

Characterizing Embedded Applications via Multicore Reuse Distance Analysis

Meng-Ju Wu, Minshu Zhao, Mike Badamo, Jeff Casarona, and Donald Yeung
University of Maryland at College Park

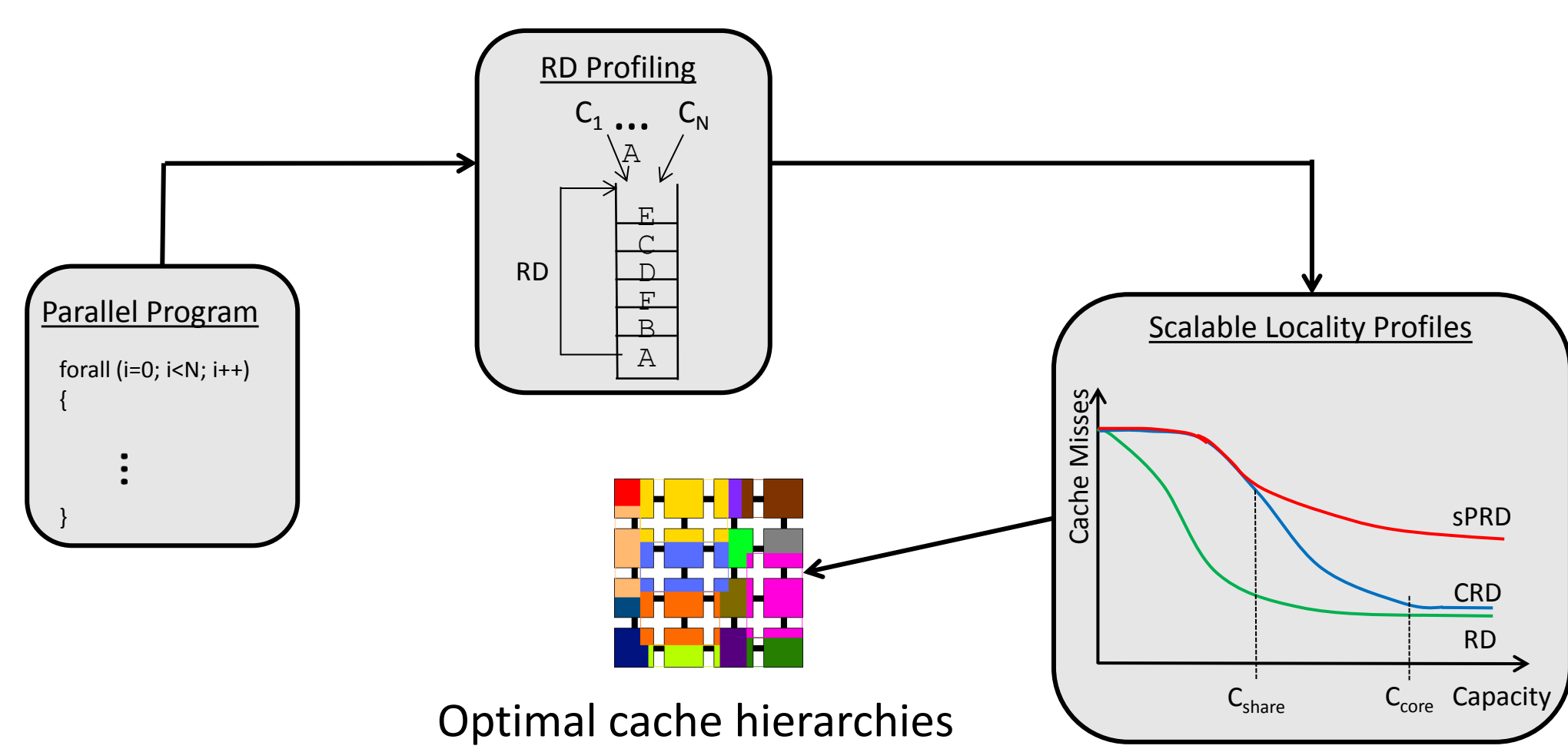
Overview

- Develop a model of parallel program locality that characterizes parallel embedded program behavior for performance and power consumption analysis
 - The model provides insights into memory behavior of parallel programs
 - The model can determine very good or optimal cache configurations that can drive application-specific system design and/or on-line cache reconfiguration
 - The model can determine the best degree of parallelism to exploit at runtime to achieve high efficiency

