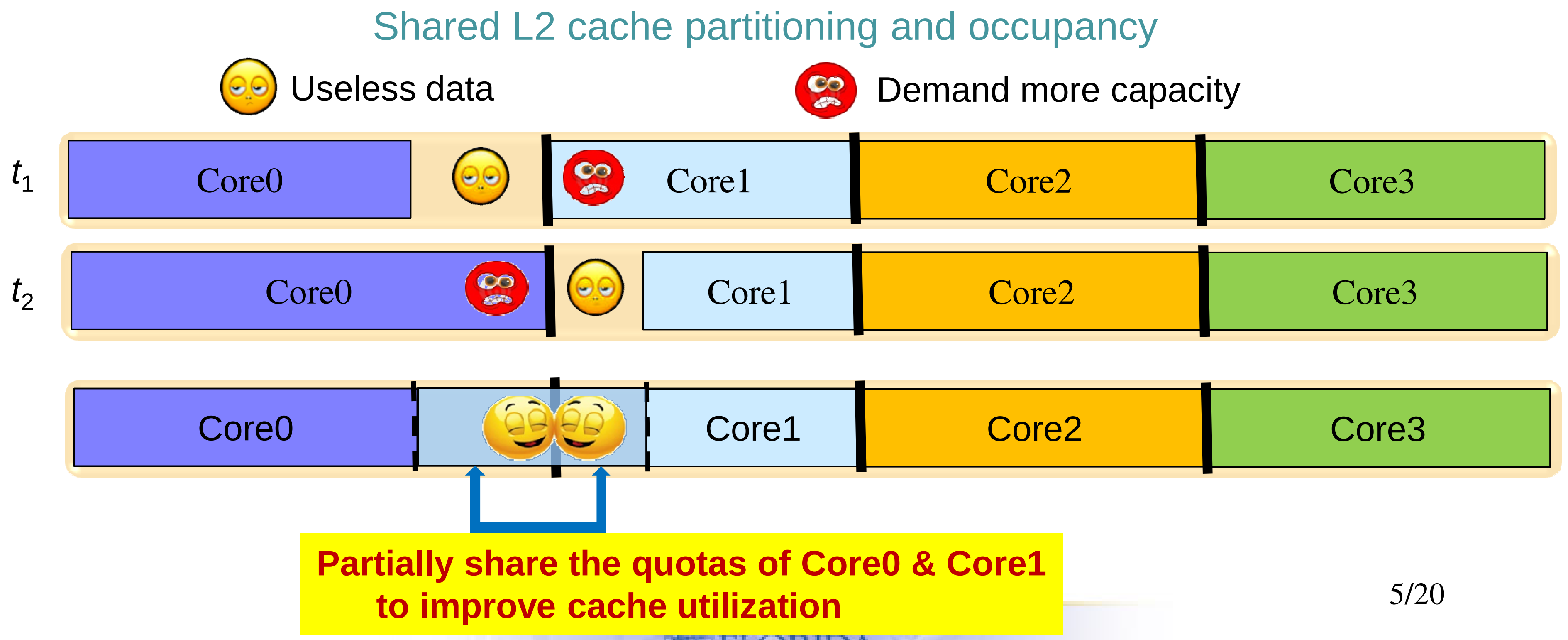
- Uncontrolled cache occupancy degrades performance
- Cache partitioning eliminates shared cache contention
  - Partition cache
  - Allocate quotas (subset of partitions) to the cores
  - Each core's cache occupancy is constrained to that core's quota
  - Partitions/quotas are configurable



- Partition boundaries
  - Set partitioning: OS-based page coloring implementation
  - **Way partitioning**: Hardware-based implementation
    - Typical shared LLC partitioning: *private partitioning*, restrict quotas for exclusive use by the allocated core

Partial Sharing to Improve Cache Utilization

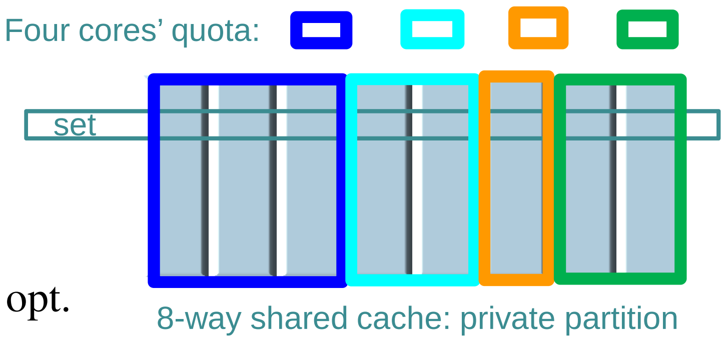
- Private partitioning:
  - Effectively eliminates cache contention
  - Leads to poor cache utilization
    - If a core does not occupy the entire allocated quota temporarily, other cores cannot utilize the vacant quota!



