

*Thread Criticality Predictors
for Dynamic Performance, Power,
and Resource Management
in Chip Multiprocessors*

Abhishek Bhattacharjee

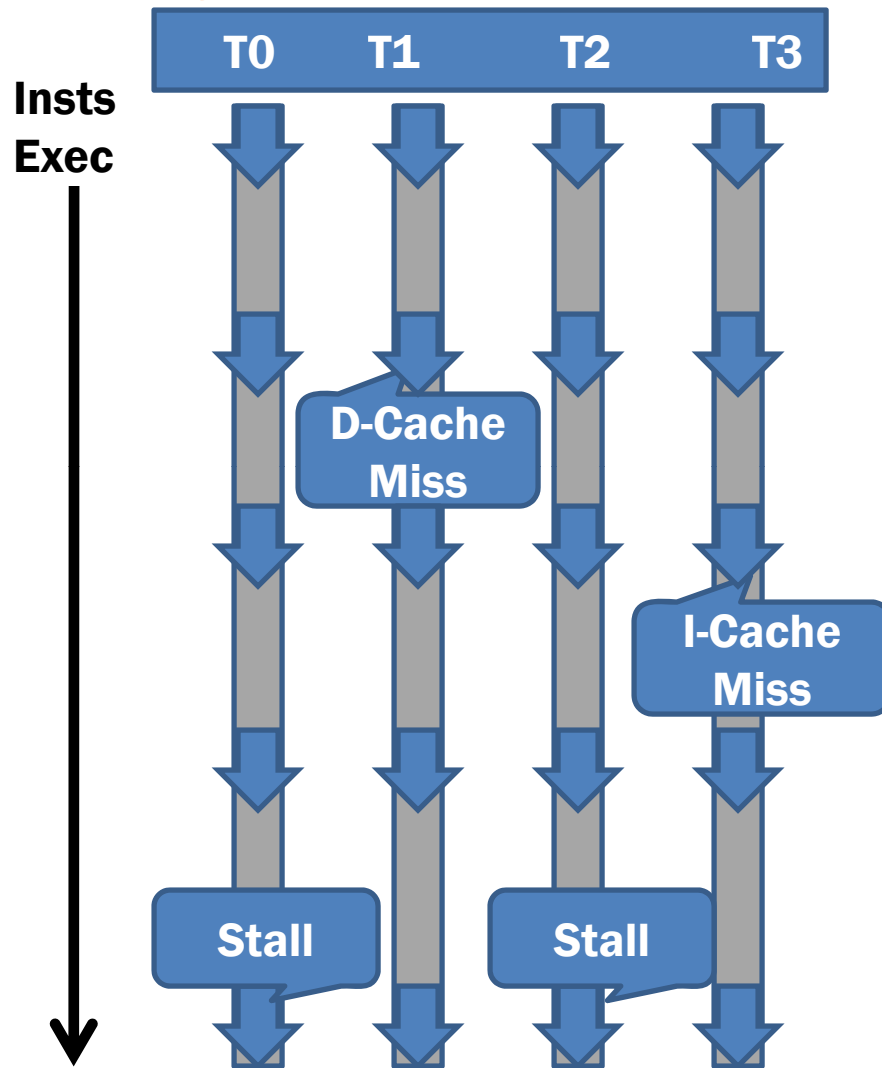
Margaret Martonosi

Princeton University



PRINCETON

Why Thread Criticality Prediction?



- Threads 1 & 3 are *critical* → Performance degradation, energy inefficiency
- Sources of variability: algorithm, process variations, thermal emergencies etc.
- With thread criticality prediction:
 1. Task stealing for performance
 2. DVFS for energy efficiency
 3. Many others ...



Related Work

- Instruction criticality [Fields et al., Tune et al. 2001 etc.]
- Thrifty barrier [Li et al. 2005]
 - Our Approach:
 1. Also handles non-barrier code
 2. Works on constant or variable loop iteration size
 3. Predicts criticality at any point in time, not just barriers
- Meeting points [Car et al. 2008]
 - DVFS non-critical threads by tracking loop iterations completion rate across cores (parallel loops)



Thread Criticality Prediction Goals

Design Goals

1. Accuracy
 - Absolute TCP accuracy
 - Relative TCP accuracy
2. Low-overhead implementation
 - Simple HW (allow SW policies to be built on top)
3. One predictor, many uses

Design Decisions

1. Find suitable arch. metric
2. History-based local approach versus thread-comparative approach
3. This paper: TBB, DVFS
Other uses: Shared LLC management, SMT and memory priority, ...



Outline of this Talk

- Thread Criticality Predictor Design
 - Methodology
 - Identify architectural events impacting thread criticality
 - Introduce basic TCP hardware
- Thread Criticality Predictor Uses
 - Apply to Intel's Threading Building Blocks (TBB)
 - Apply for energy-efficiency in barrier-based programs



Methodology

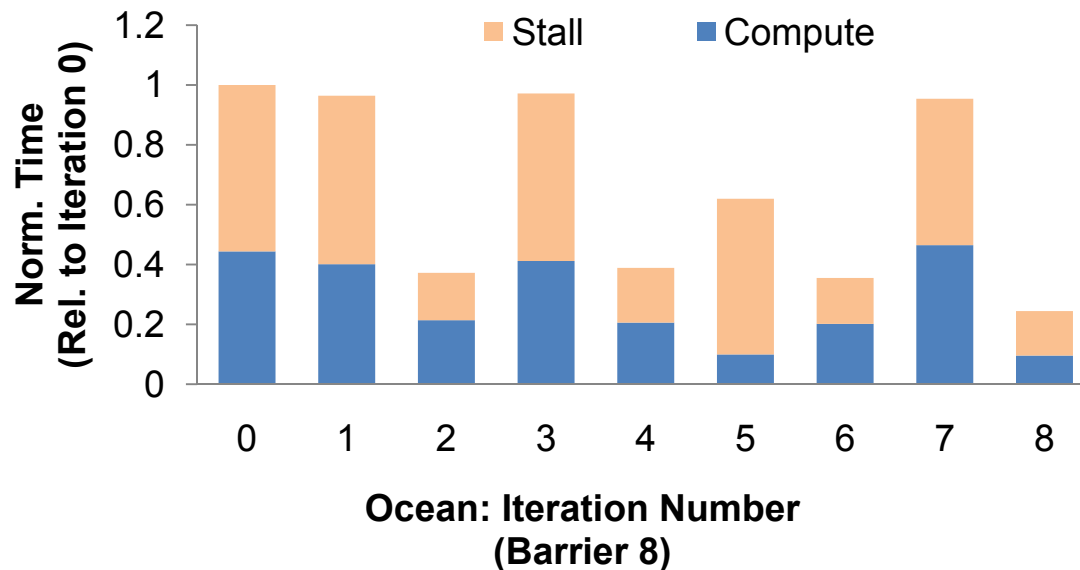
- Evaluations on a range of architectures: high-performance and embedded domains
- Full-system including OS
- Detailed power/energy studies using FPGA emulator

