Data Sharing or Resource Contention: Toward Performance Transparency on Multicore Systems

Sharanyan Srikanthan
Sandhya Dwarkadas
Kai Shen

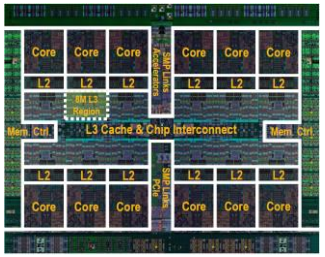
Department of Computer Science
University of Rochester

UNIVERSITY of ROCHESTER

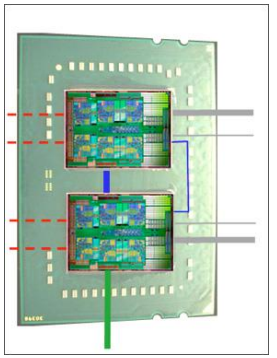
The Performance Transparency Challenge

Modern multicore systems…

Qualcomm Snapdragon…



IBM Power8's 12-core…



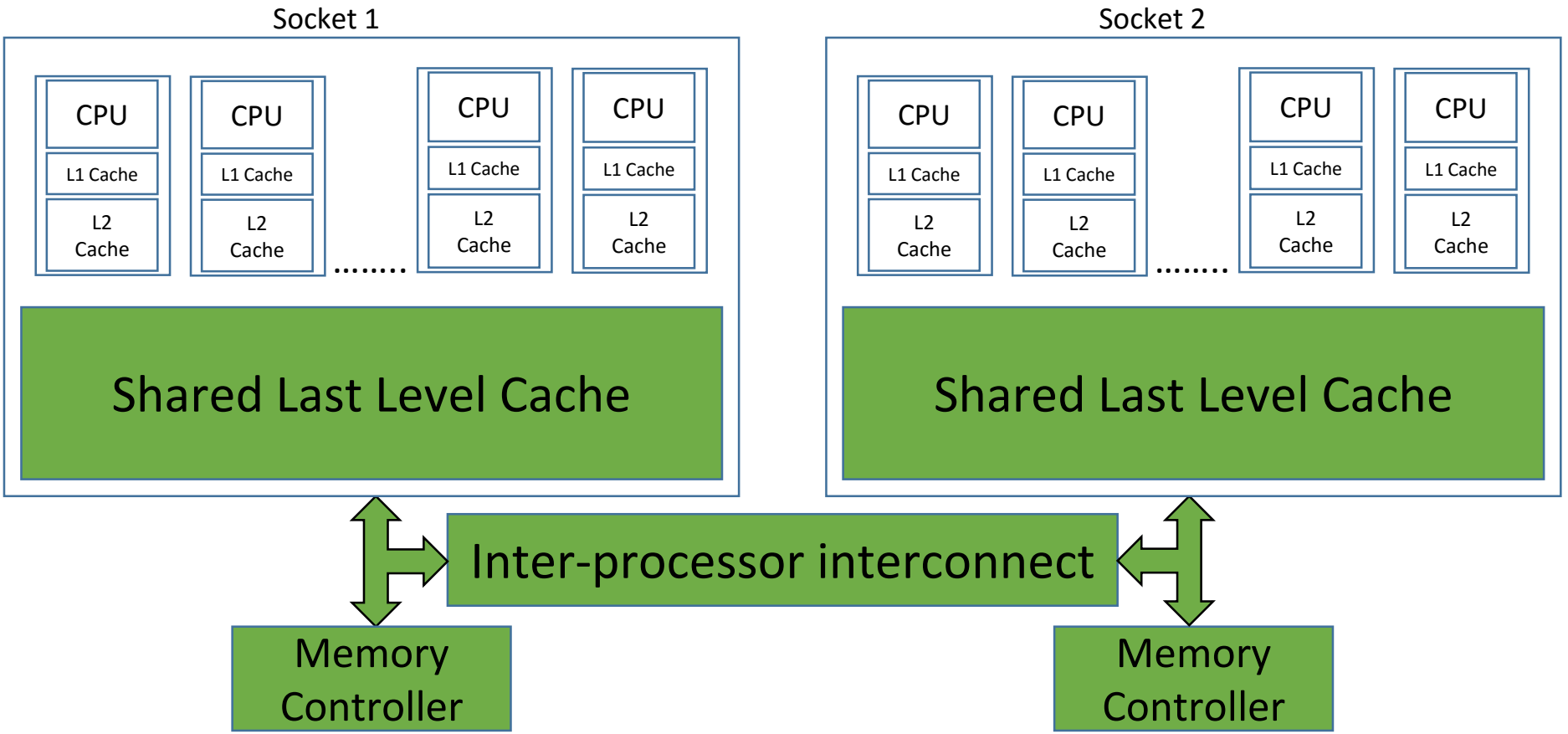
AMD's 16-core, …

Intel's 10-core (20-thread), …

- Resource sharing
  - caches, on- and off-chip interconnects, memory
- Non-uniform access latencies

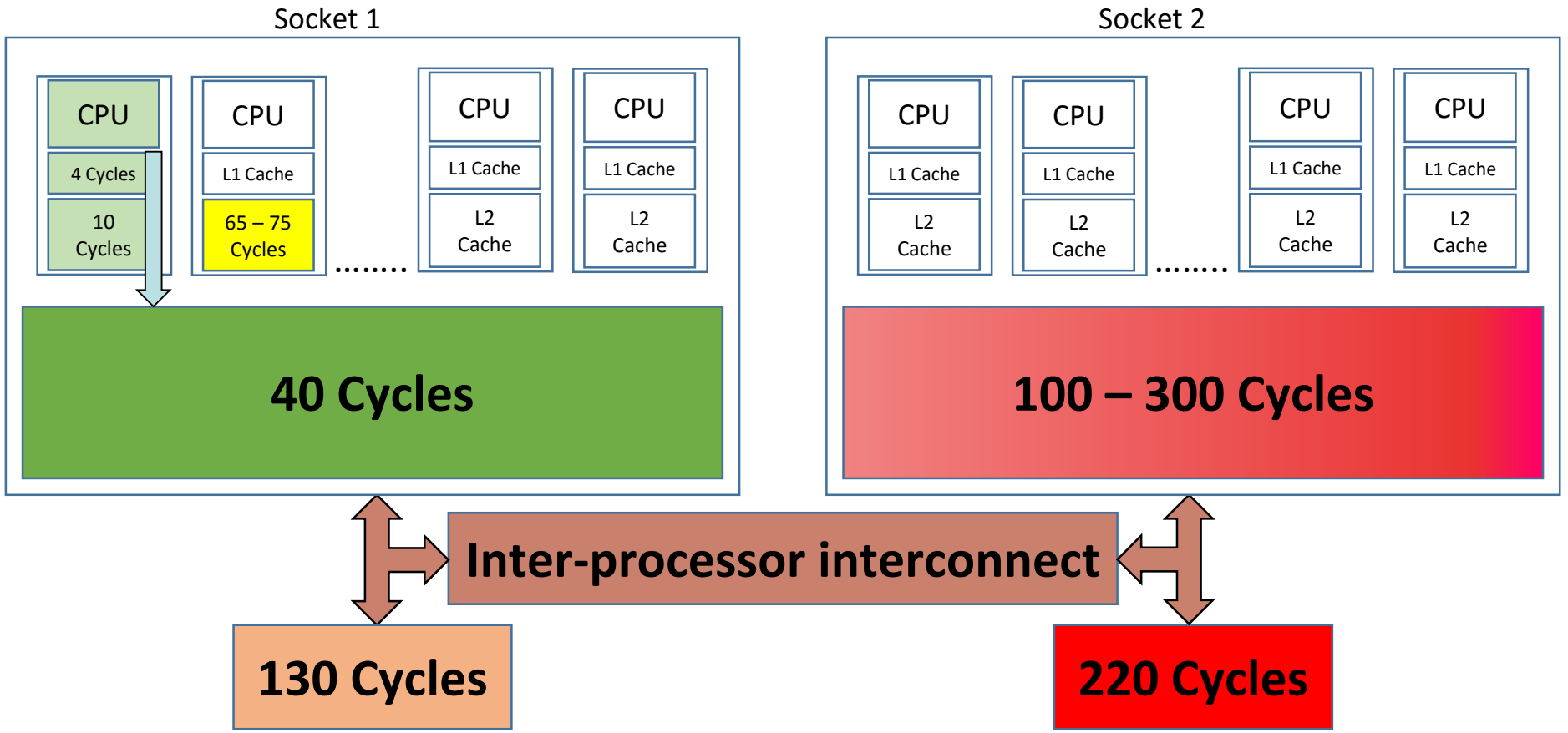
Modern Multicore Systems: Resource Sharing