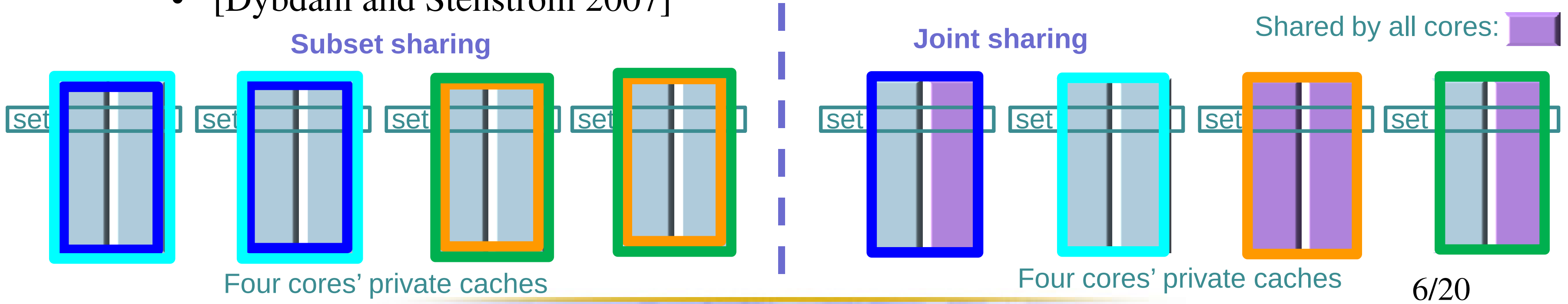
Shared L2 cache partitioning and occupancy
Useless data
Demand more capacity
t_1
Core0
Core1
Core2
Core3
t_2
Core0
Core1
Core2
Core3
Core0
Core1
Core2
Core3
Partially share the quotas of Core0 & Core1 to improve cache utilization

Previous Works on Cache Partitioning

- Shared cache partitioning
  - *Private partitioning* (no sharing)
    - [Qureshi and Patt 2006] Utility-based partitioning, Greedy and refined heuristic method
    - [Suh 2001] Greedy method
    - [Kim 2004] Static & dynamic methods, for fairness opt.

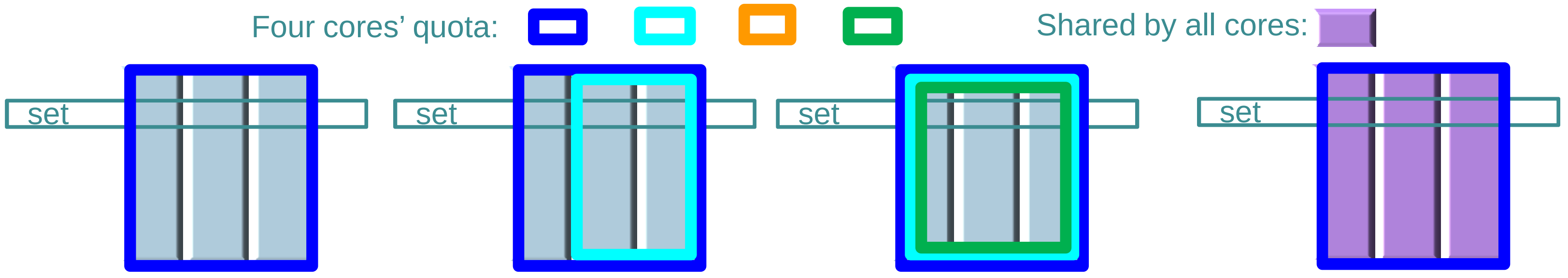


- Private cache partitioning: enables *constrained partial sharing*
  - *Subset sharing*: Cores are subsetted, fully share quotas within the subset
    - [Huh 2007] ; MorphCache
  - *Joint sharing*: a portion/all of a core's quota to be shared by all cores
    - [Dybdahl and Stenstrom 2007]



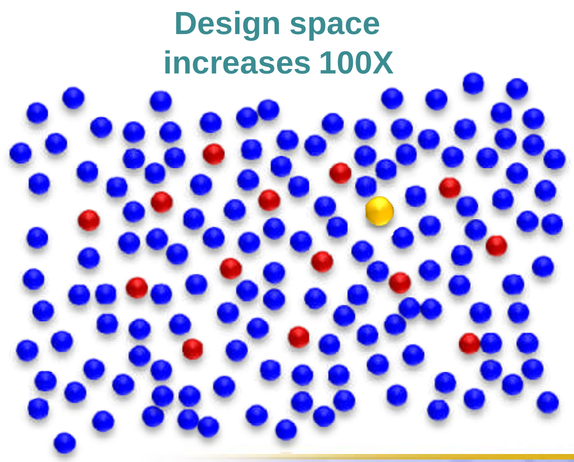
Our Contributions

- Propose CaPPS: ***Ca****che* ***P****artitioning with* ***P****artial* ***S****haring for shared LLC*
  - Partitioning: reduces cache contention
  - Partial sharing: improves cache utilization
  - *Sharing configuration* enables the core's quota to be
    - Privately used by the single core
    - Partially/fully shared with a subset of cores
    - Fully shared with all cores
  - Extensive design space to increase optimization potential