Locality Profile Model Details

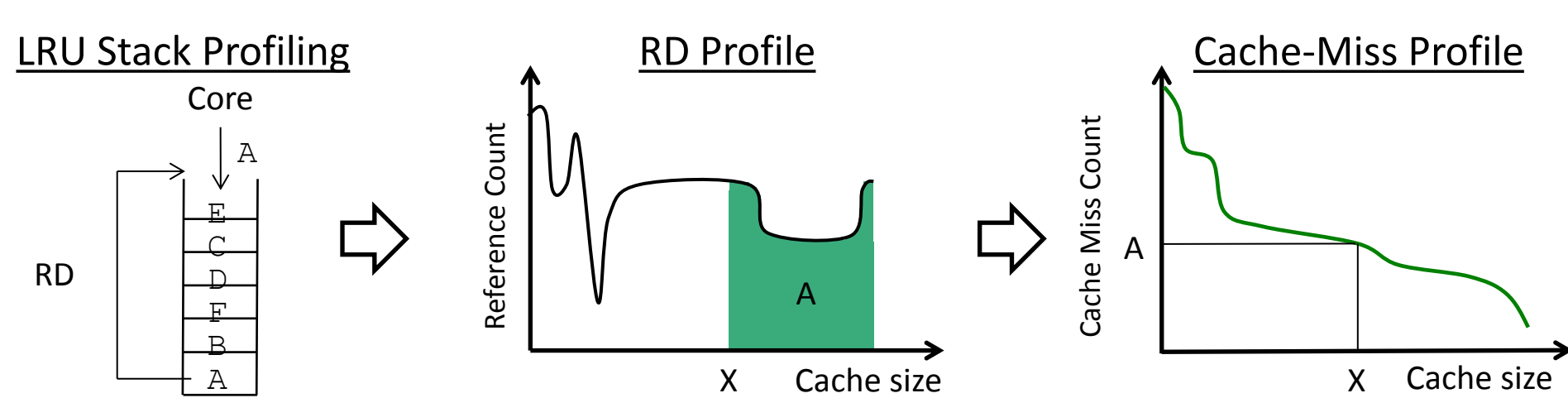
- Analyses are based on reuse distance (RD), extended to handle parallel programs to capture intra-thread locality as well as inter-thread interactions
- Focus on loop-based parallel programs (thread symmetry makes analyses tractable)
- A small number of profiles can predict performance across different cache sizes, core counts, and problem sizes

Assess 1000s of cache configurations to explore design spaces rapidly



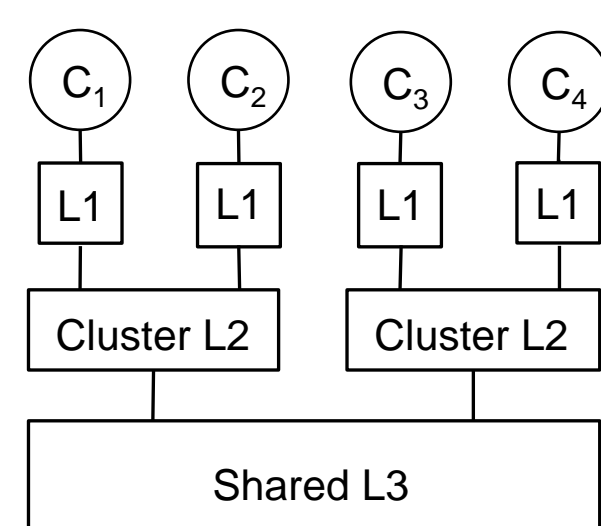
Profiling Techniques

- Uniprocessor Locality:** Reuse distance (RD) is measured by playing a sequential program's memory reference stream onto an LRU stack, and recording each memory reference's stack depth.
 - The histogram of all RD values across an entire program is the **RD profile** which can predict cache performance: the area under the RD profile beyond a reuse distance—and equivalently a cache capacity—X, is the number of cache misses for cache size X.

