Bringing the Multicore Revolution to Safety-Critical Cyber-Physical Systems

[1]University of North Carolina Chapel Hill   [2]North Carolina State University
PIs: Dr. James Anderson[1] & Dr. Frank Mueller[2]
Students: Nathan Otterness[1], Micaiah Chisholm[1], Namhoon Kim[1], Xing Pan[2]

Motivation

➤ Shared hardware like caches & memory introduce timing unpredictability for real-time systems (RTS).

➤ Worst-case execution time (WCET) analysis for RTS with shared hardware resources is often so pessimistic that the extra processing capacity of multicore systems is negated.

➤ Different levels of assurance are required for different criticality tasks.

Problem
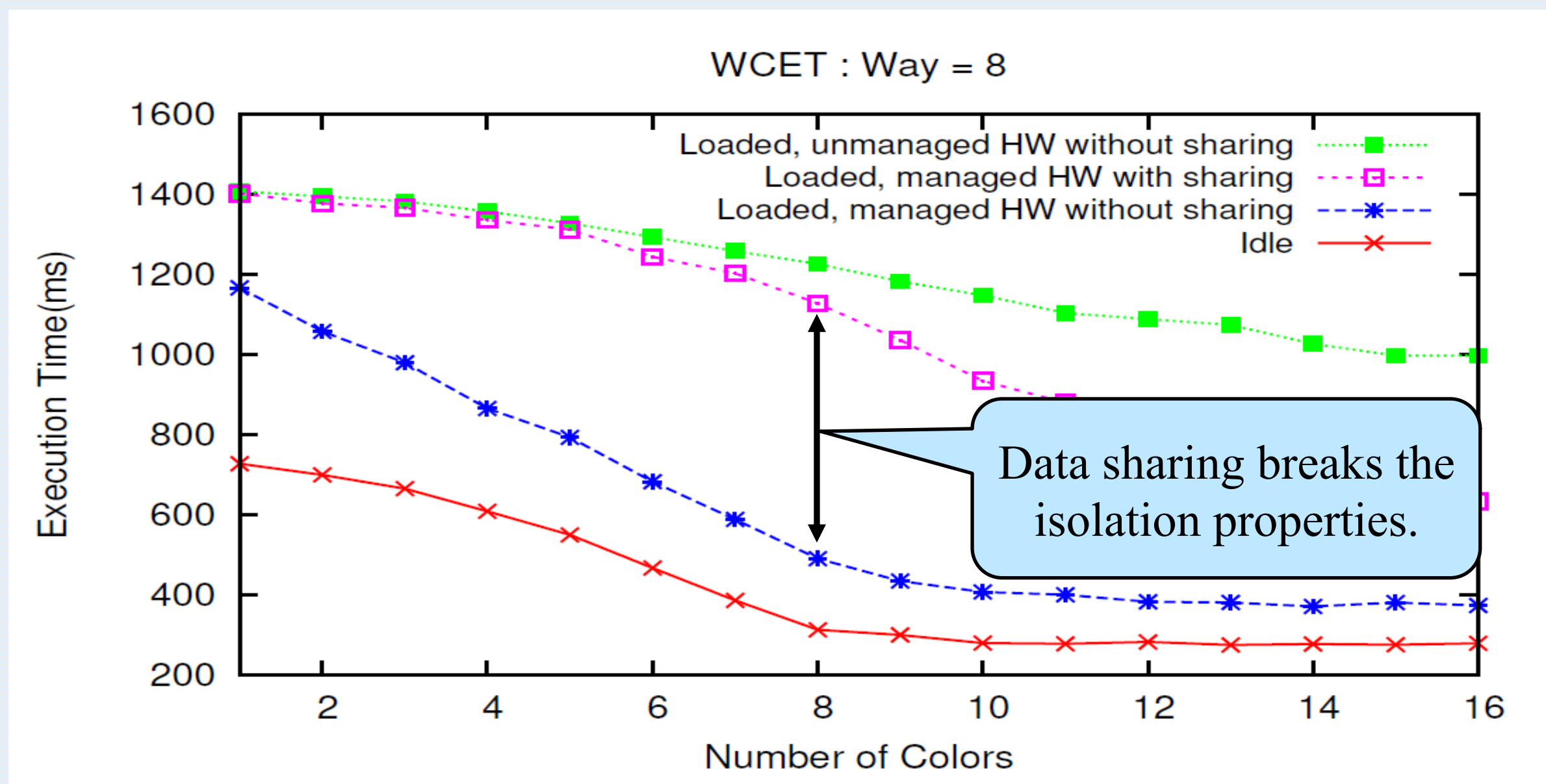
➤ Recent work has shown that, by combining hardware management and criticality-aware task provisioning, capacity loss can be significantly reduced when supporting real-time workloads on multicore platforms.