Problem: Resource contention



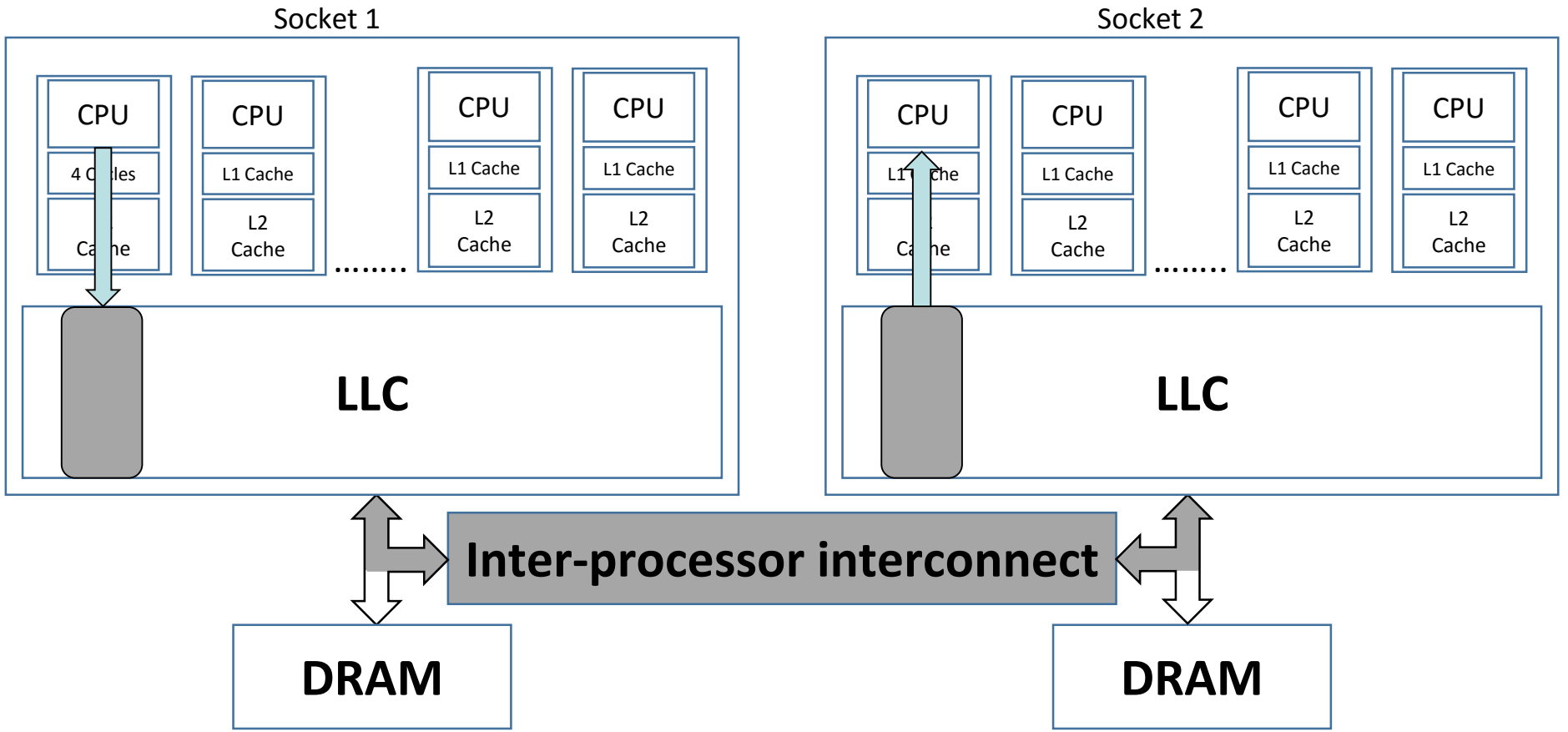
Modern Multicore Systems: Non-Uniform Access Latencies