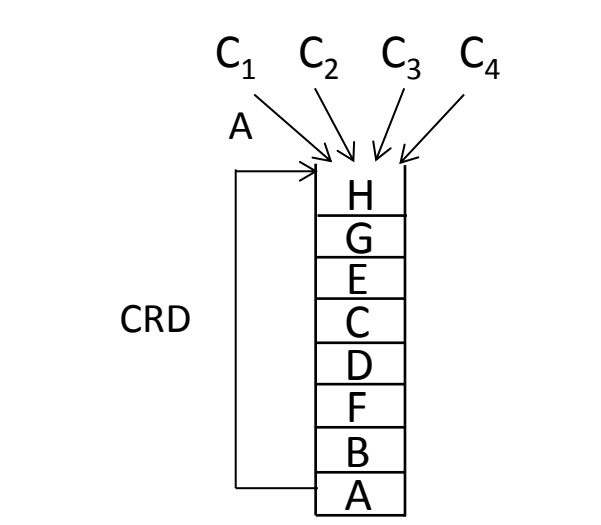


A single locality profile can predict cache misses at any cache size

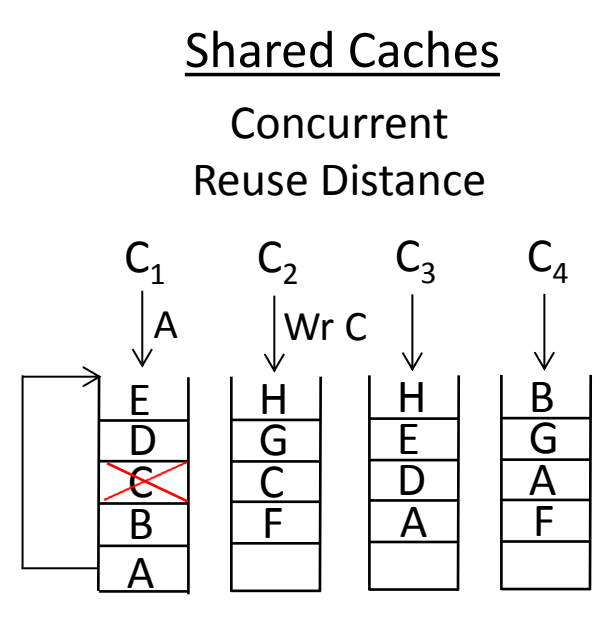
- Multicore Locality:** Compared to uniprocessors, must:
 - Handle multiple memory reference streams
 - Handle different types of caches, e.g. shared vs. private vs. cluster



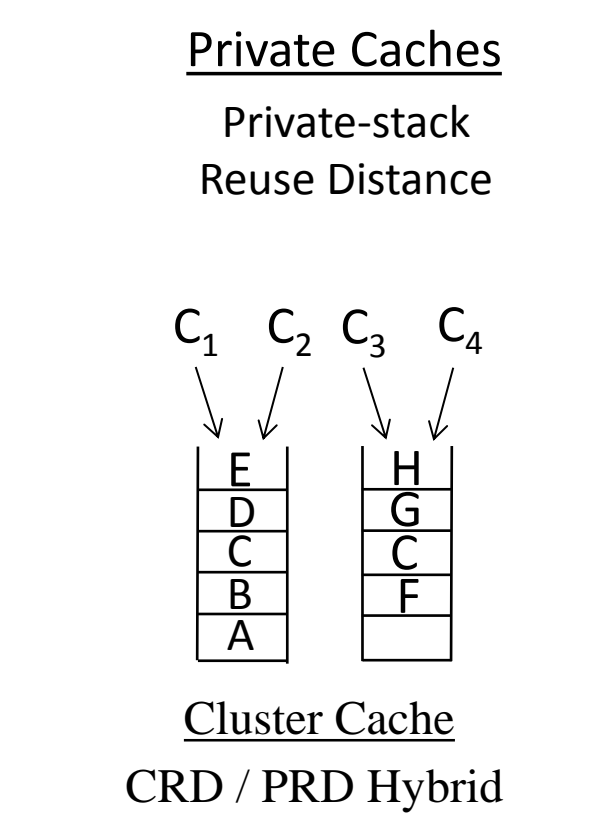
- Concurrent Reuse Distance (CRD)**
 - Quantifies locality in shared caches
 - CRD is computed by interleaving the memory reference streams of all cores, and playing the interleaved stream on a single LRU stack.



