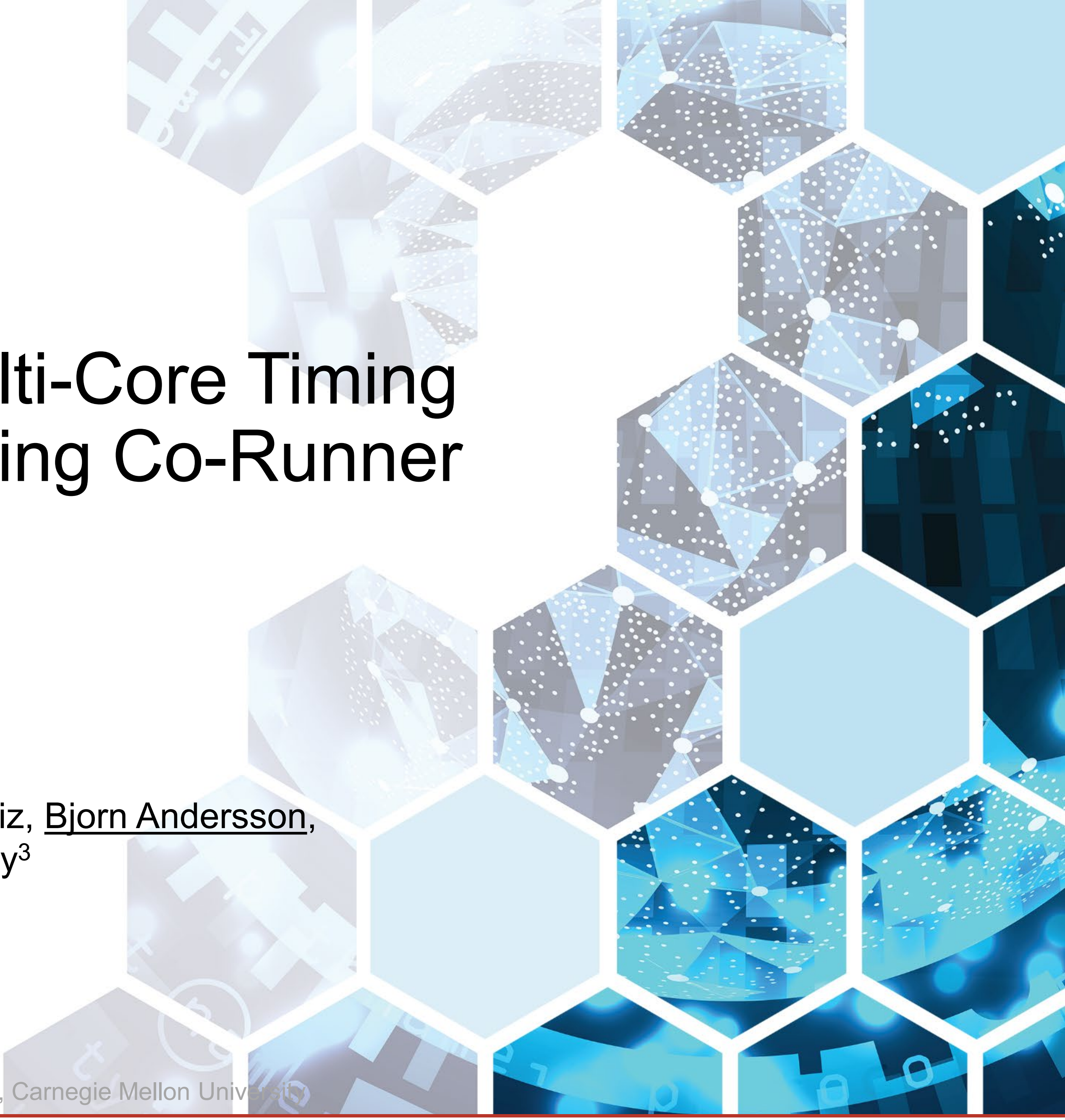


Addressing Multi-Core Timing Interference using Co-Runner Locking

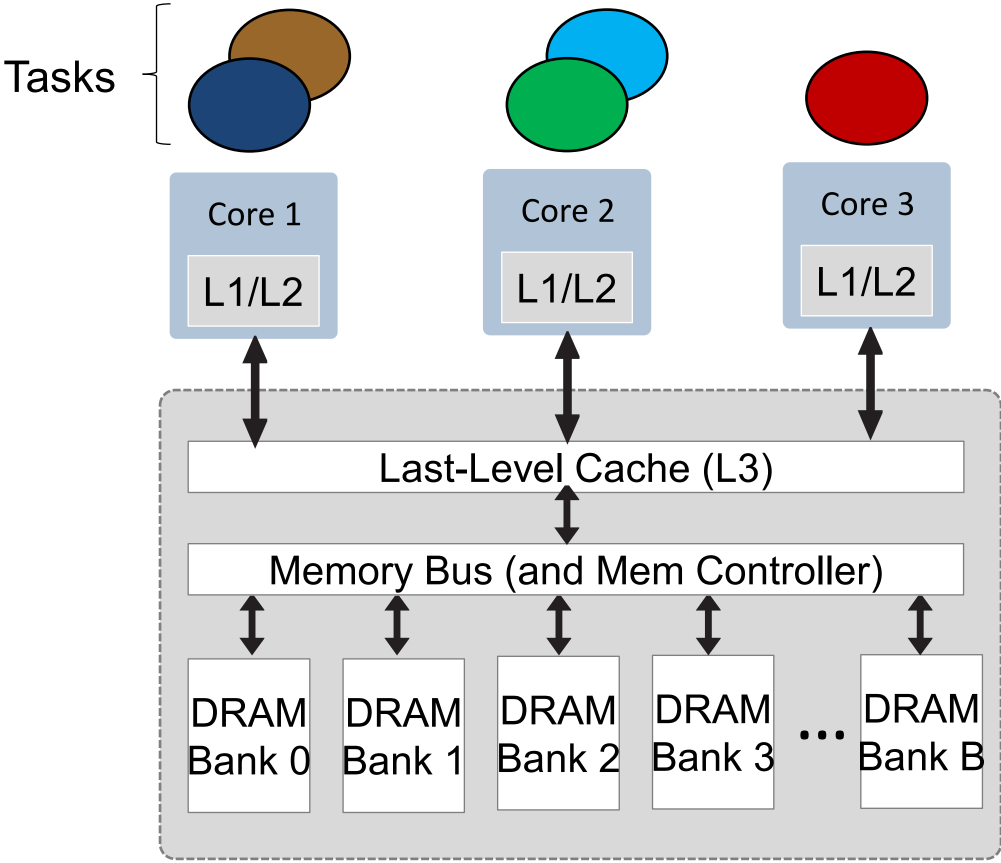
Hyoseung Kim[2], Dionisio de Niz, Bjorn Andersson,
Mark Klein, and John Lehoczky[3]

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

[2] University of California, Riverside