➤ Supporting real-world workloads has not been realized due to a lack of support for sharing among tasks.
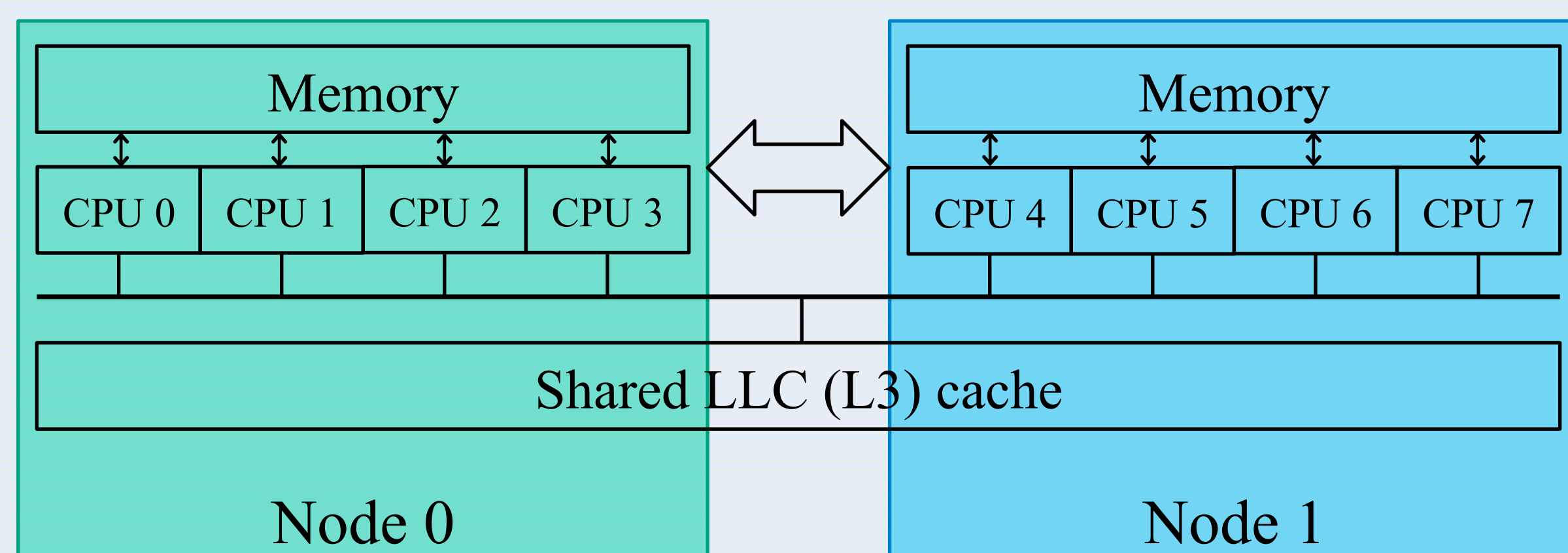

➤Shared memory shows timing unpredictability. (1) The latency of accessing remote node is significantly longer than local node. (2) conflicts between shared-bank accesses result in unpredictable memory-access latencies.