- Private-stack Reuse Distance (PRD)**
 - Quantifies locality in private caches
 - PRD is computed by replicating LRU stacks (one per core), and playing each core's memory reference stream on its local LRU stack.
 - For writes, all stacks are searched, and any remote copies of the requested block are invalidated to maintain coherence.



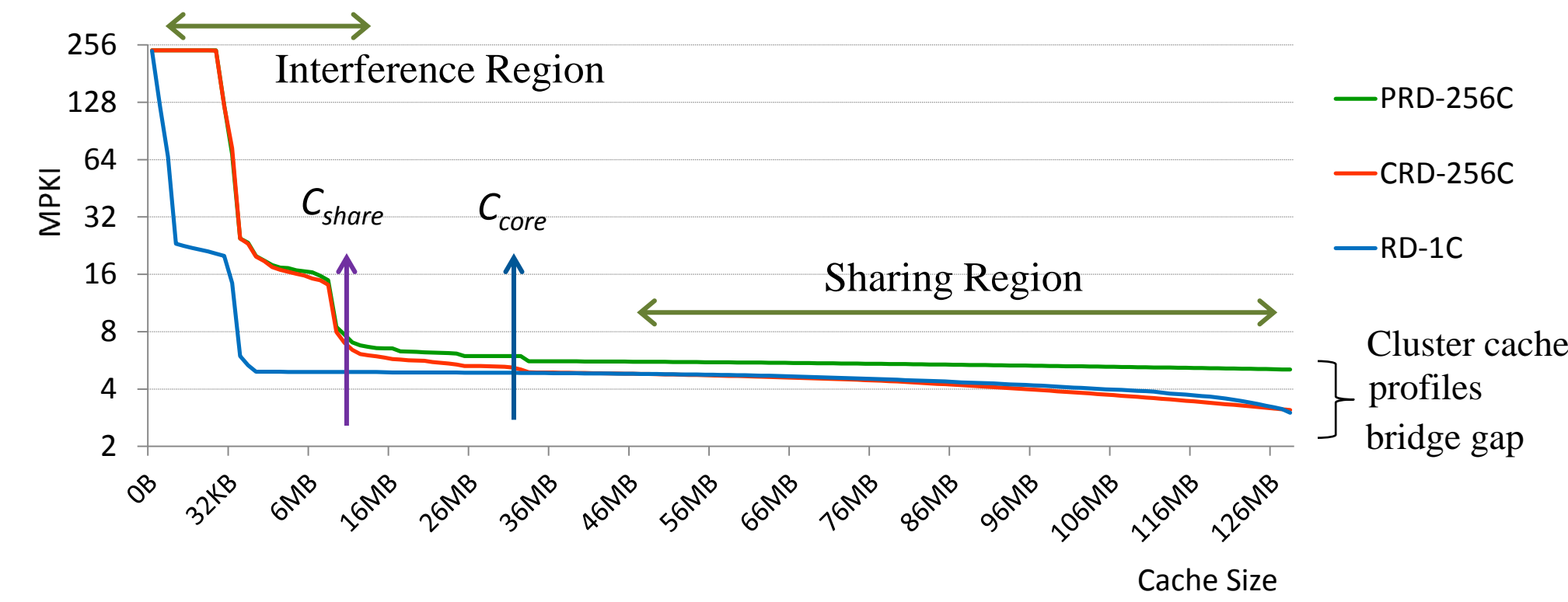
- Hybrid CRD/PRD**
 - Quantifies locality in cluster caches
 - Shared caches (cluster size = total # of cores) and private caches (cluster size = 1 core) are degenerate cases of cluster cache



CRD/PRD profiles can predict cache misses for shared/private/clustered caches of any size