[3] Department of Statistics and Data Science, Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

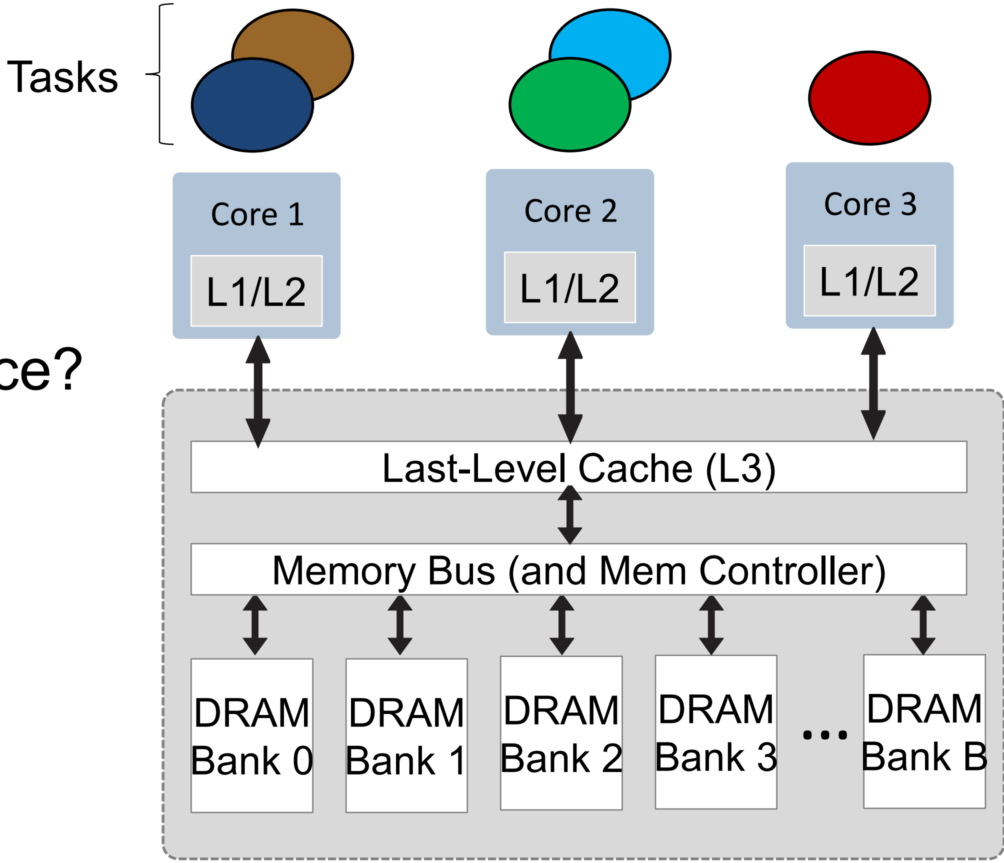
DM21-1040


Tasks
Core 1
L1/L2
Core 2
L1/L2
Core 3
L1/L2
Last-Level Cache (L3)
Memory Bus (and Mem Controller)
DRAM Bank 0
DRAM Bank 1
DRAM Bank 2
DRAM Bank 3
…
DRAM Bank B

How to

manage inter-core interference?

prove timing correctness
considering inter-core interference?