Problem: Communication costs a function of thread/data placement

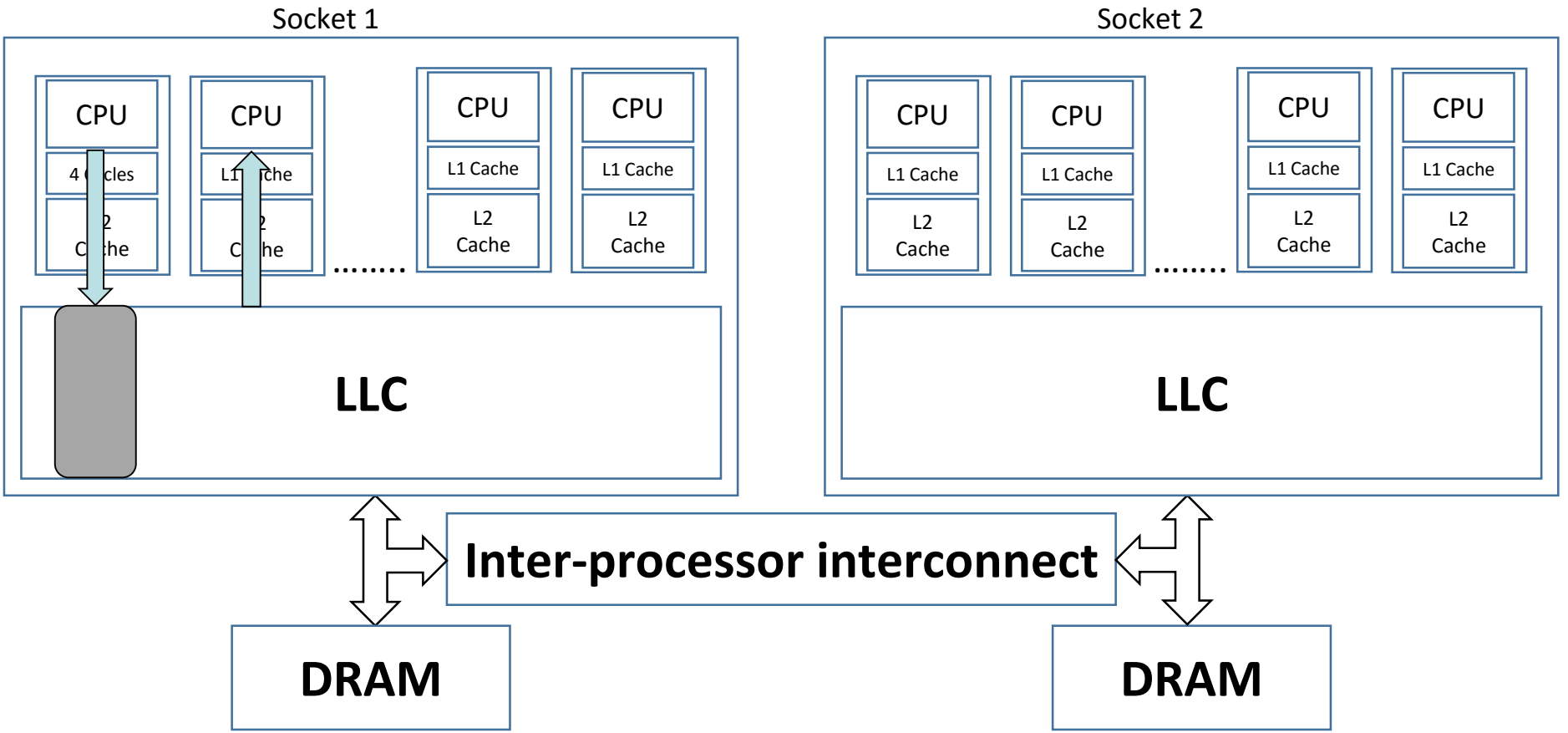


Ref: https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf

Impact of Thread Placement on Data Sharing Costs



Impact of Thread Placement on Data Sharing Costs



Solutions?

- Tam et al. [Eurosys 2007]: address sampling to identify data sharing, costly and specific to some processors
- Tang et al. [ISCA 2011]: performance counters to identify resource contention, does not consider non-uniform communication
- Lepers et al. [USENIX ATC, 2015]: *non-uniform memory bandwidth* utilization, does not consider data sharing

Our approach: Sharing Aware Mapper (SAM):
Separates data sharing from resource contention using performance counters and considers non-uniformity when mapping parallel and multiprogrammed workloads

Sharing-Aware Mapper (SAM)

- Monitoring
- Identifying bottlenecks
- Taking remedial action

Sharing-Aware Mapper (SAM)

- Monitoring: periodically reading hardware performance counters to characterize application behavior
- Identifying bottlenecks
- Taking remedial action

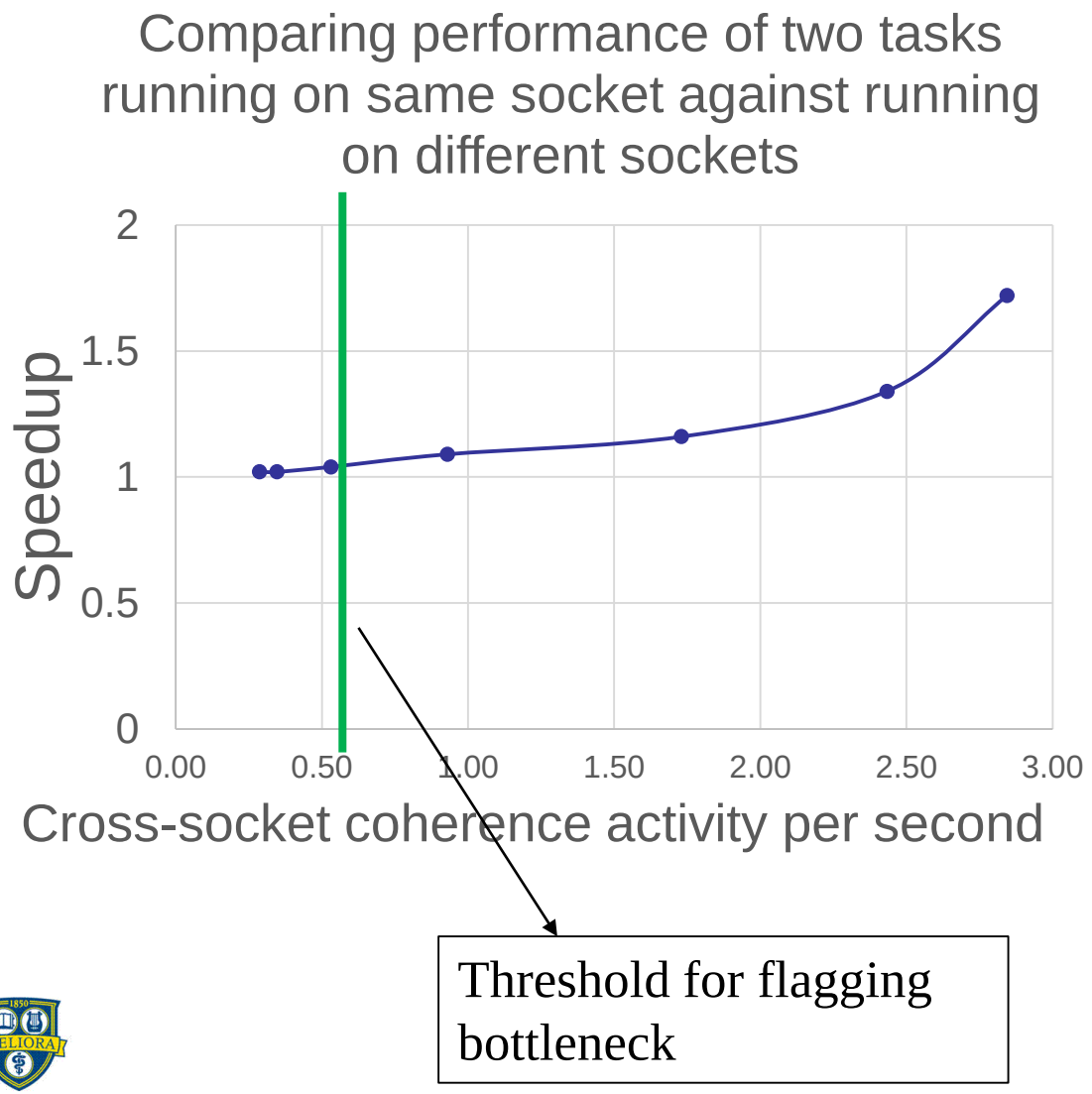
Monitoring Using Performance Counters

- 4 metrics identified: 8 counter events need to be monitored
  - Inter-socket coherence activity
    - Last level cache misses served by remote cache
  - Intra-socket coherence activity
    - Last private level cache misses – (sum of hits and misses in LLC)
  - Local Memory Accesses
    - Approximated by LLC misses
  - Remote Memory Accesses
    - LLC misses served by remote DRAM
- 4 hardware programmable counters available: requires multiplexing