Multicore Scaling Framework

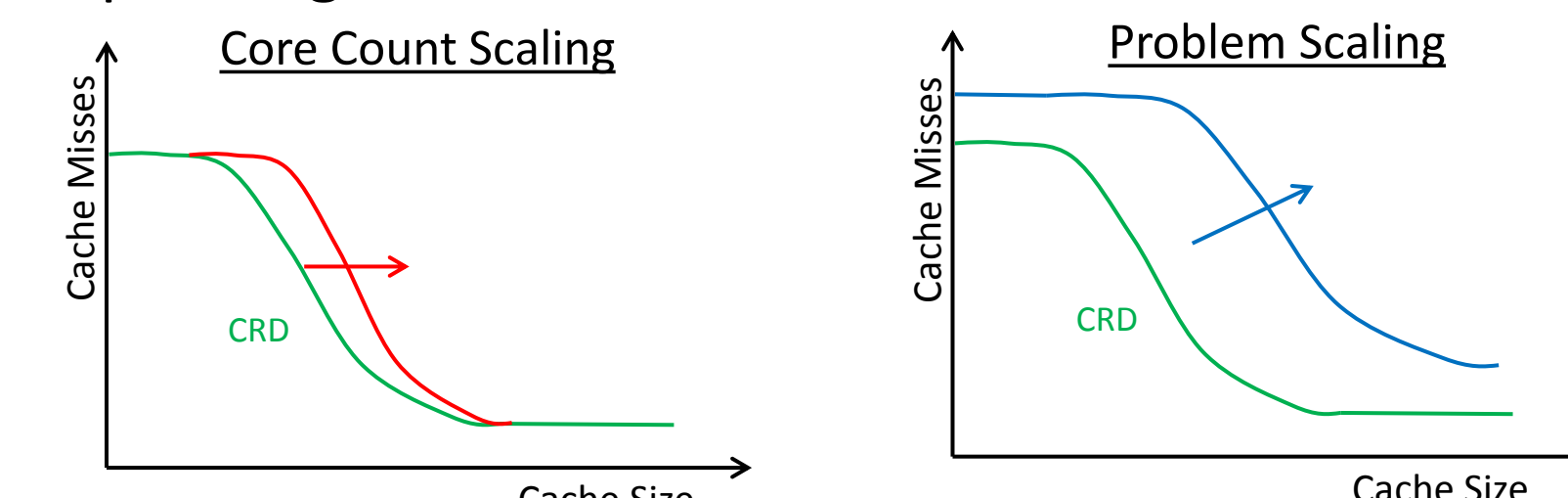
- Our insights on inter-thread interactions give rise to a model of locality for parallel programs, illustrated below.
 - This graph shows the PRD and CRD profiles (assuming 256 cores) and sequential program profile on the same plot. These profiles illustrate cache performance.



- Locality profiles typically decrease rapidly at certain capacities, marking different "working sets."
- Interference Region**
 - PRD and CRD profiles are coincident due to very little sharing at small cache sizes
 - CRD/PRD profiles are linearly shifted versions of the sequential profile
- Sharing Region**
 - PRD/CRD profiles diverge due to overlap suppressing dilation in CRD caused by sharing
 - Linear scaling continues in the PRD profile, opening a gap between PRD and CRD which is due to replication across private stacks as well as invalidations.
- Demarcation capacities have special meaning
 - C_{core}: Parallel working set size
 - C_{share}: Sharing granularity