Our Contributions

- Extensive design space in CaPPS
  - Four cores sharing an 8-way cache: 3,347 configurations
  - Prohibitive simulation time
- We developed fast design space exploration
  - Analytical model: probabilistically estimates miss rates of all configurations
    - Evaluates contention based on isolated cache access distributions
    - Evaluates any combination of co-executed applications
  - Applicable to CMPs with an arbitrary number of cores



Simulation
Three months

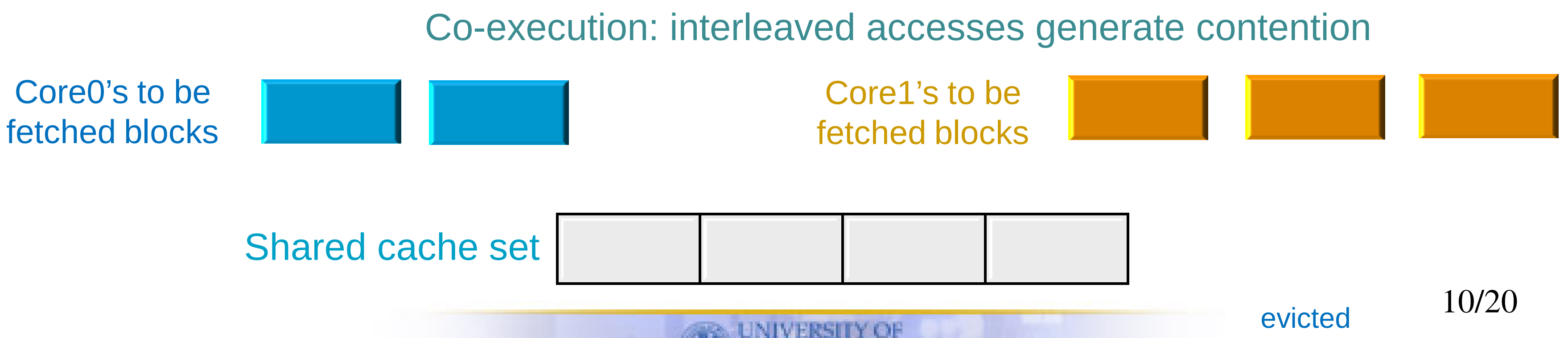
Analytical model
Three hours!

Partitioning and Sharing Configuration in CaPPS

- Way partitioning: physical way boundary
  - Least recently used (LRU) replacement within each core's quota
  - To reduce sharing configurability (design space) and minimize contention
    - Within each core's quota, shared ways begins with the LRU ways
- CaPPS partition design space
  - Private partitioning; Fully shared (no partitioning); Partially shared
    - Constrained partial sharing is a subset of CaPPS: evenly partition the shared cache and each partition a core's private caches
- Partitioning management
  - Restrict a core's occupancy to not exceed the core's quota
    - Core's data can only reside in a particular subset of ways
    - Determine replacement candidate to maintain quota
  - Lightweight overhead: energy, area, and performance
  - Leverage a modified LRU replacement policy and column caching

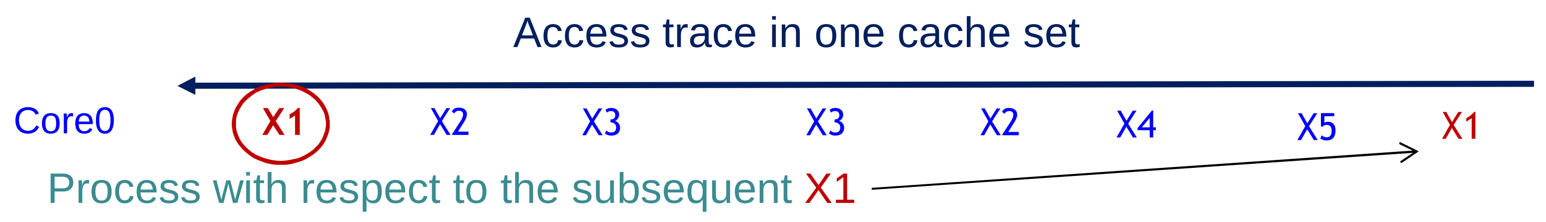
Analytical Modeling for CaPPS

- Cache performance optimization
  - Determine the optimal LLC configuration for each core:
    - Number of ways allocated to each core
    - Number of private and shared ways in the cores' quotas
    - Which cores should share ways
- Analytical model to estimate miss rates
  - Probabilistically analyze contention-induced cache misses
    - Based on isolated (application executing singly) cache access distribution
  - For any combination of co-executed applications & any sharing configuration


Co-execution: interleaved accesses generate contention
Core0's to be fetched blocks
Core1's to be fetched blocks
Shared cache set
evicted

Stack and Reuse Distances

- Isolated cache access characteristics dictate shared contention
  - Determine distances between two consecutive accesses to the same block
  - **Stack distance**
    - Number of conflicts: *unique* blocks that map to the same cache set as the processed address
  - **Reuse distance**
    - Number of accesses that map to the same cache set as the processed address


Access trace in one cache set
Core0
X1
X2
X3
X3
X2
X4
X5
X1
Process with respect to the subsequent X1