Solution

➤ **Controller-Aware Memory Coloring:**
• Assigned colors to the entire memory space of heap, static, stack, and instruction segments with locality affinity for controller and bank-awareness.
• Avoided memory accesses to remote node.
• Reduced conflicts among banks.

➤ **We considered two types of sharing among tasks: shared buffers and shared libraries.**

➤ **Supporting Data Sharing in Mixed-Criticality, Multicore Systems:**
• Implemented two inter-process communication (IPC) mechanisms.
  ▪ Producer/consumer buffers (PCBs) and wait-free buffers (WFBs)
• Proposed three techniques to mitigate interference due to shared memory.
  ▪ <u>S</u>elective LLC <u>ByP</u>ass (SBP): Designate a buffer as uncacheable and allocate it from the Level-C banks.
  ▪ <u>C</u>oncurrency <u>E</u>limination (CE): If a buffer is shared by two tasks at Levels A and/or B, assign both tasks to the same core and allocate the buffer from that core's bank.
  ▪ <u>LLC</u> Locking (CL): Permanently lock a buffer in the LLC.

➤ **Allowing Shared Libraries while Supporting Hardware Isolation:**
• Introduced per-partition library replicas.
• Implemented a system call to replicate shared libraries.

Supporting Data Sharing in Mixed-Criticality – Solutions & Results

LLC Allocation for locked buffers
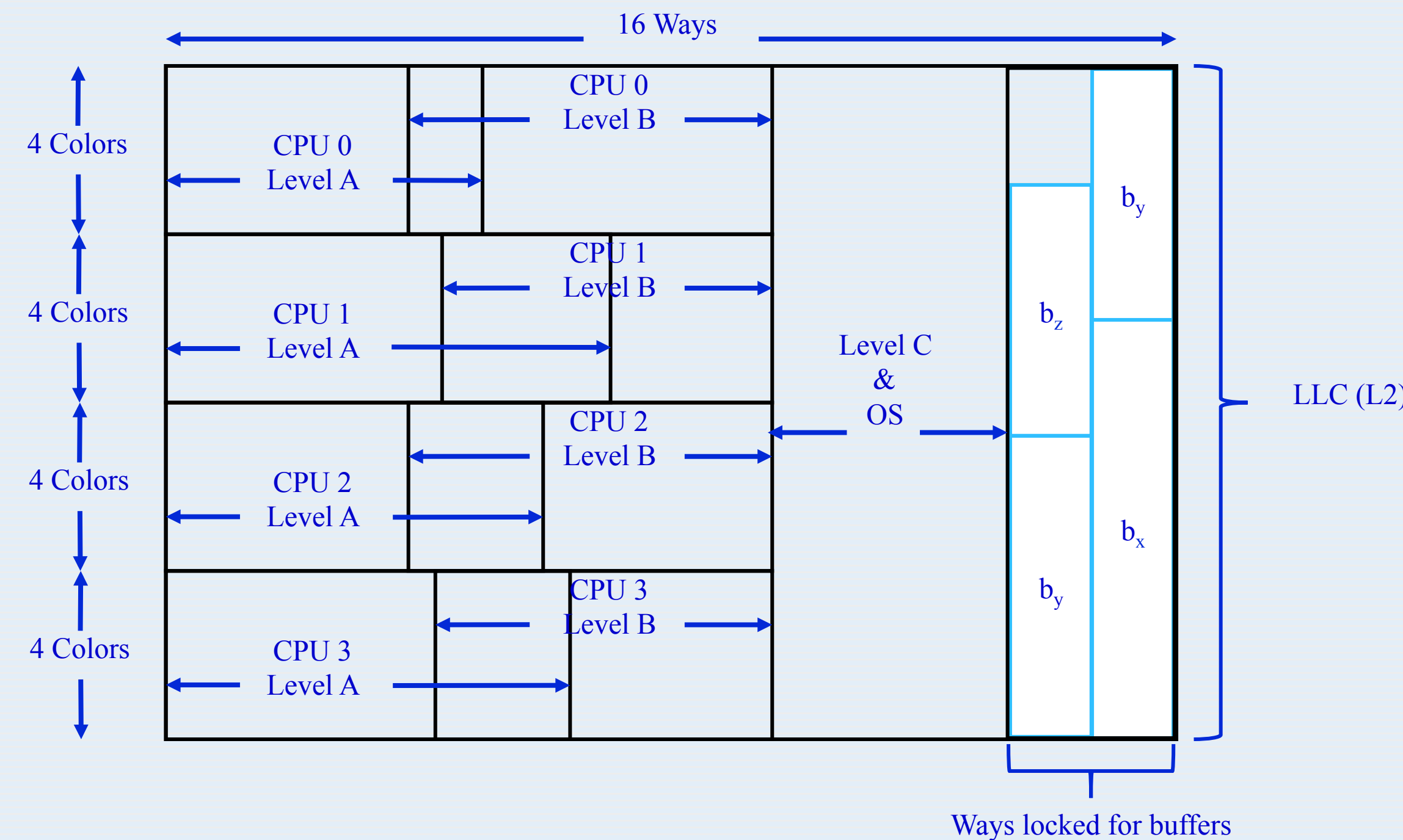
• PCBs are used for tasks of the same criticality.
• WFBs are used for tasks of different criticalities.
• SBP eliminates unpredictable LLC interference at Levels A and B.
• CE eliminates concurrent interference.
• CL eliminates any DRAM-bank contention but reduces the LLC size for caching code and local data.
• Partitioning heuristics are proposed to support CE.
• A modified criticality-aware optimization technique based on linear programming is used for applying CL.