Profile Prediction

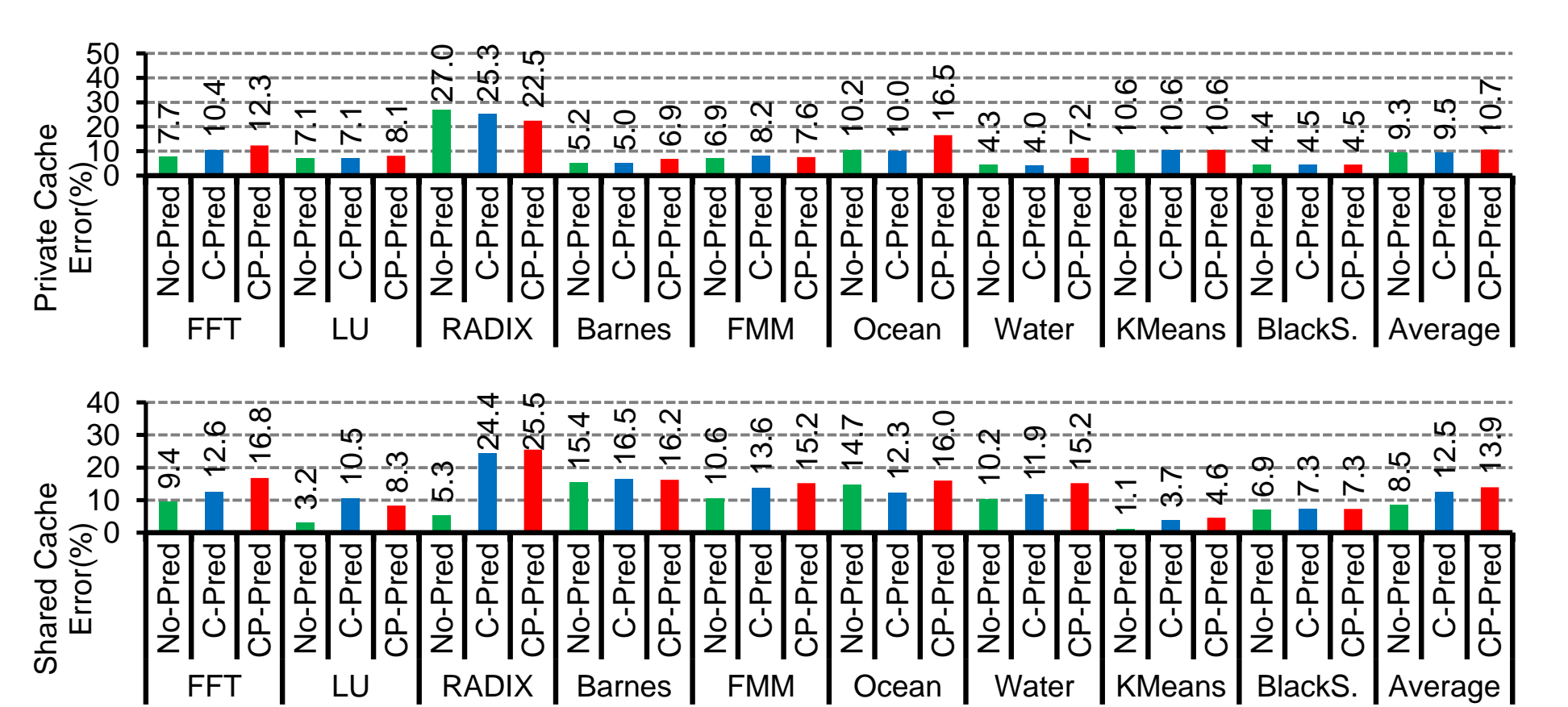
- Machine and problem scaling tend to shift locality profiles in a **shape-preserving** fashion
 - When scaling core count, memory interleaving between threads increases, which causes greater destructive interference. For programs with symmetric threads, this phenomenon is systematic because all threads exhibit similar locality. Therefore causing **shape-preserving shift**
 - Problem scaling can also cause systematic profile shift due to similar memory access patterns on larger data structures
- Reference Groups** technique: Profiles of scaled configurations can be predicted by "diffing" small-scale profiles, and extrapolating observed shift rate



Scaled profiles can be predicted by extrapolating observed shift in small-scale profiles

Validation Study

- Validated accuracy of locality profiles across 9 parallel benchmarks
 - Considered a large architecture-application design space consisting of 8 core counts, 5 private L2 sizes, 6 shared L3 sizes, and 4 problem sizes, 3,168 configurations in total
 - Simulated all configurations on a detailed architectural simulator
 - Predicted cache performance for all configurations using as few as 72 locality profiles (CP-Pred)