Unique conflicts: X5, X4, X2, X3 ➔ stack distance = 4
If no cache sharing, hit if associativity >= 5

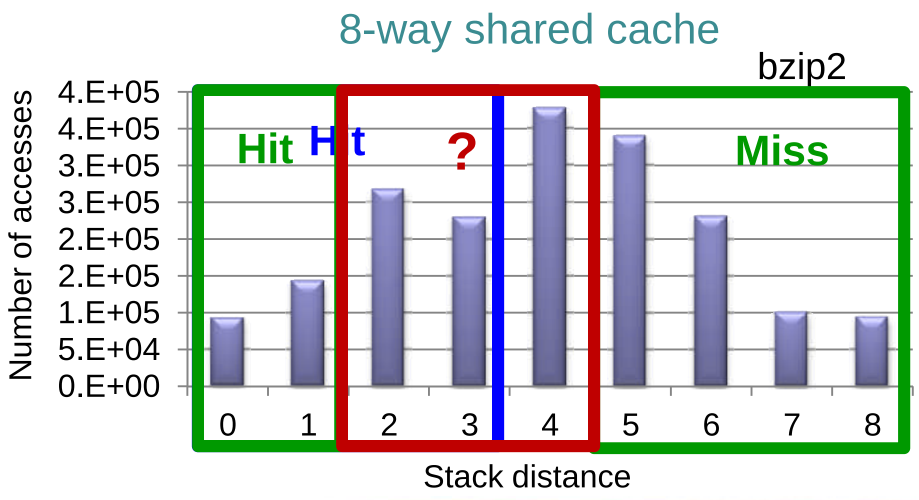
Number of accesses between the two X1: X5, X4, X2, X3, X3, X2, X1 ➔ reuse distance = 7
Dictates contention, used to determine # of interleaved accesses from other cores

Previous Works on Cache Contention Analysis

- Only target fully shared caches
  - [Chandra 2005]: used access traces for isolated threads to predict cache contention
    - Did not consider the interaction between CPI variations and cache contention
  - [Eklov 2011]: simpler model, predicted reuse distance distribution when an application is co-executed with other applications
  - [Chen and Aamodt 2009]: Markov model for multi-threaded applications with inter-thread communication
- CaPPS enables partially sharing of a core’s quota
  - Analytical modeling is more challenging
    - Only other cores’ cache accesses that evict blocks to the partially shared ways affect a core’s miss rate
  - Developed based on the fundamental ideas in previous works [Chandra 2005, Eklov 2011]
    - We similarly assume no shared data among applications

Cache Contention's Effects on Miss Rate Evaluation

- With no sharing
  - Hit/miss of an access is determined by the stack distance
    - Generate isolated access trace
    - Evaluate stack distance for each accessed address using stack-based trace-driven cache simulator
    - Accumulate histogram of stack distances
- With sharing
  - Consider interleaved accesses from other cores

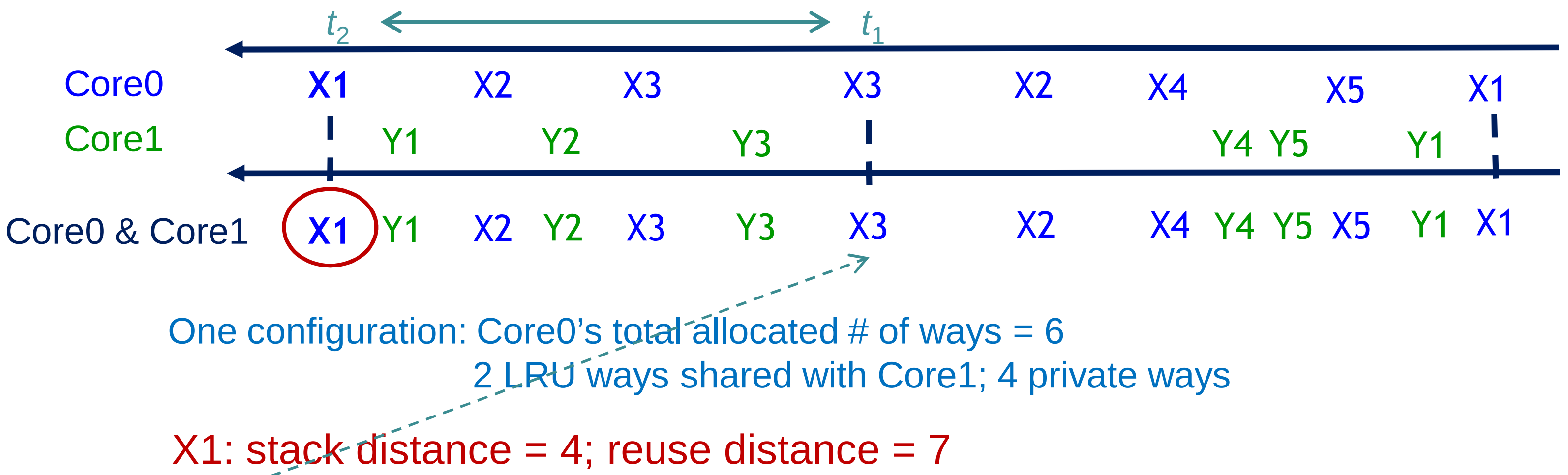


No sharing: e.g., four private ways
Hit when stack distance < 4

Sharing: e.g., two private ways & three shared ways
Hit when stack distance < 2
Miss when stack distance ≥ 5
How about 2 ≤ stack distance < 5 ?