Results
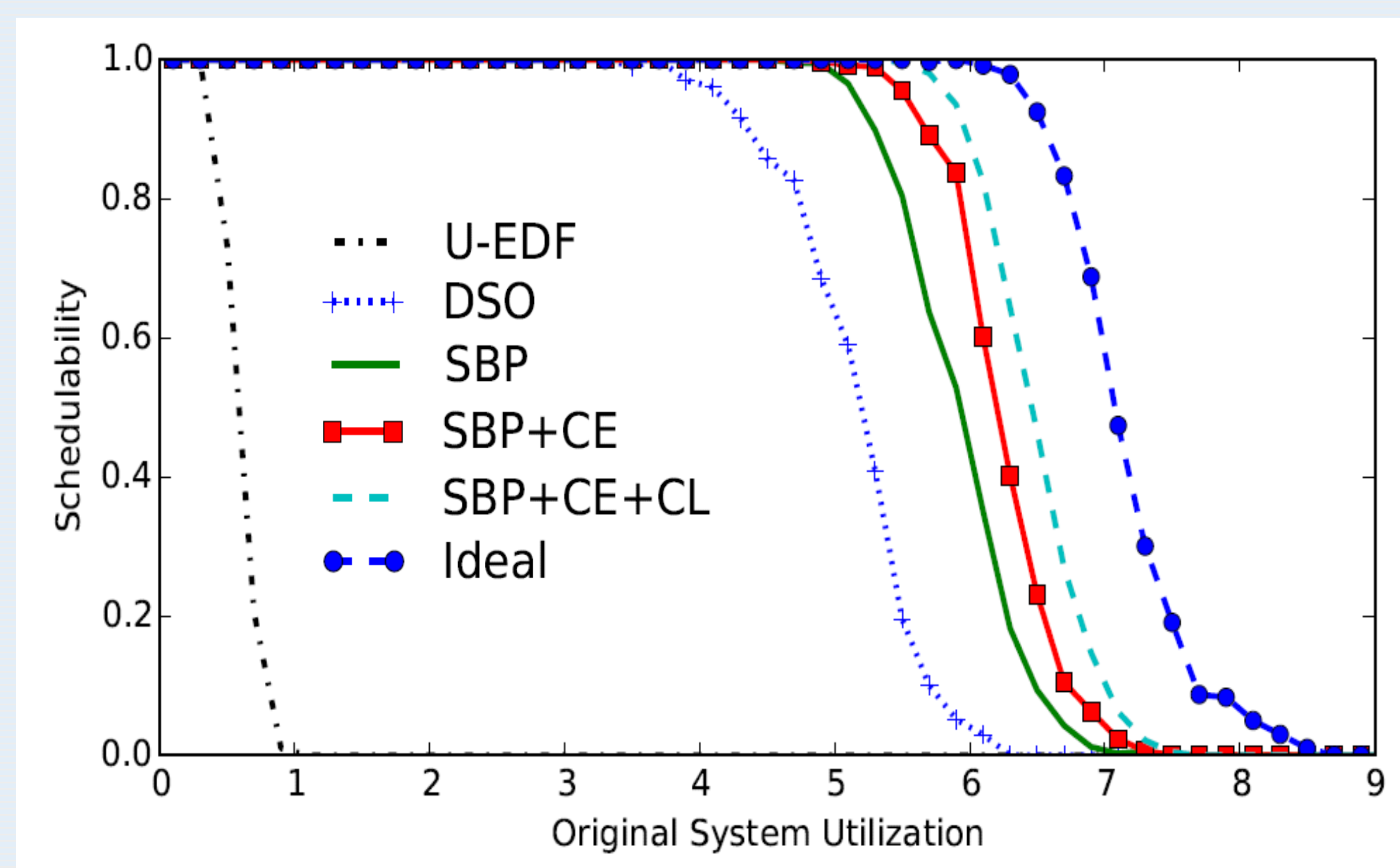
• We conducted micro-benchmarks and a large-scale overhead-aware schedulability study.
• CL writing times were 2 to 7% of SBP writing times.
• CE writing times were 50 to 60% of SBP writing times.
• SBP provided schedulability benefits in 60% of scenarios. In 30% of cases, SBP provided schedulability near to Ideal.
• CE and CL provided mild improvement to schedulability in 20% of considered scenarios.