Infrastructure	GEMS Simulator	ARM Simulator	FPGA Emulator
Domain	High-performance, wide-issue, out-of-order	Embedded, in-order	Embedded, in-order
System	16 core CMP with Solaris 10	4-32 core CMP	4-core CMP with Linux 2.6
Cores	4-issue SPARC	2-issue ARM	1-issue SPARC
Caches	32KB L1 , 4MB L2	32KB L1, 4MB L2	4KB I-Cache, 8KB D-Cache



Why not History-Based TCPs?

- + Info local to core: no communication
- Requires repetitive barrier behavior
- Problem for in-order pipelines: variant IPCs

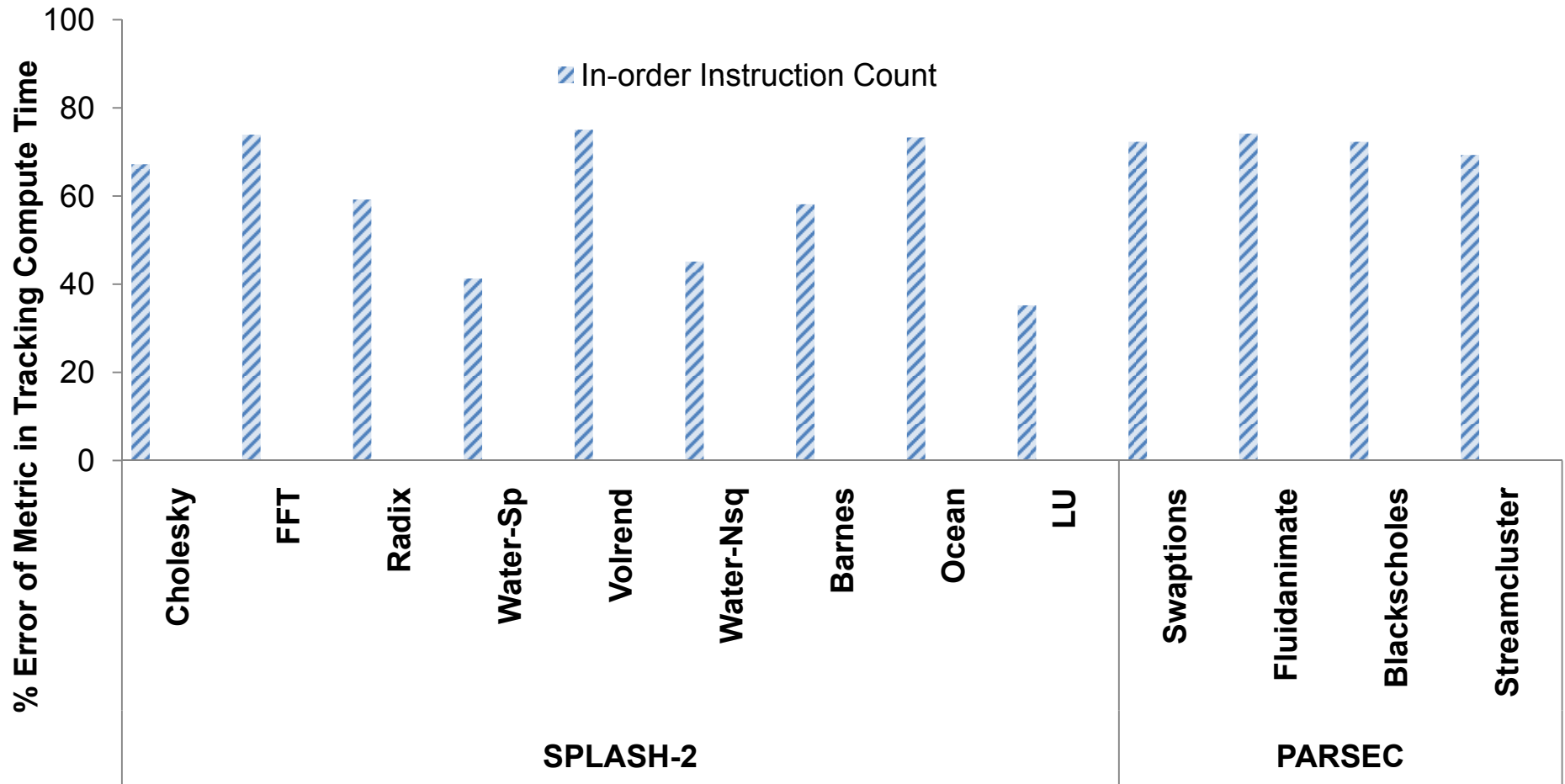


→ Time →

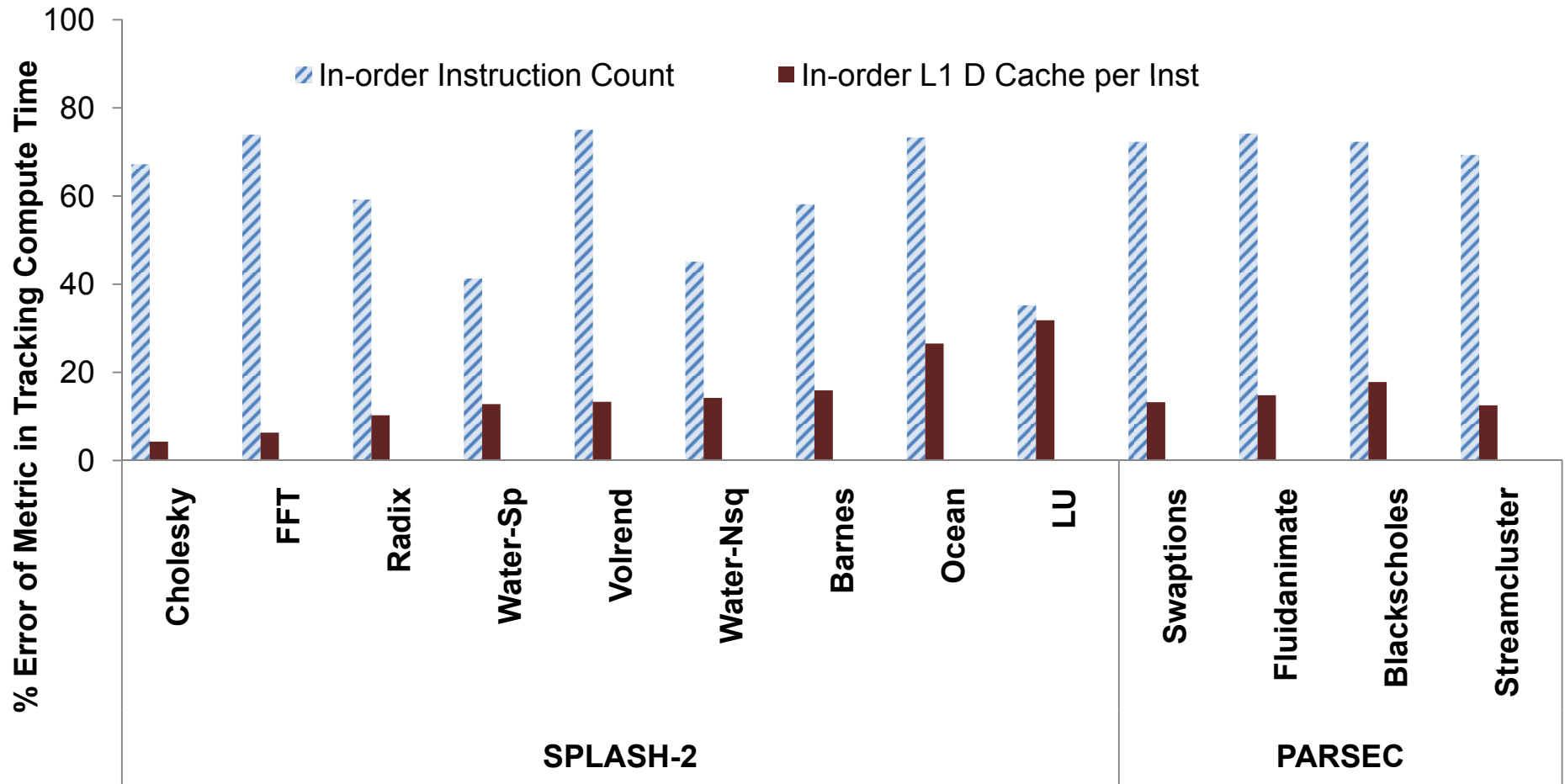


PRINCETON

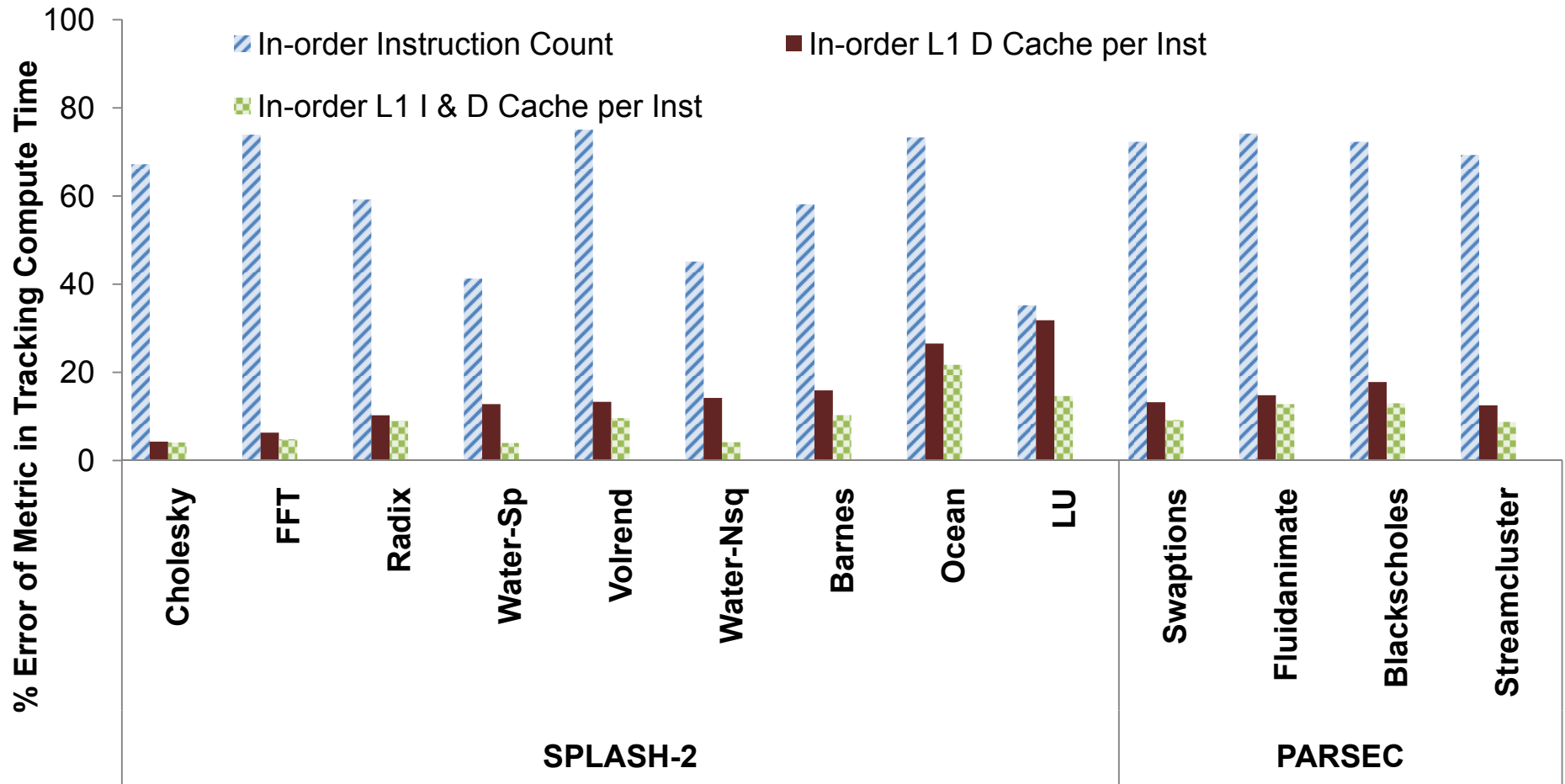
Thread-Comparative Metrics for TCP: Instruction Counts