Interleaved Access Traces

Access trace in one cache set



One configuration: Core0's total allocated # of ways = 6
2 LRU ways shared with Core1; 4 private ways

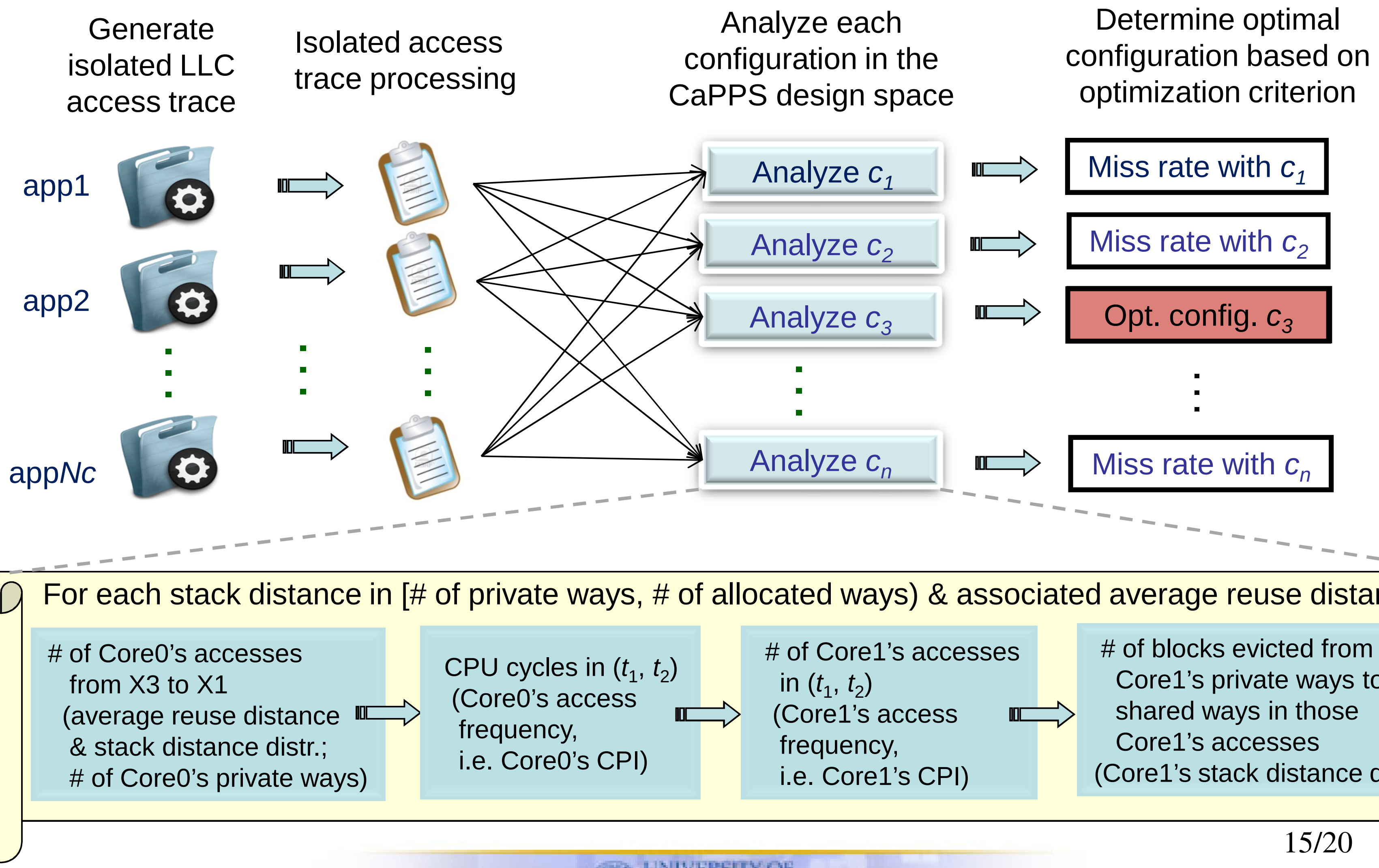
X1: stack distance = 4; reuse distance = 7

X3 evicts X1 from Core0's private ways
Core1's accesses after X1's eviction dictate whether X1 is in shared ways
i.e., accesses in (t_1, t_2): **Y3, Y2, Y1**
If accessing Y1, Y2, Y3 evicts two or more blocks into shared ways, X1 is a miss