Sharing-Aware Mapper (SAM)

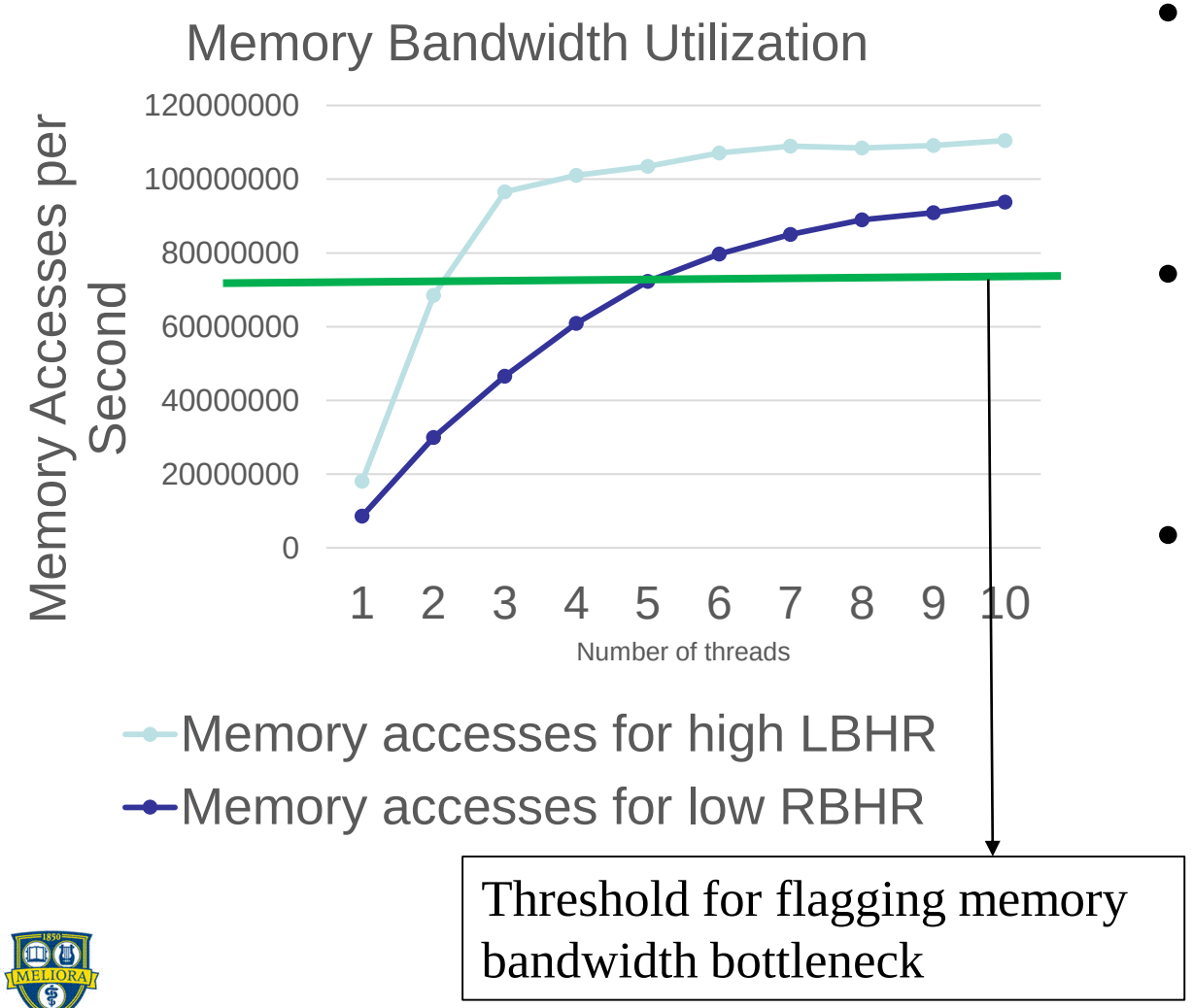
- Monitoring: periodically reading hardware performance counters to characterize application behavior
- Identifying bottlenecks: using pre-characterized thresholds
- Taking remedial action

Identifying Bottlenecks: Coherence Activity Threshold



- Microbenchmark forces data to move from one task to the other, generating coherence activity
- Rate of coherence activity varied by varying ratio of private to shared variable access

Identifying Bottlenecks: Memory Bandwidth Threshold



- Variable number of memory intensive threads run on the same processor
- ~10% difference in available bandwidth at saturation
- Conservative estimate can sufficiently identify bottlenecks

Sharing-Aware Mapper (SAM)

- Monitoring: periodically reading hardware performance counters to characterize application behavior
- Identifying bottlenecks: using pre-characterized thresholds
- Taking remedial action: periodically remapping to manipulate process-processor affinity

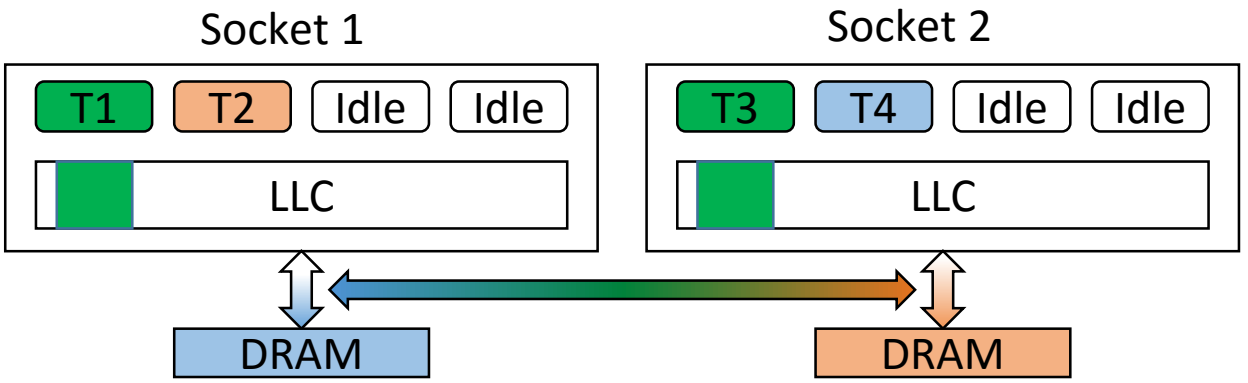
SAM Decision Making

- Prioritize reduction of
  - Inter-socket coherence >
  - Remote memory access >
  - Socket bandwidth contention
- Co-locate threads with inter-socket coherence activity if
  - Idle cores are available >
  - CPU bound tasks can be moved >
  - Memory bound tasks can be moved
  - But not at the expense of separating threads with intra-coherence activity
- Return threads with remote memory accesses to original socket if possible
- Balance memory accesses across sockets
  - Transfer excess memory intensive threads to other sockets
    - Idle processors > CPU bound