Allowing Shared Libraries while Supporting Isolation – Solutions & Results

Design and Implementation

• Per-partition replicas can allow shared libraries to be used while preserving isolation properties.
• The first time a shared library is used by a task, a set of replicas is created by allocating new pages from the appropriate DRAM bank.
• Library content is copied into them.
• The page table entries are modified to reference the replica pages instead of the original ones.
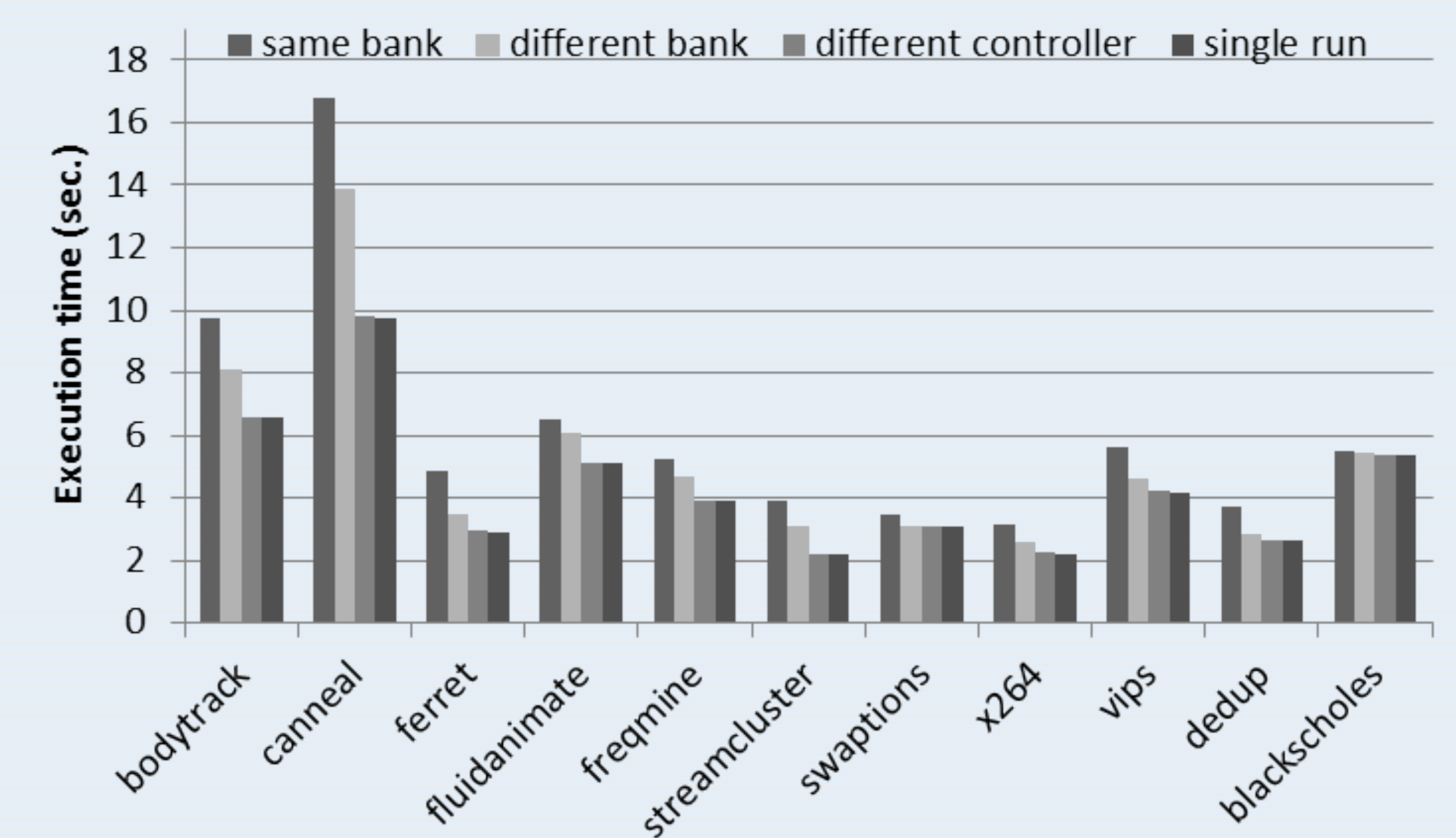
Controller-Aware Memory Coloring – Solutions & Results