For each stack distance in [# of private ways, # of allocated ways) & associated average reuse distance

$$\begin{tikzcd}
\text{\# of Core0's accesses from X3 to X1 (average reuse distance \& stack distance distr.; \# of Core0's private ways)} \arrow[r, Rightarrow] & \text{CPU cycles in } (t_1, t_2) \text{ (Core0's access frequency, i.e. Core0's CPI)} \arrow[r, Rightarrow] & \text{\# of Core1's accesses in } (t_1, t_2) \text{ (Core1's access frequency, i.e. Core1's CPI)} \arrow[r, Rightarrow] & \text{\# of blocks evicted from Core1's private ways to shared ways in those Core1's accesses (Core1's stack distance distr.)}
\end{tikzcd}$$

Analytical Modeling Overview


Generate isolated LLC access trace
Isolated access trace processing
Analyze each configuration in the CaPPS design space
Determine optimal configuration based on optimization criterion
app1
app2
appNc
Analyze c_1
Analyze c_2
Analyze c_3
Analyze c_n
Miss rate with c_1
Miss rate with c_2
Opt. config. c_3
Miss rate with c_n
For each stack distance in [# of private ways, # of allocated ways) & associated average reuse distance
# of Core0's accesses from X3 to X1 (average reuse distance & stack distance distr.; # of Core0's private ways)
CPU cycles in (t_1, t_2) (Core0's access frequency, i.e. Core0's CPI)
# of Core1's accesses in (t_1, t_2) (Core1's access frequency, i.e. Core1's CPI)
# of blocks evicted from Core1's private ways to shared ways in those Core1's accesses (Core1's stack distance distr.)

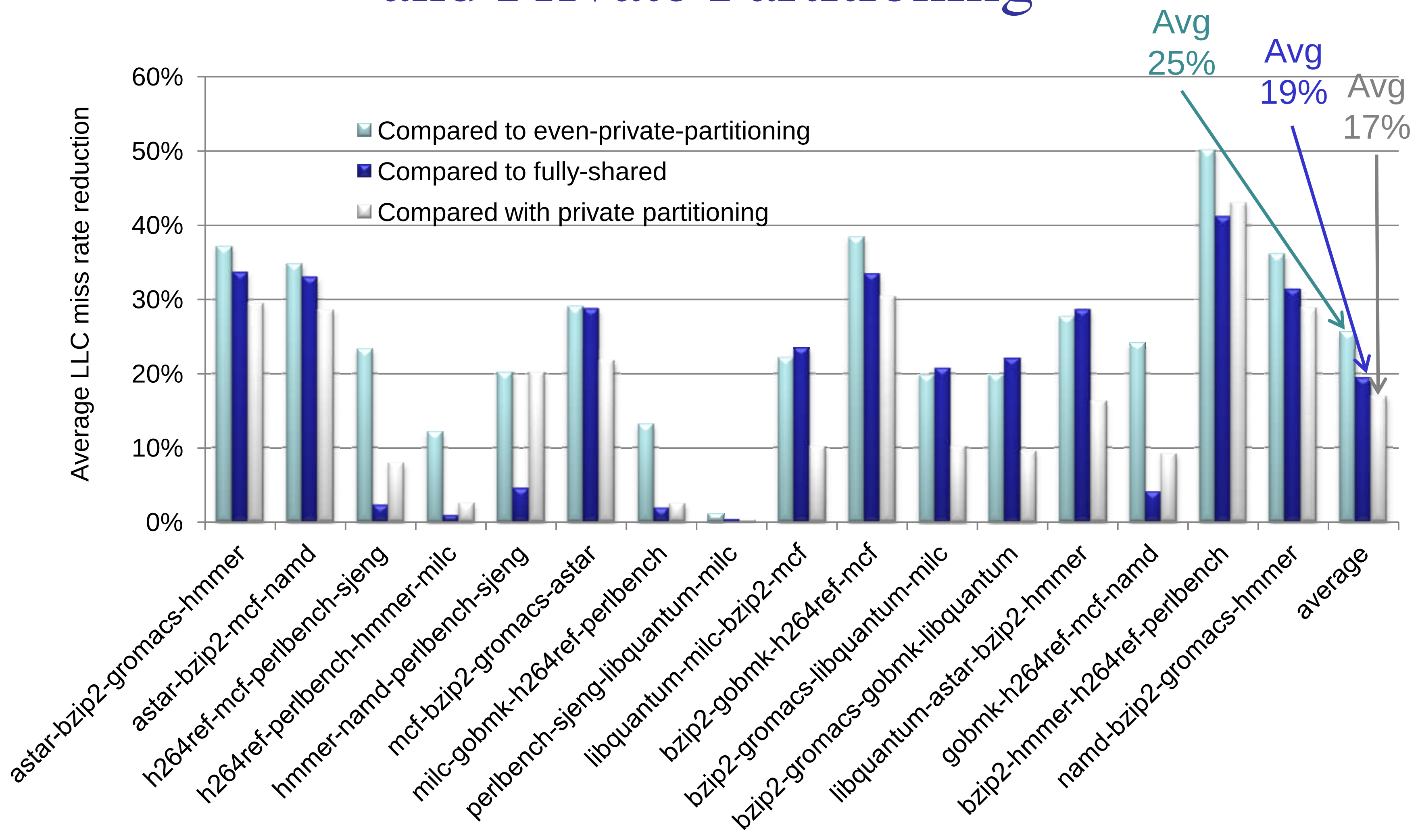
Experiment Setup

- Twelve benchmarks selected from SPEC CPU 2006 suite
  - Performed phase classification to select 500 million consecutive instructions with similar behavior: *simulation interval*
- 4-core CMPs parameters