SAM Decision Making

- Prioritize reduction of
  - Inter-socket coherence >
  - Remote memory access >
  - Socket bandwidth contention
- Co-locate threads with inter-socket coherence activity if
  - Idle cores are available >
  - CPU bound tasks can be moved >
  - *Memory bound tasks can be moved*
  - But not at the expense of separating threads with intra-coherence activity
- Return threads with remote memory accesses to original socket if possible
- Balance memory accesses across sockets
  - Transfer excess memory intensive threads to other sockets
    - Idle processors > CPU bound

Example: Data Sharing and Resource Contention

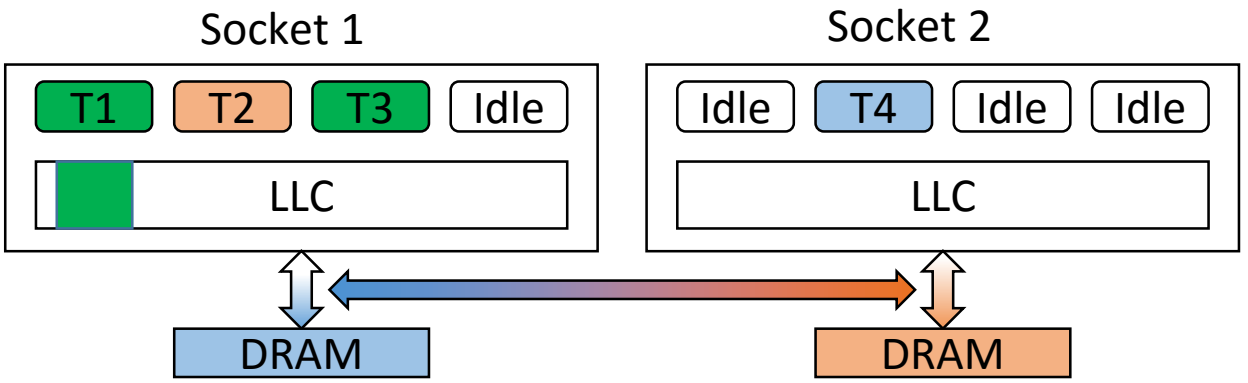


Tasks T1, T3 share data. Tasks T2 and T4 use memory.

- **There are 16 ways to map tasks to cores**
- Linux's default load balancing can result in
  - Inter-socket communication
  - Imbalanced memory bandwidth utilization
  - Remote memory accesses

Example: Data Sharing and Resource Contention

Step 1: Reduce inter-socket coherence

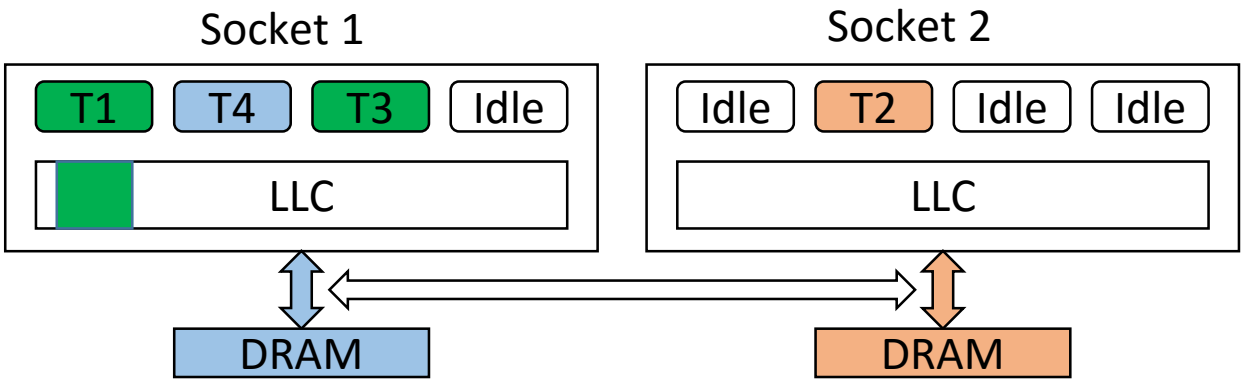


Tasks T1, T3 share data. Tasks T2 and T4 use memory.

- Co-locate T1 and T3 to reduce cross-socket coherence

Example: Data Sharing and Resource Contention

Step 2: Reduce remote memory accesses

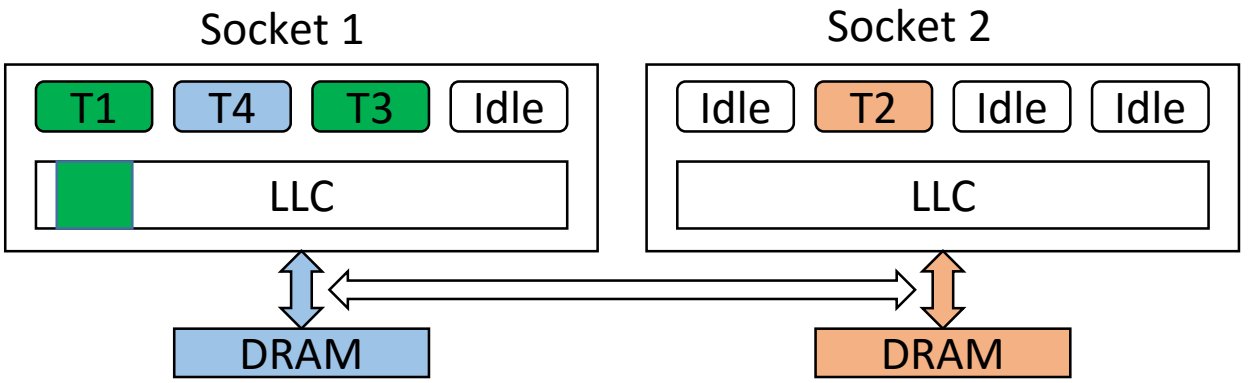


Tasks T1, T3 share data. Tasks T2 and T4 use memory.