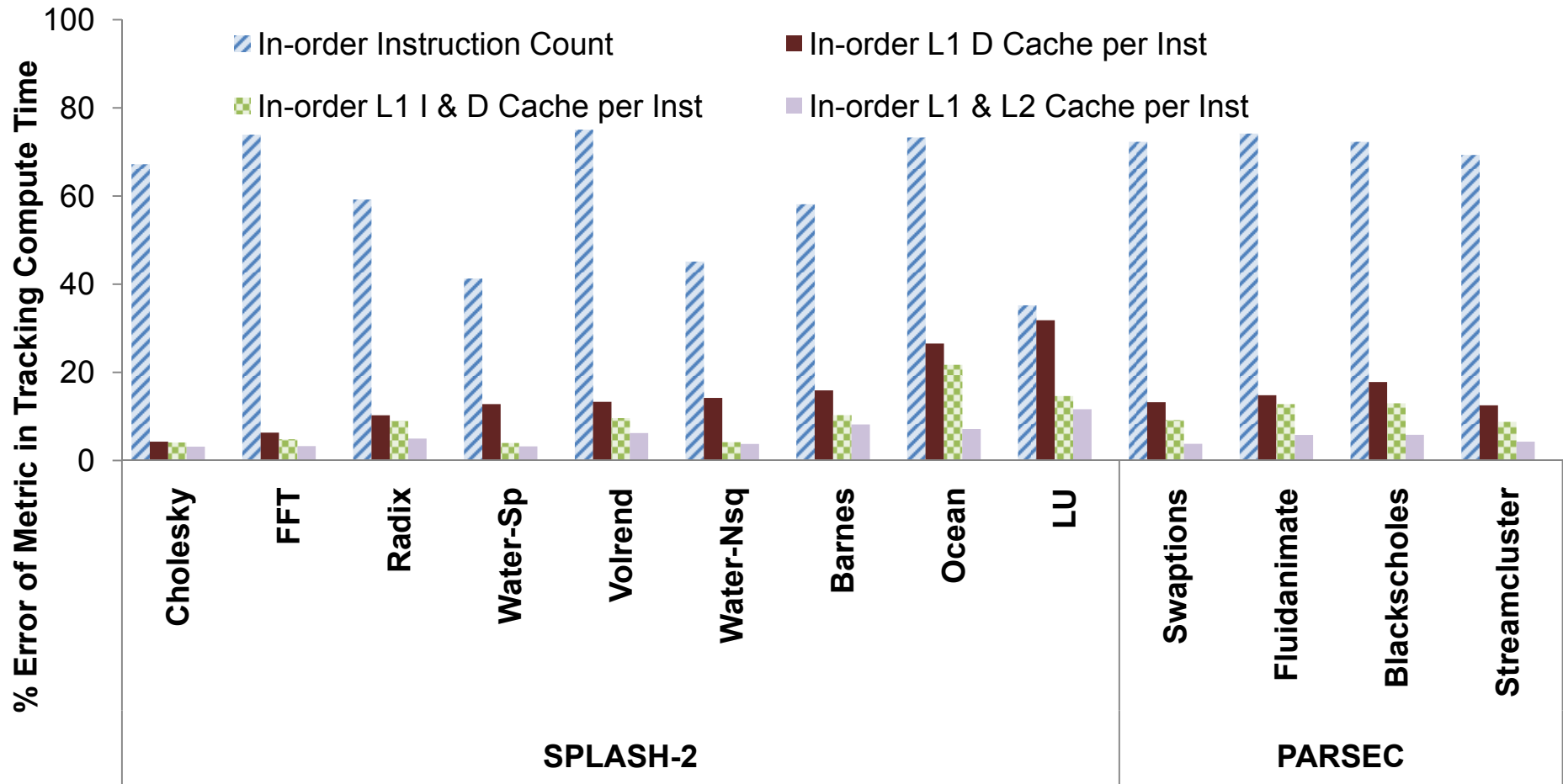
Thread-Comparative Metrics for TCP: L1 D Cache Misses