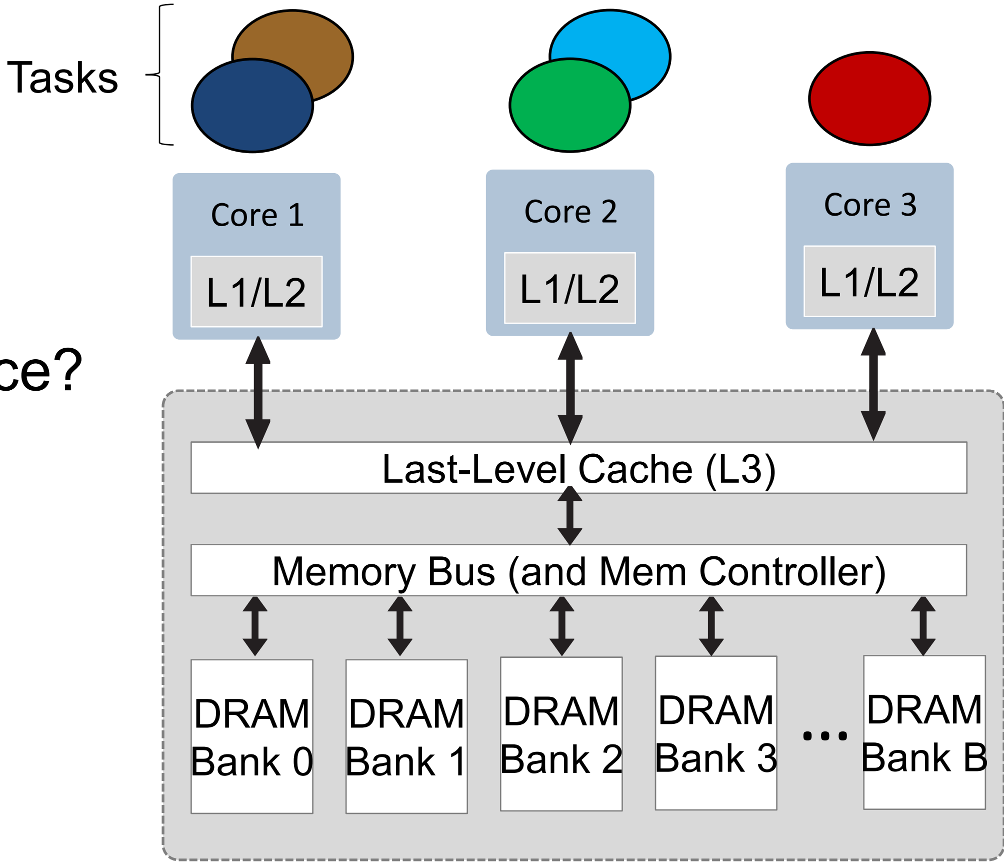


How to

manage inter-core interference?

prove timing correctness considering inter-core interference?