- Swap T2 and T4 to eliminate remote memory accesses

Example: Data Sharing and Resource Contention

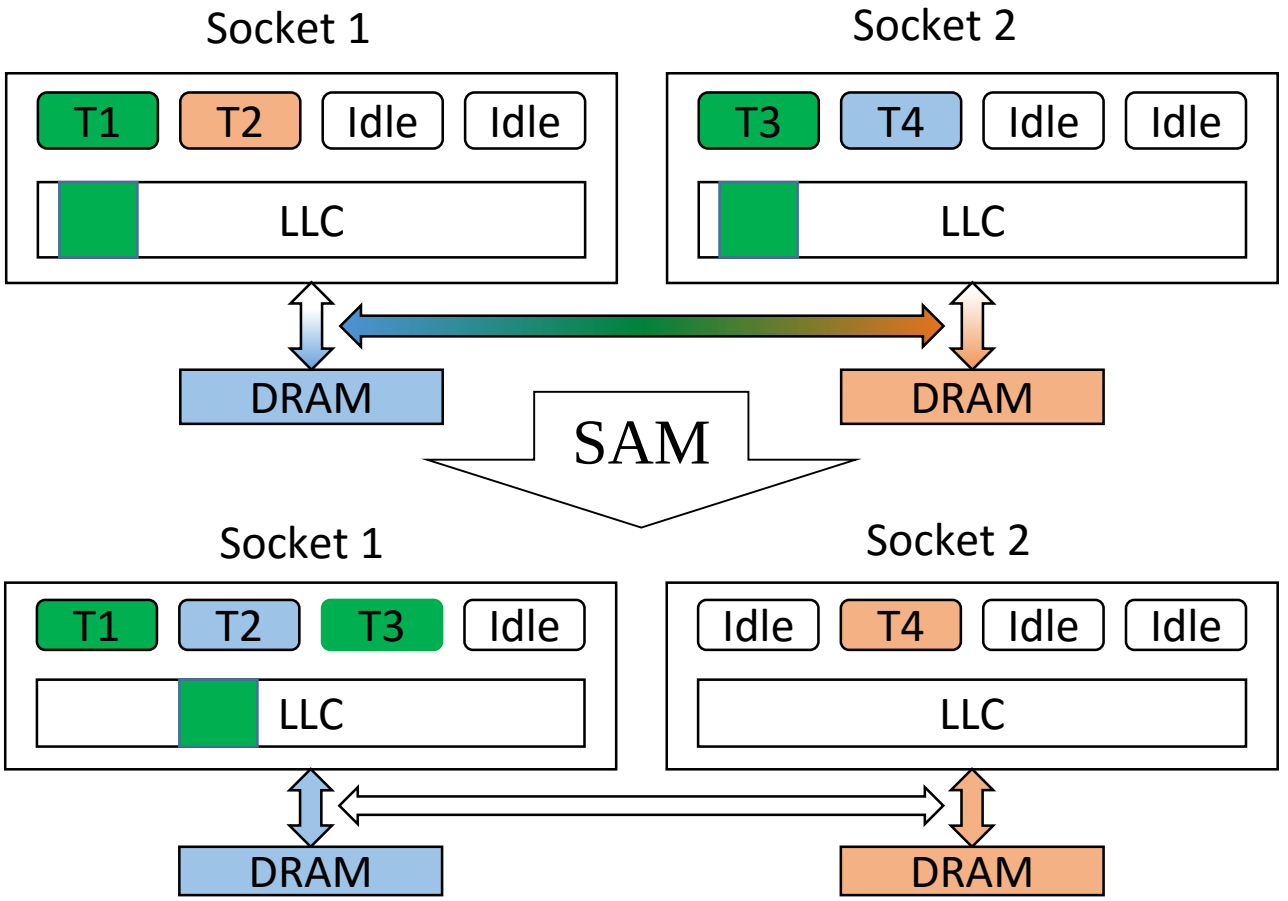
Step 3: Balance memory bandwidth utilization



Tasks T1, T3 share data. Tasks T2 and T4 use memory.

- Memory bandwidth is already balanced so no additional action is performed

Example: Data Sharing and Resource Contention



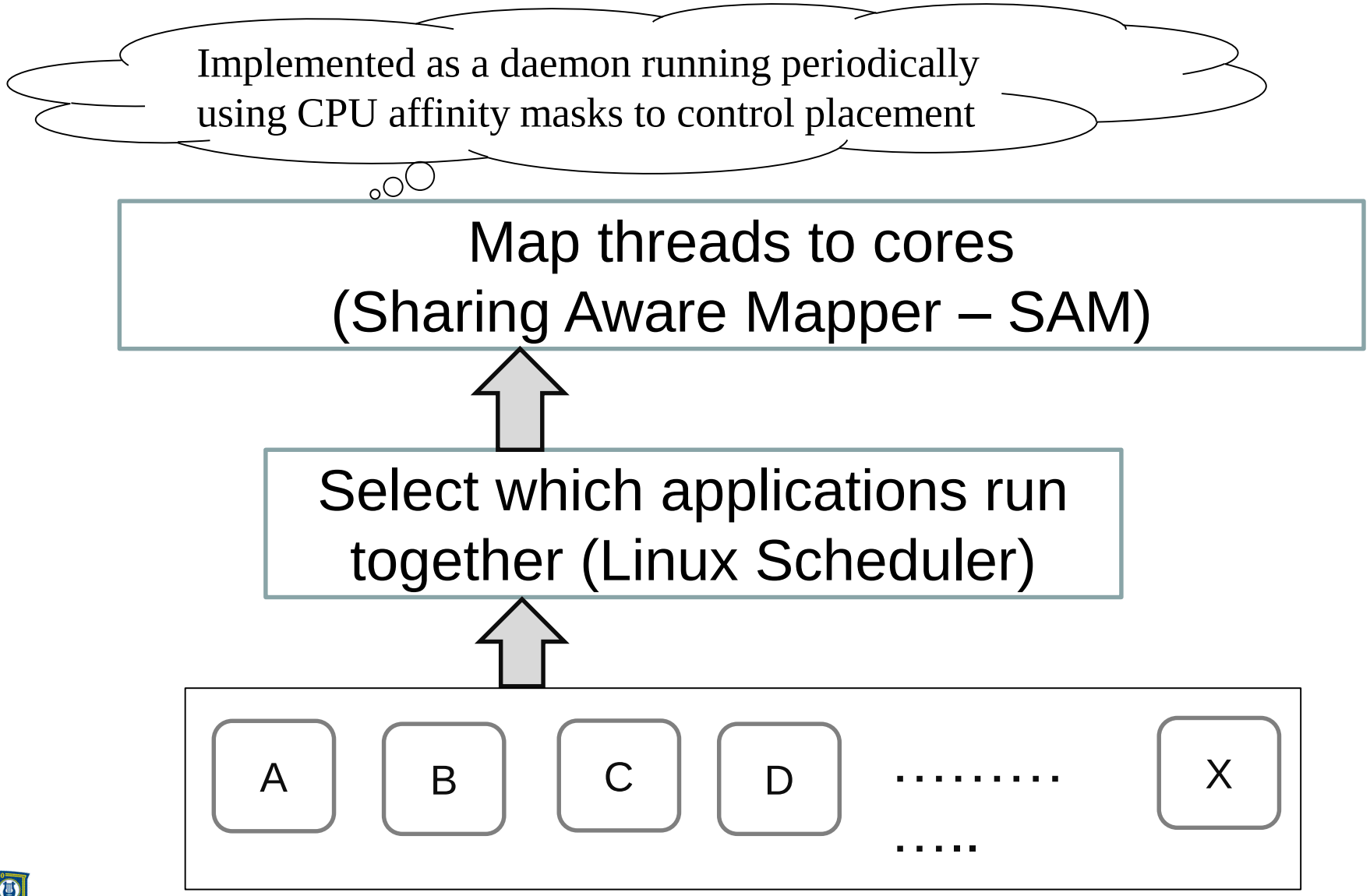
Optimal Solution

- SAM identifies task characteristics with performance counters
- Minimizes inter-processor communication, remote memory accesses while maximizing memory bandwidth utilization

