Contributions
- Shared last-level cache partitioning for balanced data-parallel applications
- Balanced allocation is suboptimal for balanced programs
- Increasing allocation for one thread at a time improves utilization
- High imbalance helps both preferred and un-preferred threads
  - Preferred thread benefits because working set now fits into partition
  - Un-preferred threads benefit by using the data left behind in the preferred partition
- Prioritizing each thread in turn ensures balanced progress
- 17% drop in miss rate, 8% drop in execution time on average for 4-core 8MB cache
- Negligible overheads

Motivation
Last level cache partitioning heavily studied for multiprogramming workloads
- Multithreading different than multiprogramming
  - All threads have to progress equally
  - Pure throughput maximization is not enough
- Data-parallel threads are similar to each other in their data access patterns

Only way to improve utilization is through imbalance in allocation
- Preferred thread benefits since increased allocation covers working set
- Un-preferred thread benefits from data remaining in preferred partition

Evaluation
4-core CMP with 32 way shared L2, 9 data-parallel workloads from PARSEC AND SPEC OMP suites

Baselines
- Compared to a statically equi-partitioned cache and a CPI-based adaptive partitioning scheme
- Misses and execution time normalized to an un-partitioned cache

Evaluation Stage
- Triggers at the start of a new program phase
- Divide the cache sets into equal-sized segment
- Each segment with a different level of imbalance
- A segment for un-partitioned cache
- Each core is prioritized in turn
- Select configuration with least number of misses

Stable Stage
- Maintain the chosen configuration till the next program phase change
- Choose preferred thread in round-robin manner

Method
2-Stage Partitioning

Overheads
- Per-segment way partitioning and counters
- Program phase detection
- Evaluation stage overhead for small cache (1 % ave., 5 % max.)

Conclusion
Effective cache utilization and balanced progress for data-parallel applications through:
A. High Imbalance in partitions
B. Prioritizing each thread in turn

Acknowledgements
Funded by: NSF (CNS-0751153, CNS-1117726, & OCI-1216809)
& the DOE (DE-FC02-12ER26104)
Thanks to: T.N. Vijaykumar, Malek Musleh