Components	**Parameters**
CPU	2 GHz clock, 1 thread
L1 instruction cache	Private, total size of 8 KB, block size of 64 B, 2-way associativity, LRU replacement, access latency of 2 CPU cycles
L1 data cache	Private, total size of 8 KB, block size of 64 B, 2-way associativity, LRU replacement, access latency of 2 CPU cycles
L2 unified cache	Shared, total size of 1 MB, block size of 64 B, 8-way associativity, LRU replacement, access latency of 20 CPU cycles, non-inclusive
Memory	3 GB size, access latency of 200 CPU cycles
L1 caches to L2 cache bus	Shared, 64 B width, 1 GHz clock, first come first serve (FCFS) scheduling
Memory bus	64 B width, 1 GHz clock

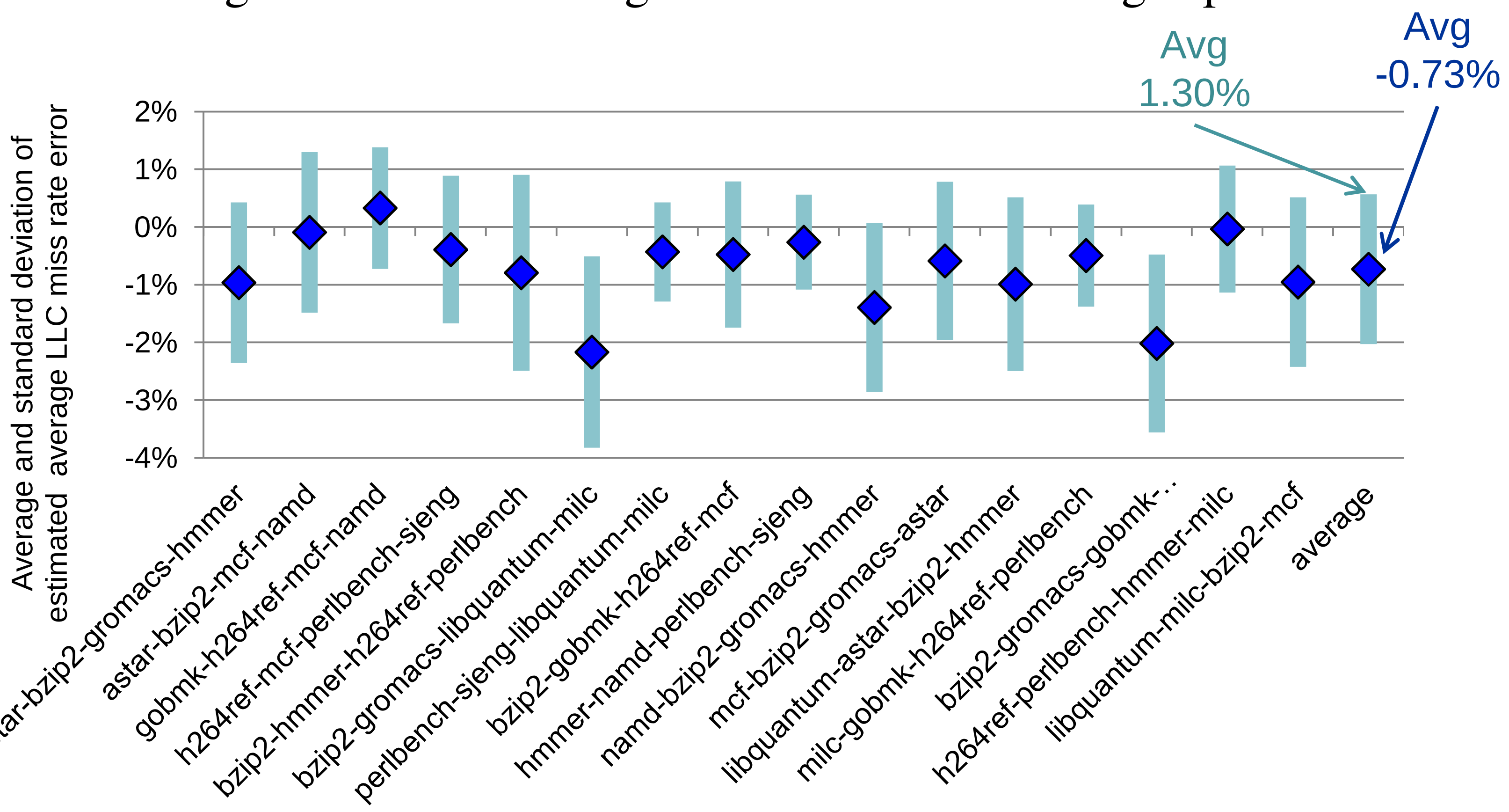
- Modified “gem5” to simulate CaPPS and generate exact results
- Executed each benchmark in isolation to generate isolated access trace
- Arbitrarily selected four benchmarks to be co-executed as one benchmark set
  - Evaluated sixteen benchmark sets