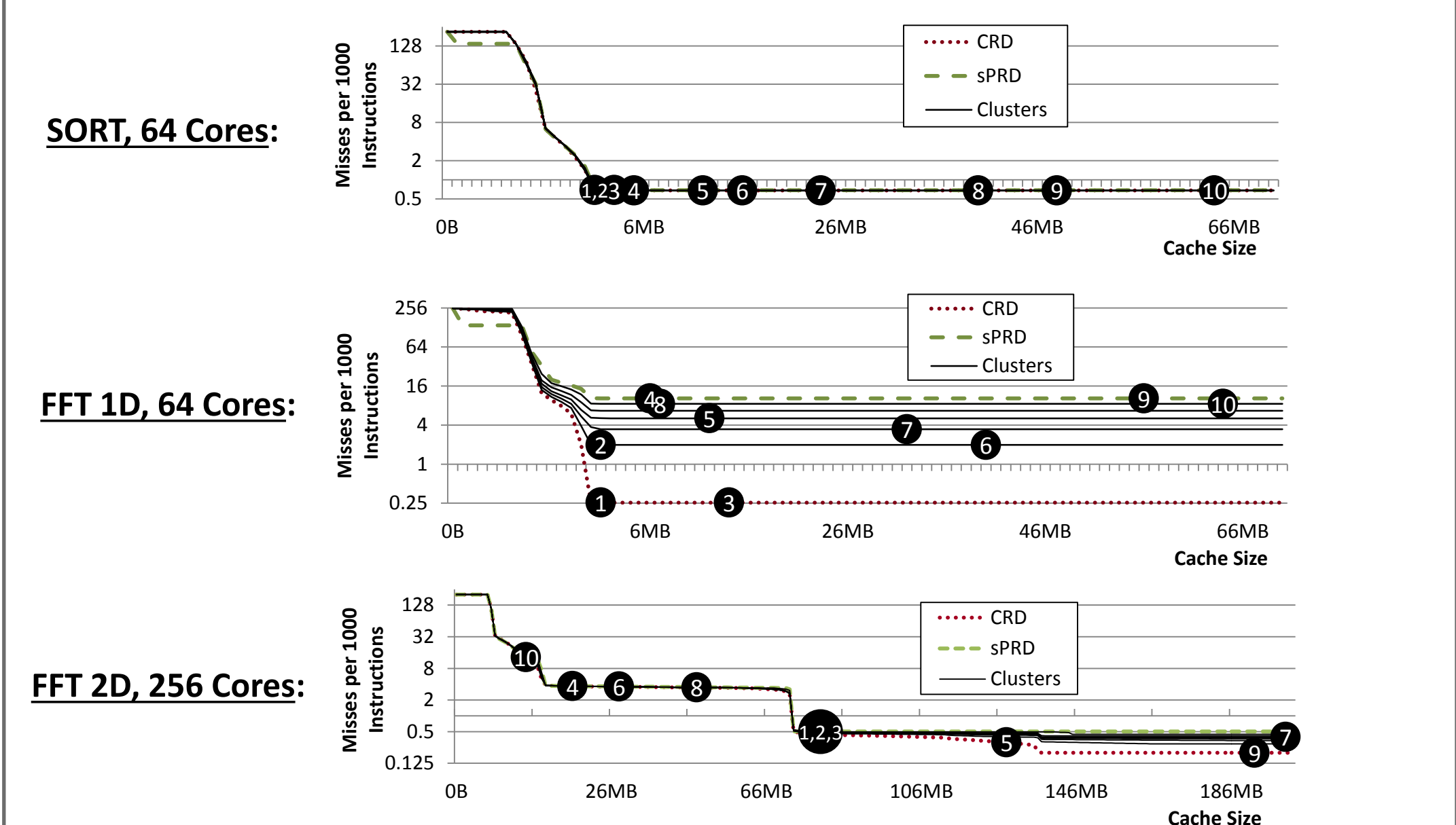
Locality profiles can estimate multicore cache performance (MPKI) to within 12.2% of simulations on average

Predicting Performance and Power

- We created models for predicting performance (CPU execution time) and power consumption from the cache-miss counts that our locality profiles provide to identify optimal configurations
- CPU Execution Time Prediction**
 - CPU execution time is predicted based on average memory access time (AMAT) from the cache-miss counts that our locality profiles provide
- Power Consumption Prediction**
 - For static power, we use the values that McPAT computes for each of the architectural components in the CPU
 - For dynamic power, we use the McPAT-computed values for per-event dynamic energy. The event counts are derived from the locality profiles

Best Configurations Results

- Predict power efficiency using our locality profiles for three PERFECT kernels.
 - Compute execution time and power consumption from profiles
 - Compute throughput (GIPs, or billions of instructions per second) from CPU execution time
 - Compute power efficiency (GIPs/Watt) from throughput and power
- For each PERFECT kernel, from all possible predicted configurations, we identify the configuration that achieves the highest power efficiency.
- We chose 9 other (presumably non-optimal) configurations at random for comparison.



- Validate the quality of the predictions made by our locality profiles
 - Using Graphite, we simulated the 10 configurations including the one with highest predicted power efficiency, and measured the achieved power efficiency on Graphite.