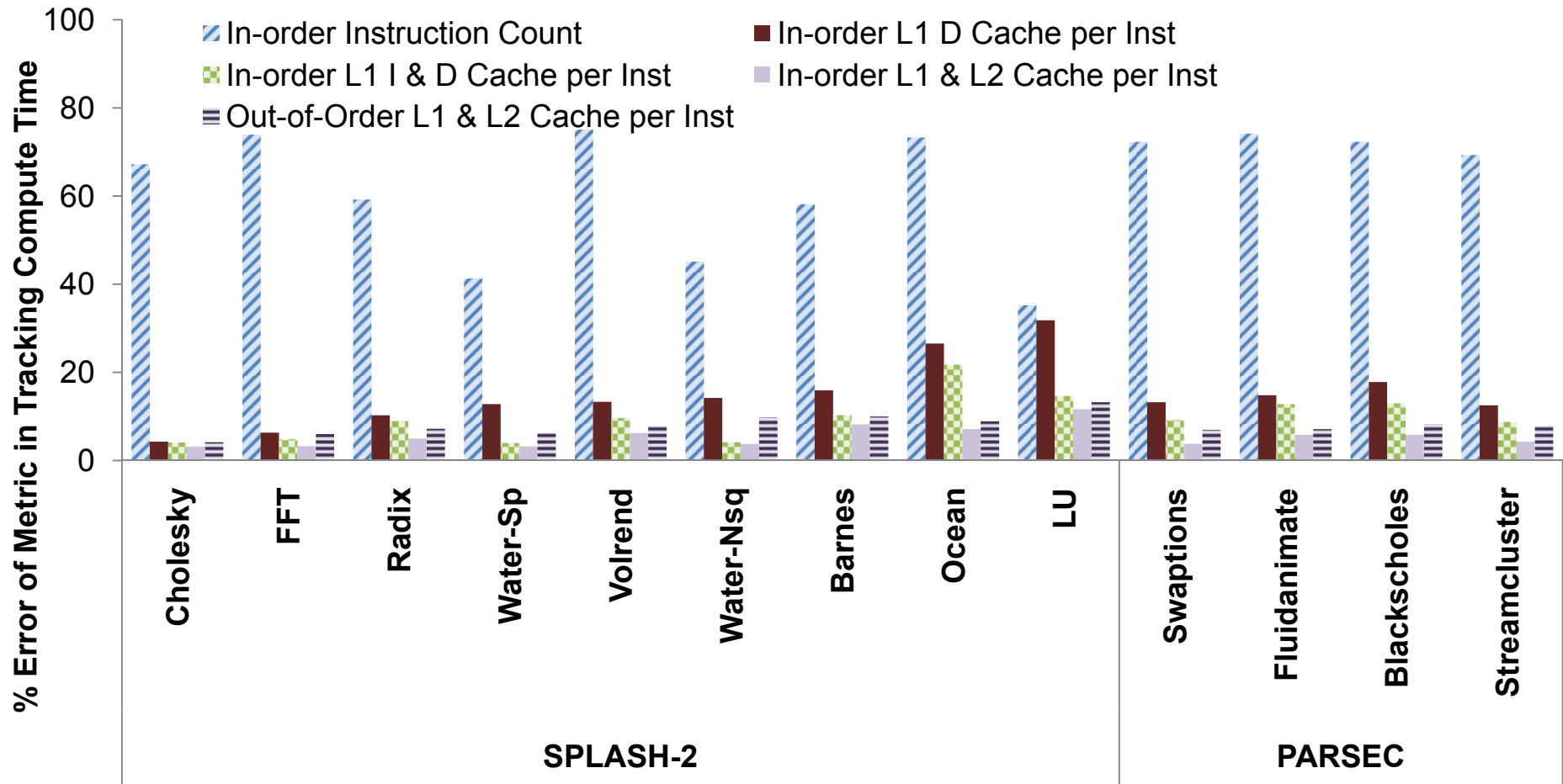
Thread-Comparative Metrics for TCP: L1 I & D Cache Misses