Design and Implementation
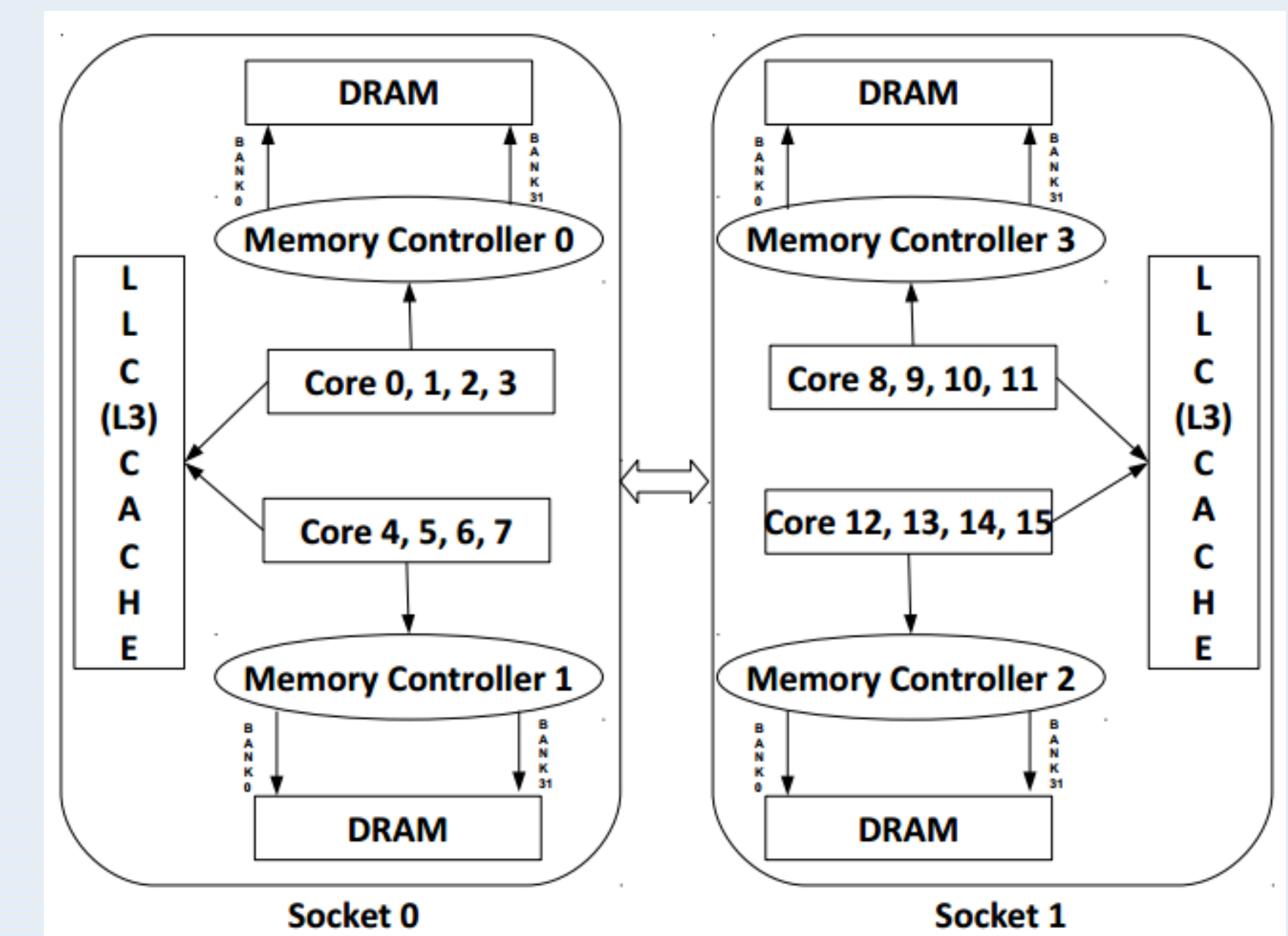
• Partition the entire memory space into multiple "colors" based on memory architecture.
• Assign multiple private memory colors to each real-time task based on coloring configuration automatically.
• Configure memory coloring policy for entire memory space (heap, static, stack, and instruction segments).
• Analyze the overhead of controller-aware memory coloring for real-time task's WCET.
• Require no change for applications.

Results

• System performance for Parsec code is enhanced by our controller-aware memory coloring scheme since it can avoid remote access penalty and reduce shared bank conflict.
• The "different_controller" of our approach is a policy that provides single core equivalence.

Conclusions

➤ Designed controller-aware memory coloring techniques to support memory coloring allocations for entire memory space.

➤ Avoided remote memory node access, and reduced bank-level contention.

➤ Evaluated on Intel and AMD 16 core platforms and I.MX6 quad-core platform.

➤ Implemented two IPC mechanisms based on shared memory.

➤ Designed three techniques to reduce interference due to sharing data among tasks.

➤ Proposed partitioning heuristics and a criticality-aware optimization technique for allocating shared buffers.

➤ Proposed per-partition replicas of shared libraries to allow sharing libraries.

➤ Conducted micro-benchmarks and large-scale overhead-aware schedulability studies.