Comparing CaPPS with Baseline Configurations and Private Partitioning

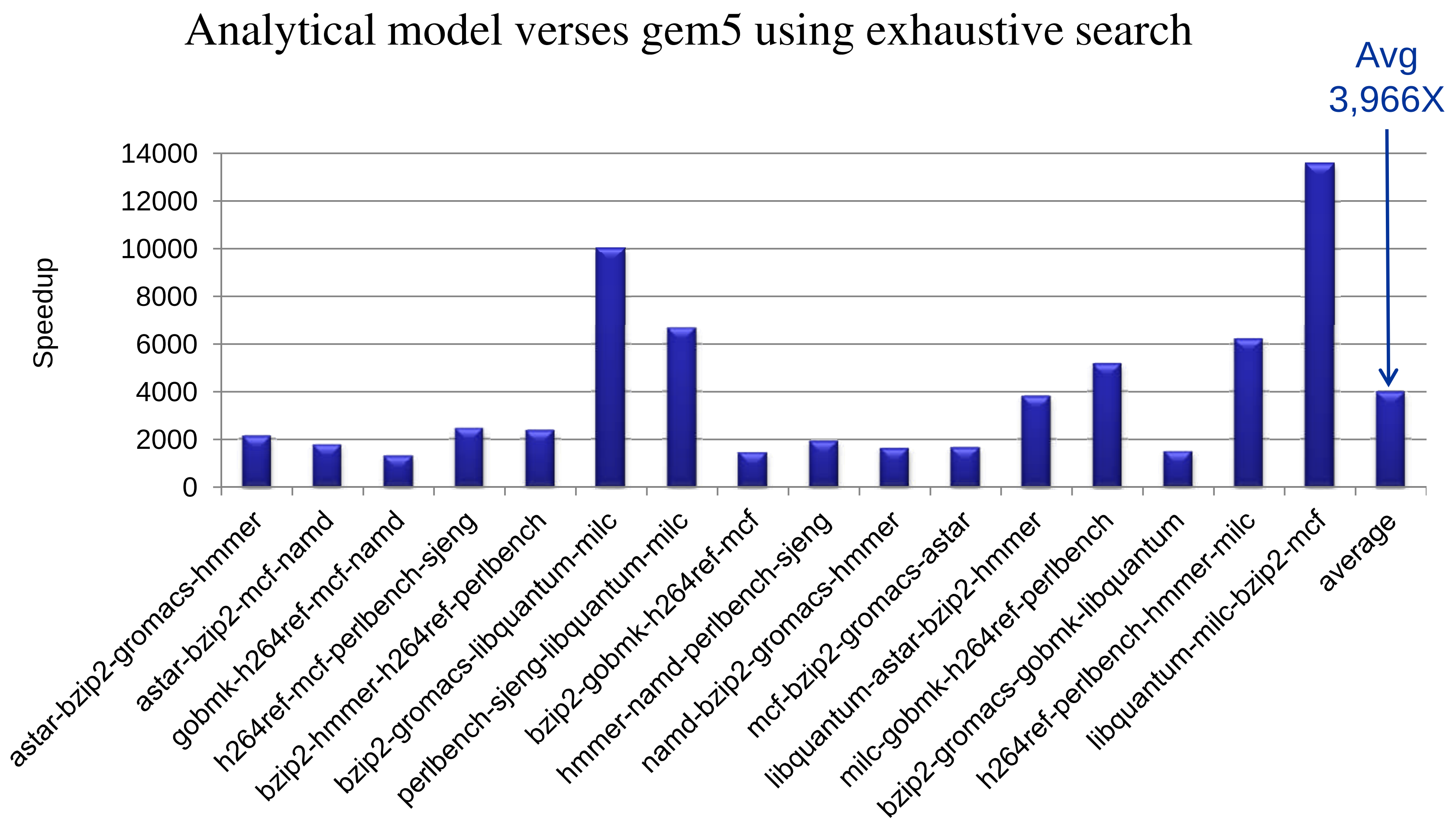


Accuracy Evaluation of Analytical Model

Compared average LLC miss rates for four cores determined by the analytical model verses gem5 for each configuration in CaPPS's design space.



Evaluation Time Speedup of Analytical Model



Conclusions and Future Work

- CaPPS: cache partitioning with partial sharing
  - Improve shared last-level cache (LLC) performance with low hardware overhead
  - Reduced average LLC miss rates by:
    - 20%-26% as compared to baseline configurations
    - 17% as compared to private partitioning
  - Developed analytical model for fast CaPPS design space exploration
    - Small errors: -0.73% on average
    - Average speedup: 3,966X as compared to a cycle-accurate simulator
- Future work
  - Extend analytical model to optimize for any design goal
  - Leverage offline analytical results to guide online scheduling
  - Extend CaPPS to proximity-aware cache partitioning for caches with non-uniform access