Thread-Comparative Metrics for TCP: All L1 and L2 Cache Misses



Thread-Comparative Metrics for TCP: All L1 and L2 Cache Misses

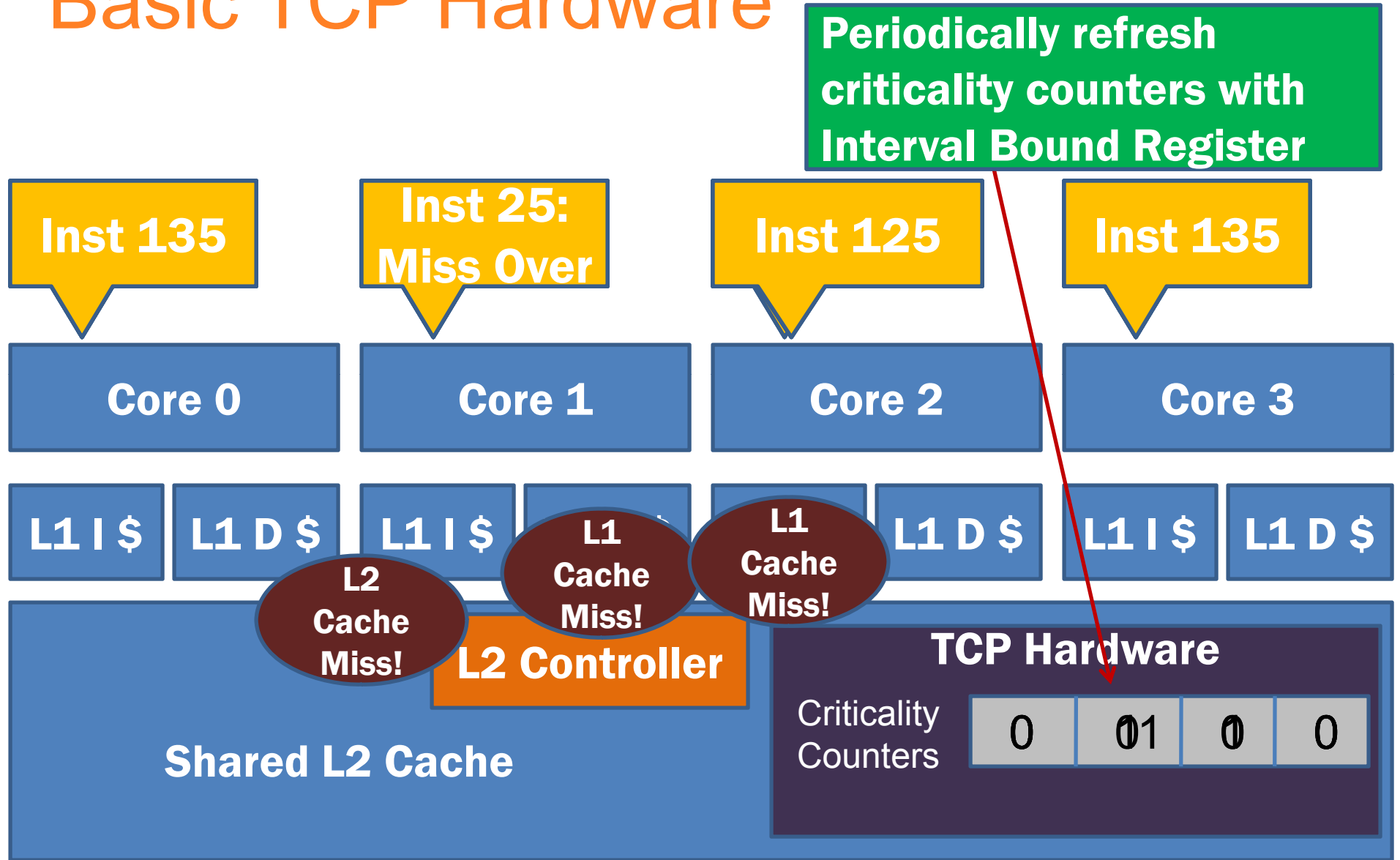


Outline of this Talk

- Thread Criticality Predictor Design
 - Methodology
 - Identify architectural events impacting thread criticality
 - Introduce basic TCP hardware
- Thread Criticality Predictor Uses
 - Apply to Intel's Threading Building Blocks (TBB)
 - Apply for energy-efficiency in barrier-based programs



Basic TCP Hardware



Outline of this Talk

- Thread Criticality Predictor (TCP) Design
 - Methodology
 - Identify architectural events impacting thread criticality
 - Introduce basic TCP hardware
- Thread Criticality Predictor Uses
 - Apply to Intel's Threading Building Blocks (TBB)
 - Apply for energy-efficiency in barrier-based programs