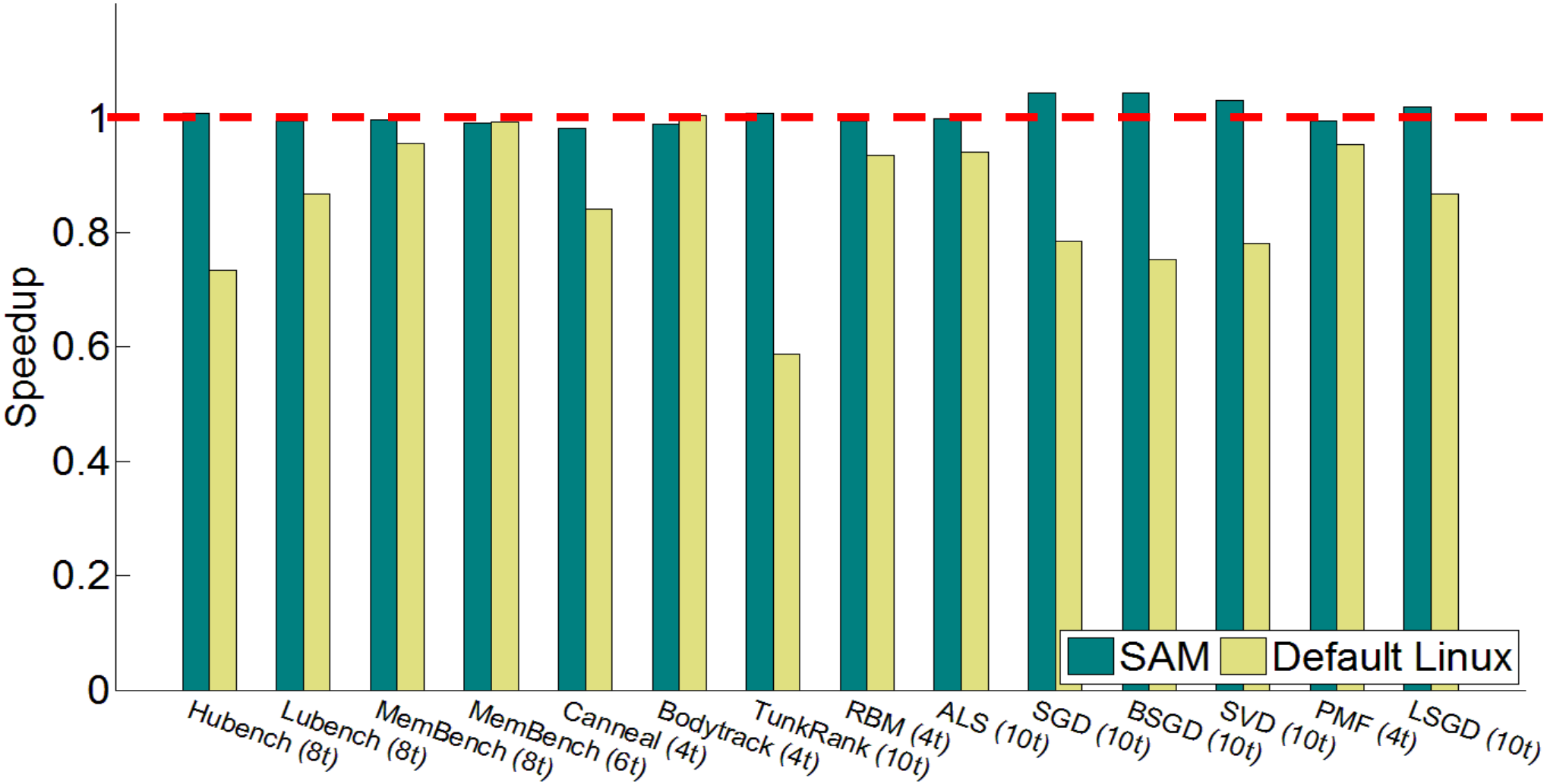
Experimental Environment

- Fedora 19, Linux 3.14.8
- Dual socket machine – Intel Xeon E5-2660 v2 “IvyBridge” processor (10 physical cores, 2 contexts each, 2.20 GHz, 25MB of L3 cache)
- Benchmarks
  - Microbenchmarks (Used for thresholding)
  - SPECCPU ’06 (CPU & Memory bound workloads)
  - PARSEC 3.0 (Parallel workloads – light on data sharing)
  - ALS, Stochastic Gradient Descent, Single Value Decomposition and other parallel workloads (Parallel workloads – stronger data sharing, dynamic behavior, and emerging workloads)

