Identifying and Mitigating Memory Resource Contention to Accelerate Data-Intensive Workloads

Justin Meza* Jichuan Chang† Parthasarathy Ranganathan† Onur Mutlu* (*CMU, †HP Labs)

Motivation / Background

- Parallel programs do multiple tasks at the same time
- Three fundamental bottlenecks in parallel code portion
  - Synchronization (locks, barriers, shared data)
  - Load imbalance (tasks with more work)
  - Resource contention (task prioritization)
- Past work has used synchronization and load imbalance bottlenecks to determine thread criticality
- In this work, we focus on using resource contention bottlenecks to identify and accelerate critical threads

Adaptive Critical Thread Selection (ACTS)

KEY IDEA

- Track resource contention in regions of memory (hot regions have high memory request delay)
- Identify threads that access hot regions as critical
- Prioritize predicted critical threads in hardware policies (caching, scheduling, prefetching, etc.) each quantum

Evaluation

SYSTEM

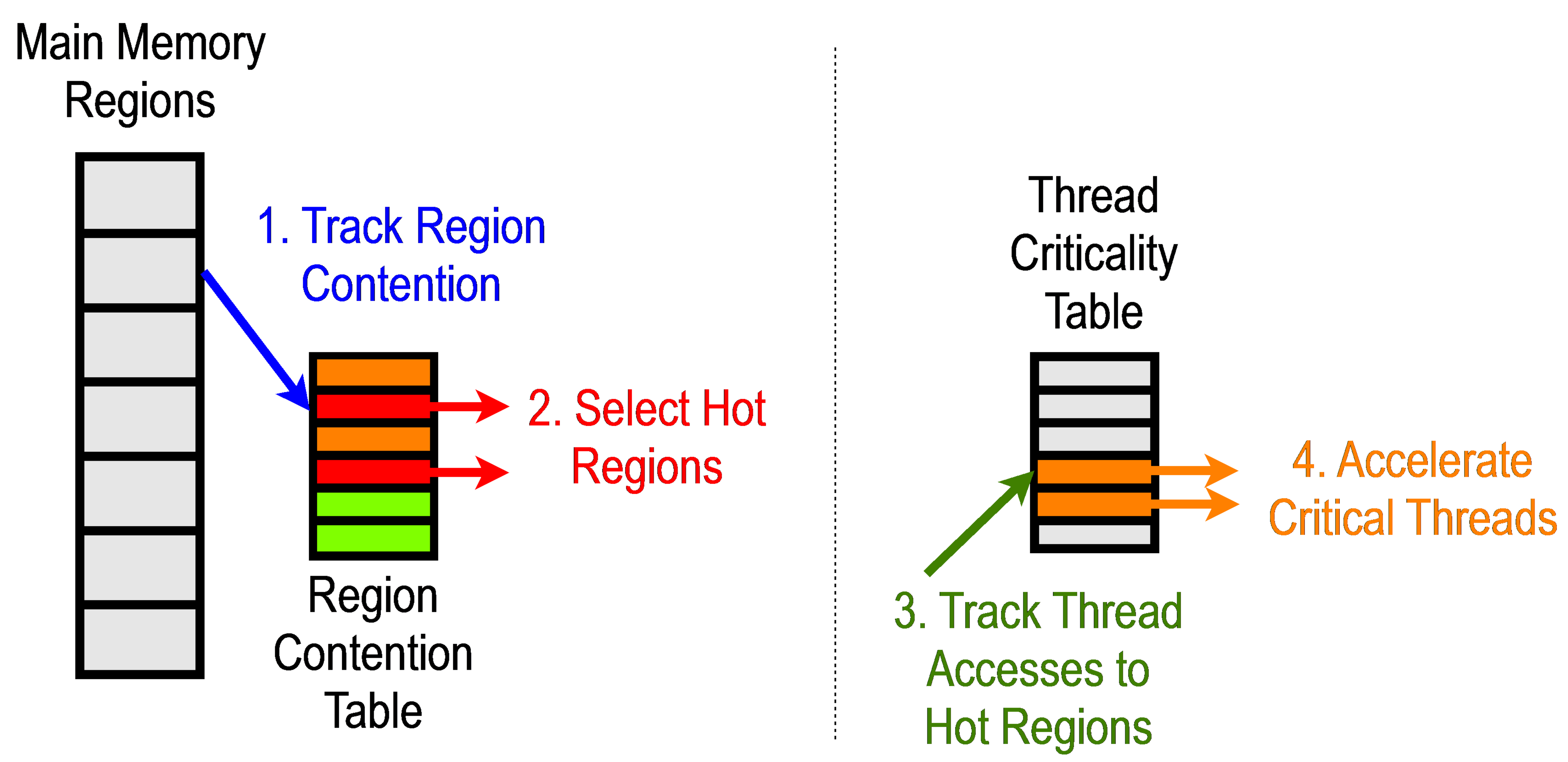
- 8-cores, 32MB DRAM cache, 8GB PCM main memory

WORKLOADS

- Traditional parallel programs (PARSEC)
- New, data-intensive workloads (GraphLab, MapReduce, Graph500)

CACHING POLICIES

- Baseline: Cache every accessed block
- CacheMiss: Cache blocks from thread with most cache miss latency
- Region: Cache blocks from the top two hottest regions of memory
- ACTS: Cache blocks from thread with the most hot region accesses



Adaptive Critical Thread Selection (ACTS)

- 11% better performance than baseline
- 6% better performance than CacheMiss-based policy
- 29% better energy efficiency due to only migrating data for critical thread