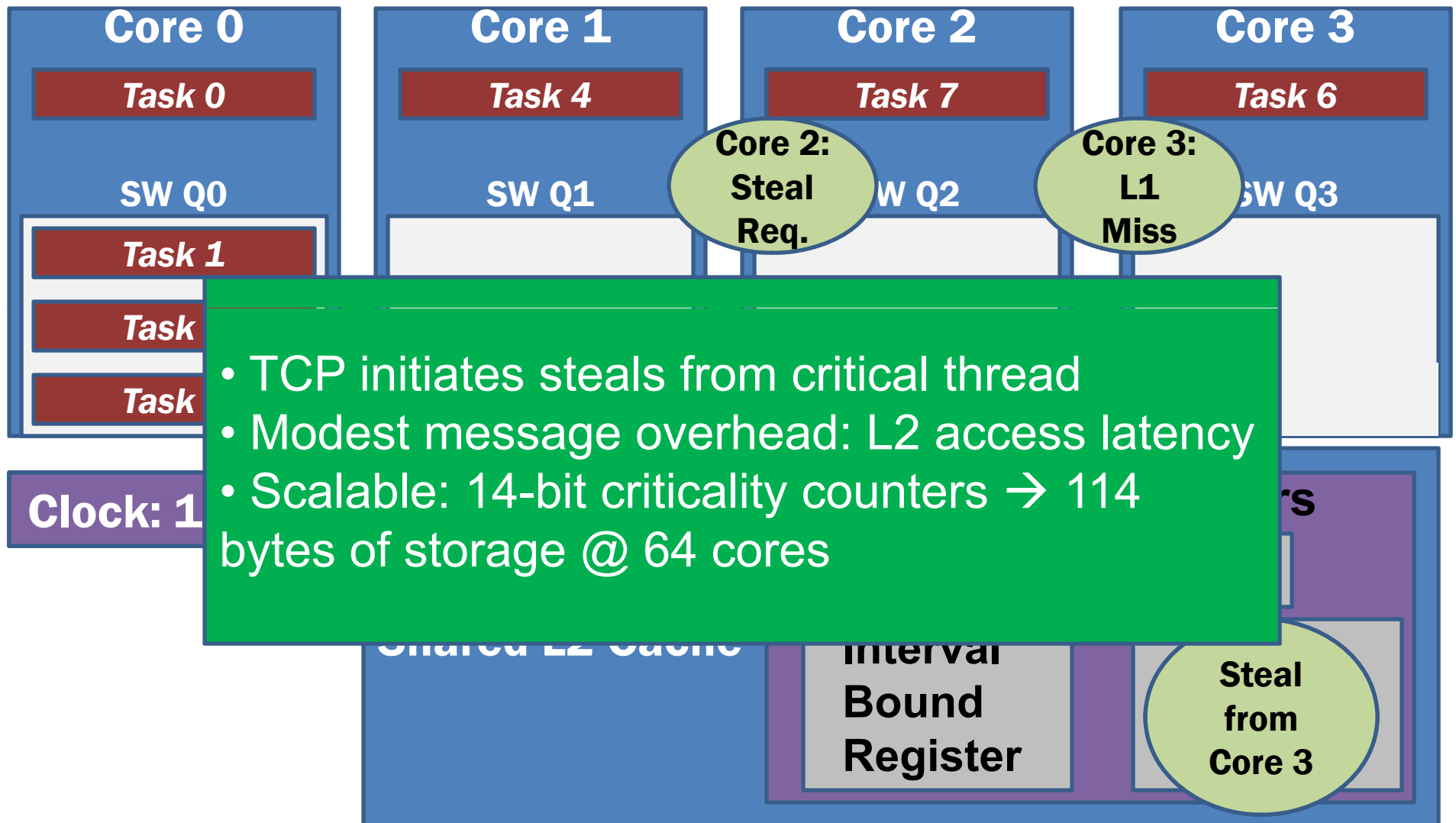


TBB Task Stealing & Thread Criticality

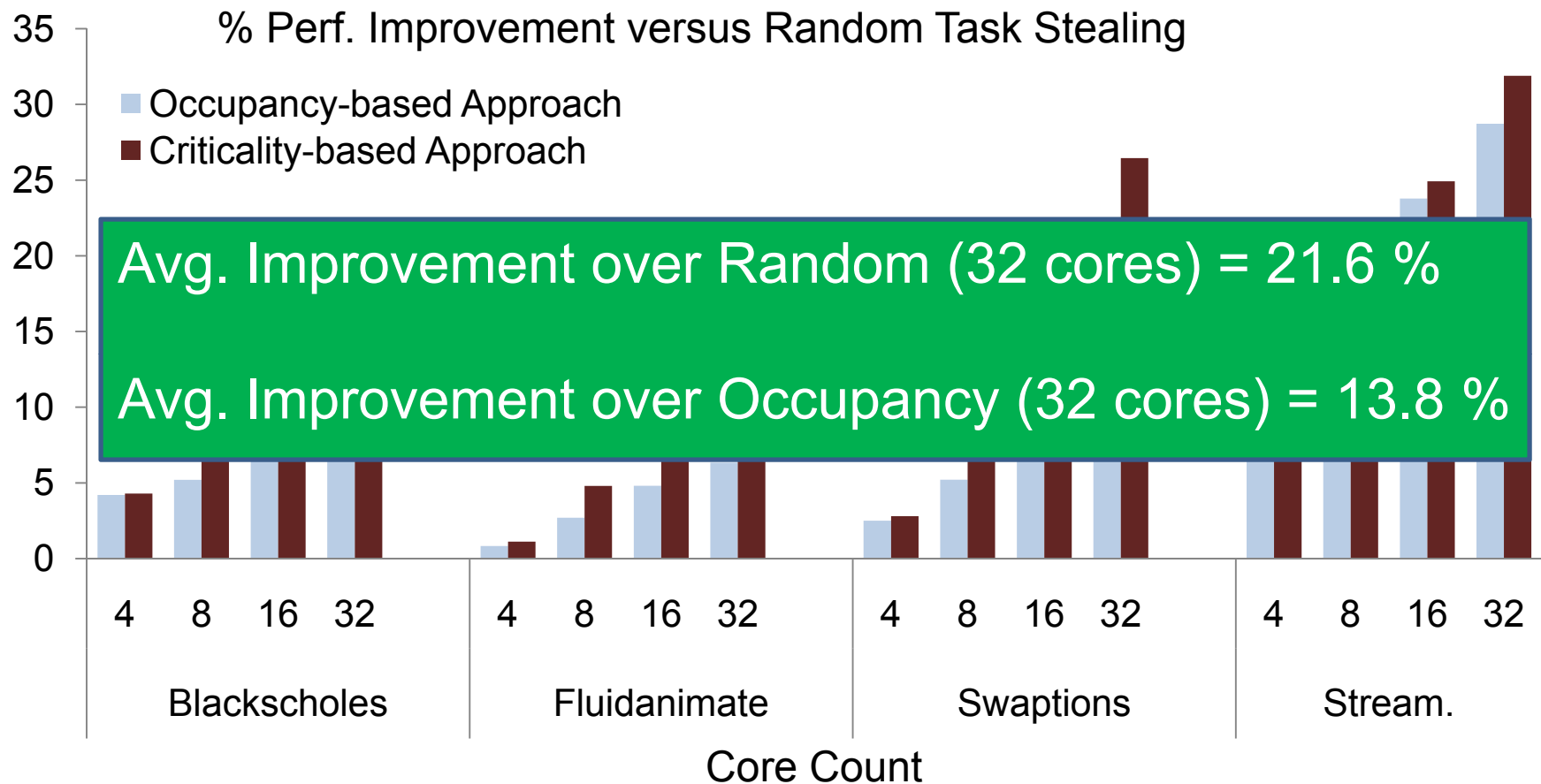
- TBB dynamic scheduler distributes *tasks*
- Each thread maintains software queue filled with tasks
 - Empty queue – thread “steals” task from another thread’s queue
- Approach 1: Default TBB uses *random* task stealing
 - More failed steals at higher core counts → poor performance
- Approach 2: Occupancy-based task stealing [Contreras, Martonosi, 2008]
 - Steal based on number of items in SW queue
 - Must track and compare max. occupancy counts