Implementation Context



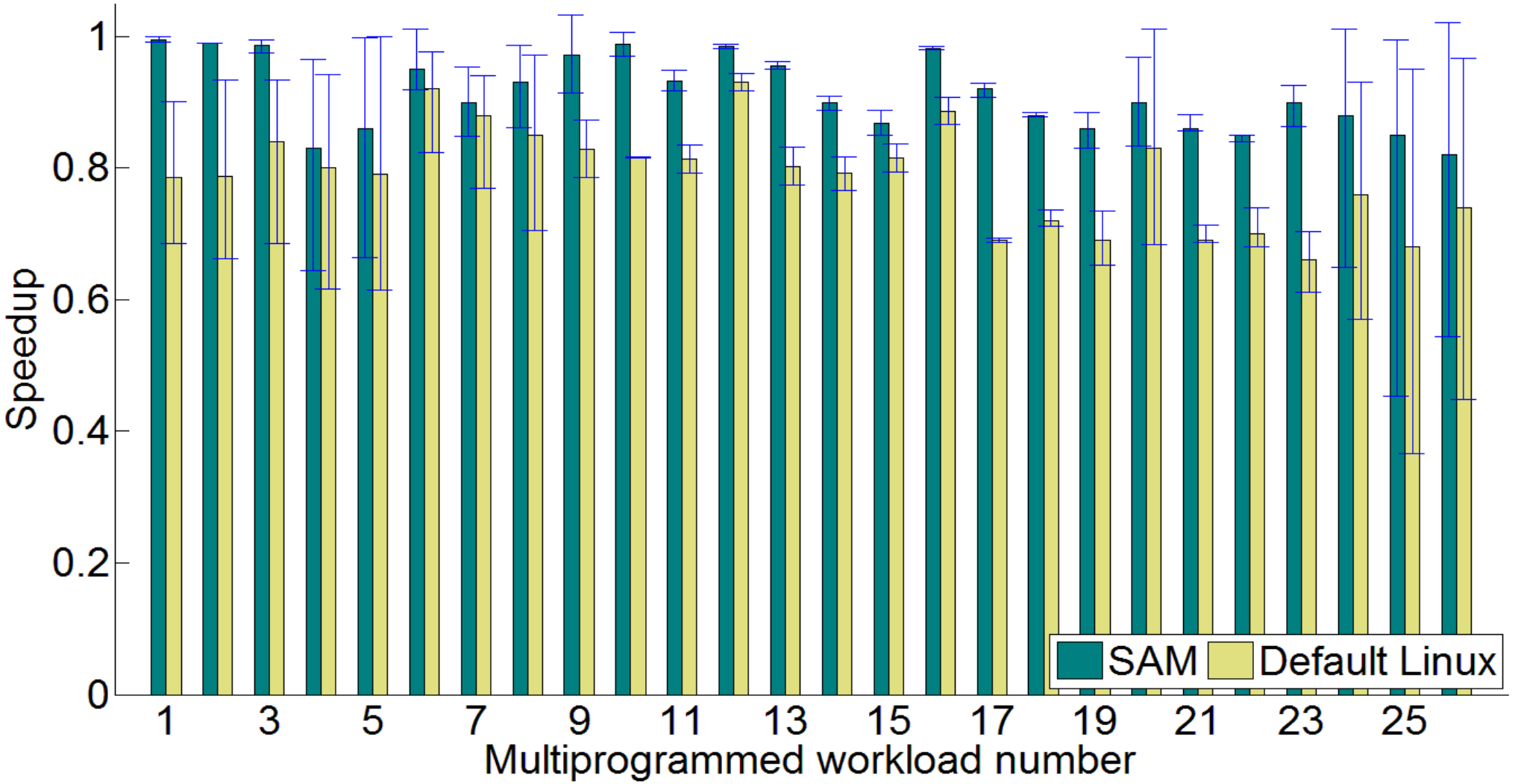
Standalone Applications



Baseline (Normalization factor): Best static mapping determined by exhaustive search

Speedup relative to Linux: Mean = 1.21, Min = 0.99, Max = 1.72

Multiple Applications

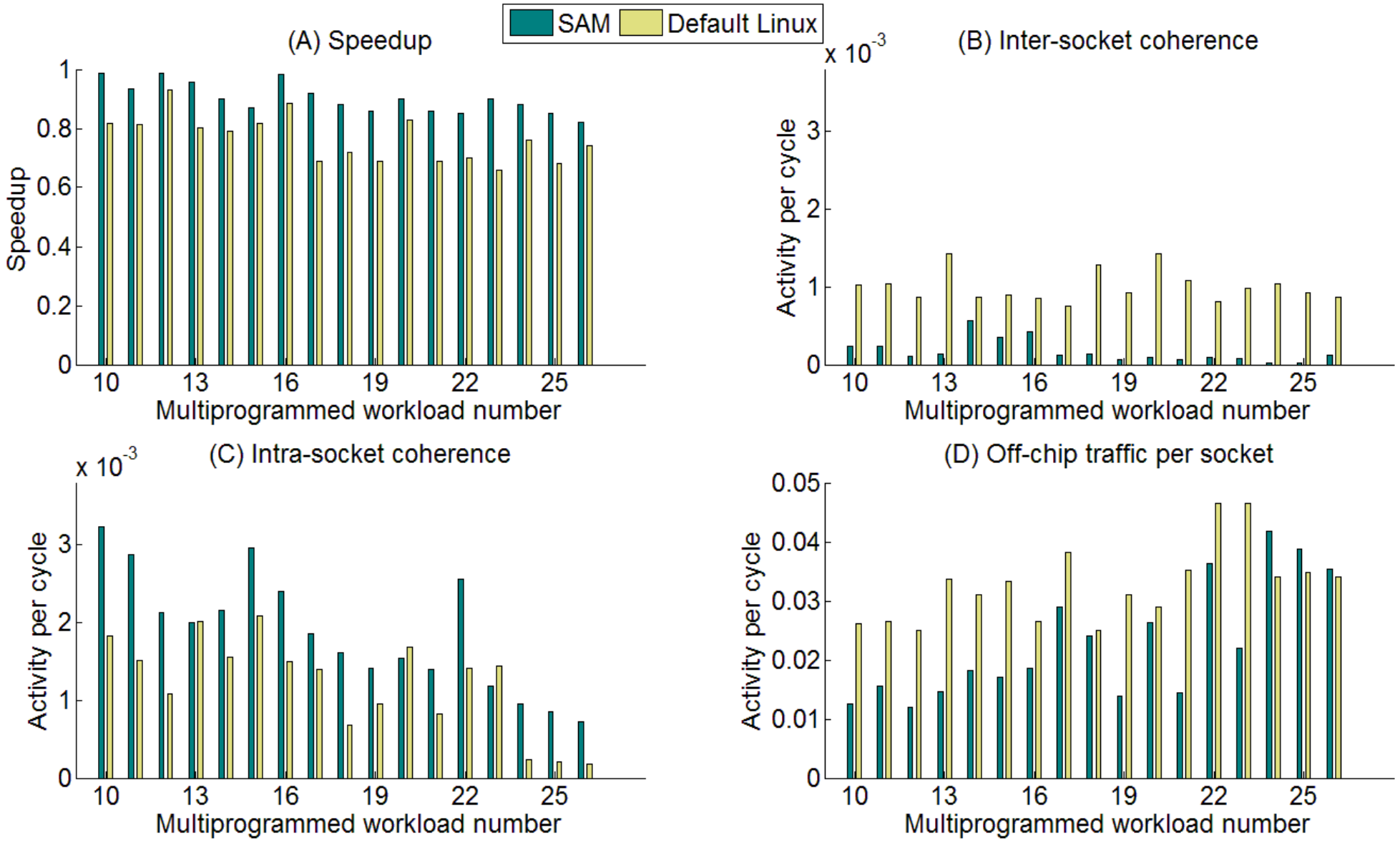


Speedup relative to Linux: Mean = 1.16, Min = 1.02, Max = 1.36

SAM improves fairness:

Average speedup spread per workload: SAM – 0.12, Linux – 0.18

Detailed Analysis



Overheads and Scaling

- SAM's overhead <1% (40 cores)
  - Performance counter reading
    - Invoked every tick – 8.9 µs per tick = 8.9 msec per second
    - Constant time overhead
  - Data consolidation and bottleneck identification
    - Invoked every 100ms – 9.9 µs per call = 99 µs per second
    - Scales linearly with number processing cores
  - Decision making
    - Invoked every 100ms – negligible time related to data consolidation
    - $O(n^2)$ complexity, but for $n \leq 40$, time spent is within measurement error
- SAM's data consolidation and decision making is centralized

Conclusions

- Performance counters provide sufficient information to identify and separate data sharing from resource contention
- SAM improves performance and reduces performance disparities across applications by:
  - Minimizing cross-socket coherence
  - Minimizing remote memory accesses
  - Maximizing memory bandwidth utilization
- Future work to improve data sharing aware scheduling
  - Benefits of hyper-threading
  - Distributed decision making for better scalability

Thank you

Questions?

Data Sharing or Resource Contention: Toward Performance Transparency on Multicore Systems

Sharanyan Srikanthan
Sandhya Dwarkadas
Kai Shen

Department of Computer Science
University of Rochester