TCP-Guided TBB Task Stealing



TCP-Guided TBB Performance



- TCP access penalized with L2 latency

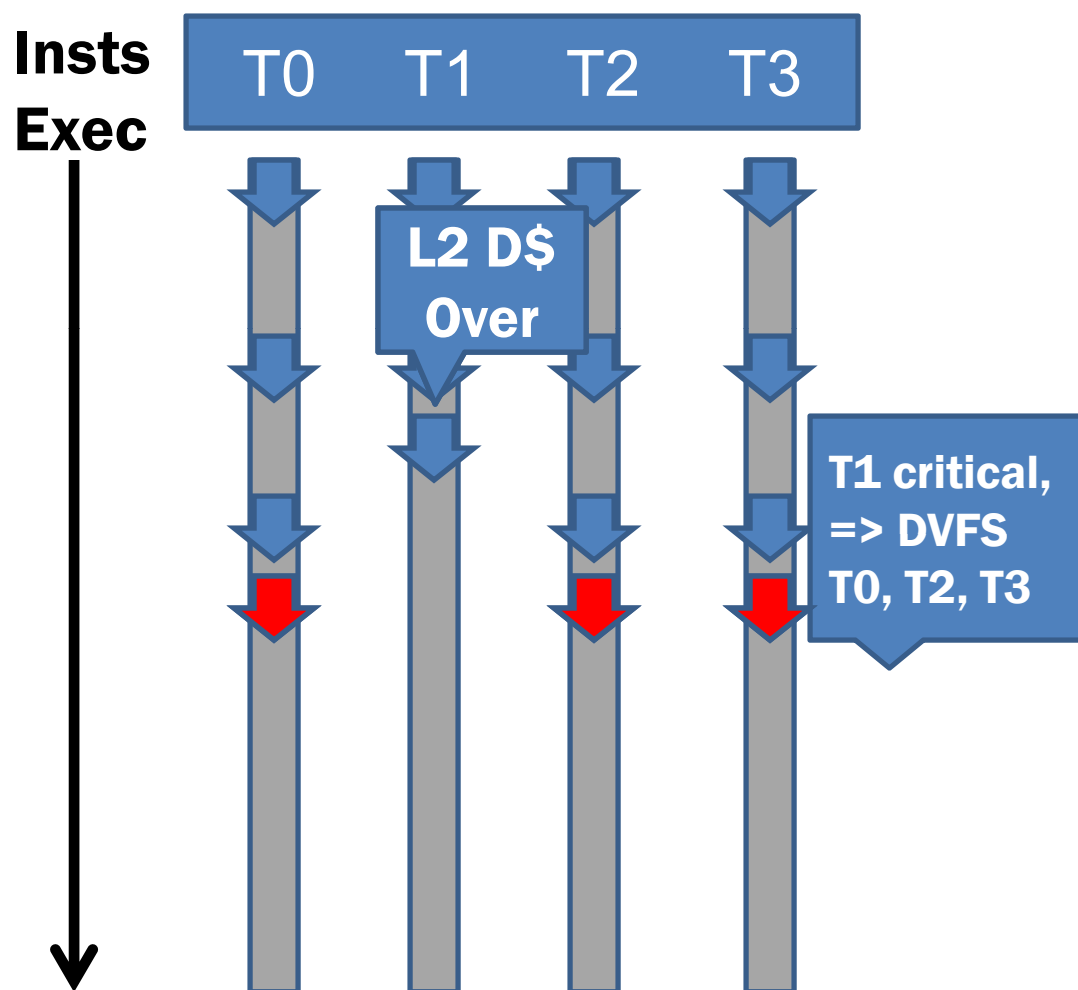


Outline of this Talk

- Thread Criticality Predictor Design
 - Methodology
 - Identify architectural events impacting thread criticality
 - Introduce basic TCP hardware
- Thread Criticality Predictor Uses
 - Apply to Intel's Threading Building Blocks (TBB)
 - Apply for energy-efficiency in barrier-based programs



Adapting TCP for Energy Efficiency in Barrier-Based Programs



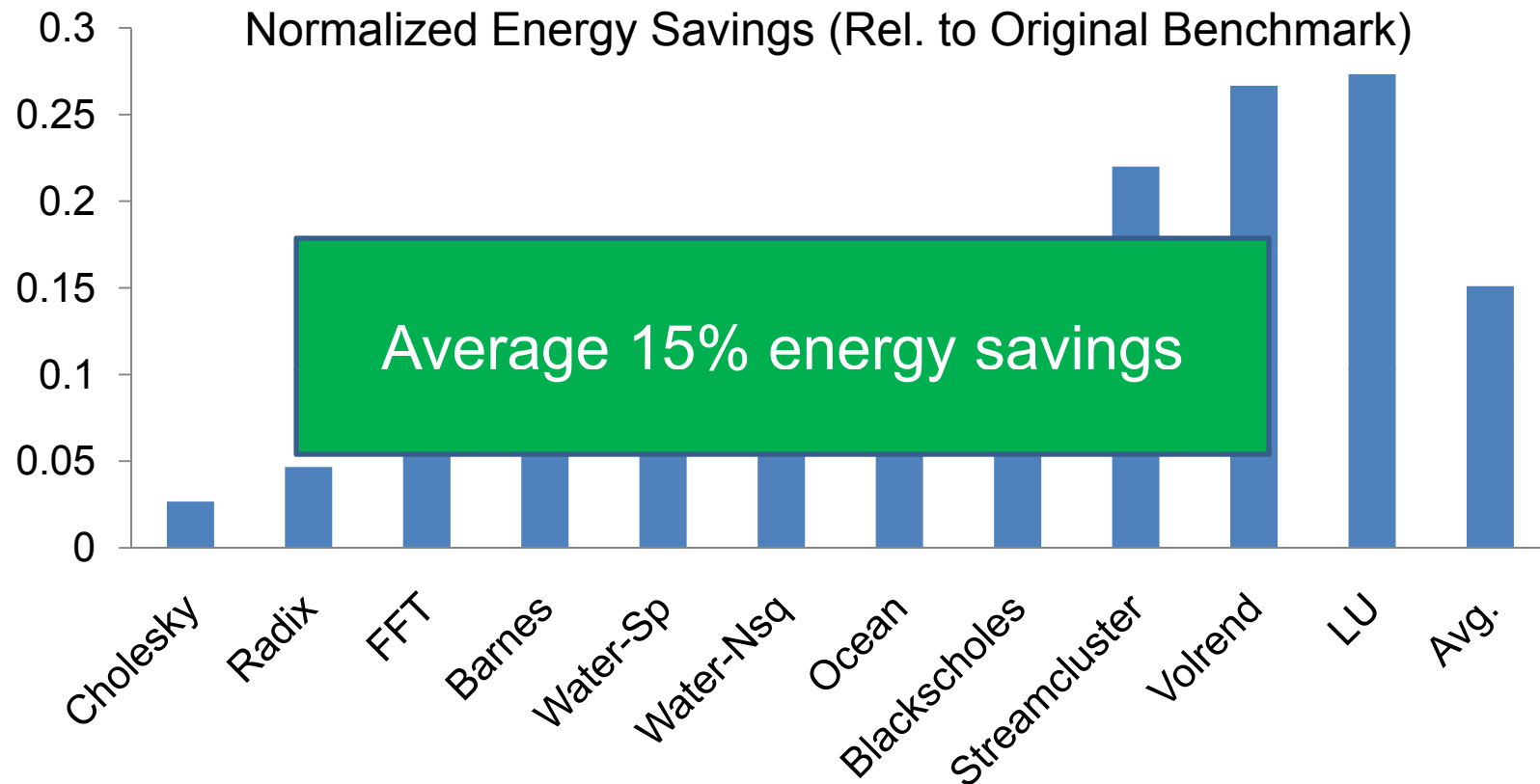
Approach: DVFS non-critical threads to eliminate barrier stall time

Challenges:

- Relative criticalities
- Misprediction costs
- DVFS overheads



TCP for DVFS: Results



- FPGA platform with 4 cores, 50% fixed leakage cost
- See paper for details: TCP mispredictions, DVFS overheads etc



Conclusions

- Goal 1: Accuracy
 - Accurate TCPs based on simple cache statistics
- Goal 2: Low-overhead hardware
 - Scalable per-core criticality counters used
 - TCP in central location where cache info. is already available
- Goal 3: Versatility
 - TBB improved by 13.8% over best known approach @ 32 cores
 - DVFS used to achieve 15% energy savings
 - Two uses shown, many others possible...