considering that resources are complex and often undocumented

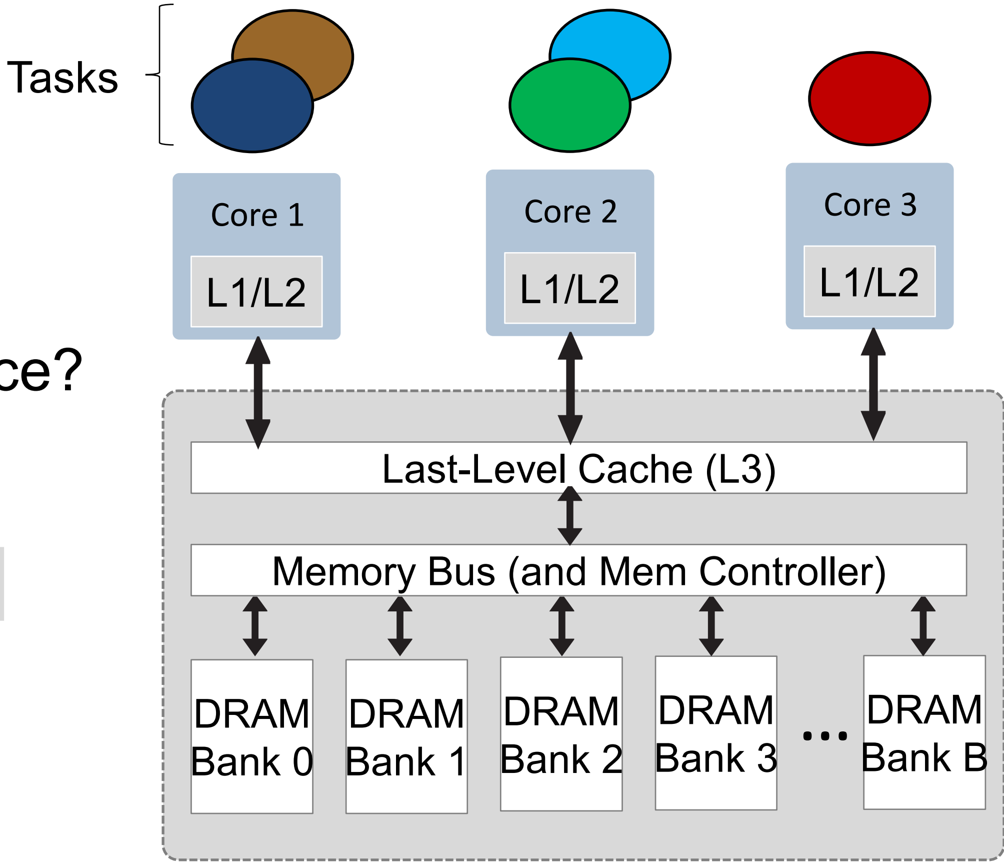


How to

manage inter-core interference?

prove timing correctness considering inter-core interference?

considering that resources are complex and often undocumented



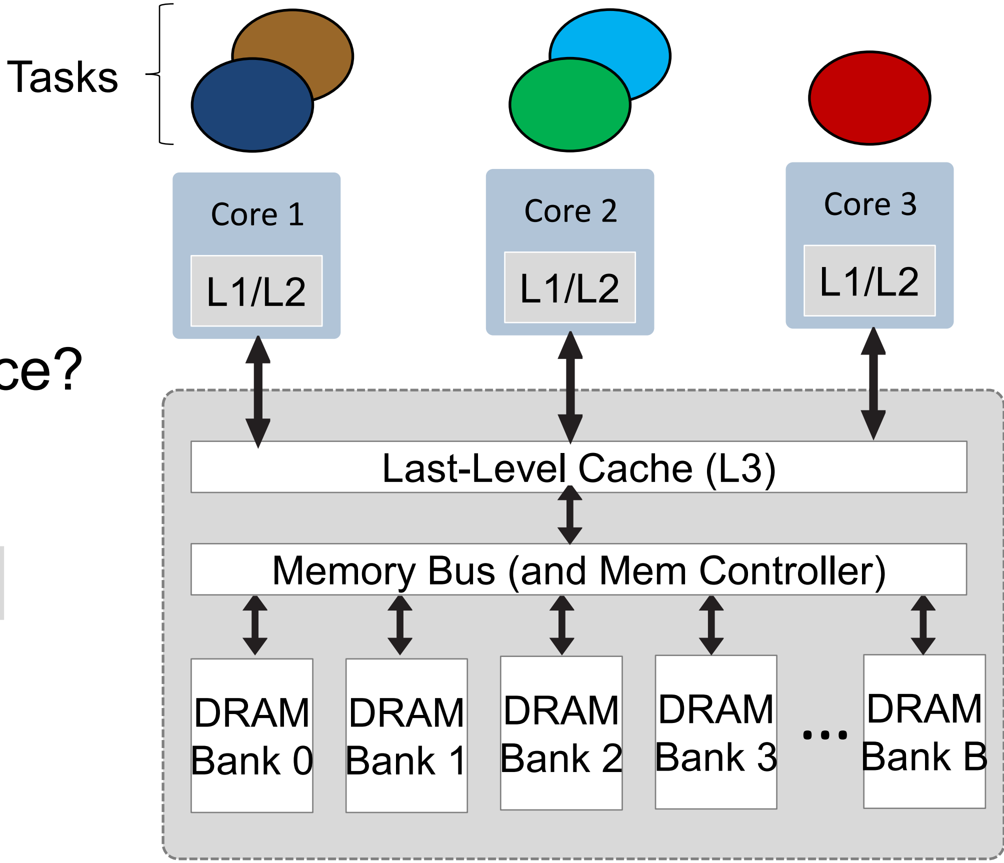
How to

manage inter-core interference?

prove timing correctness considering inter-core interference?

considering that resources are complex and often undocumented

Use an abstraction the describes the effect of shared hardware resources on timing.

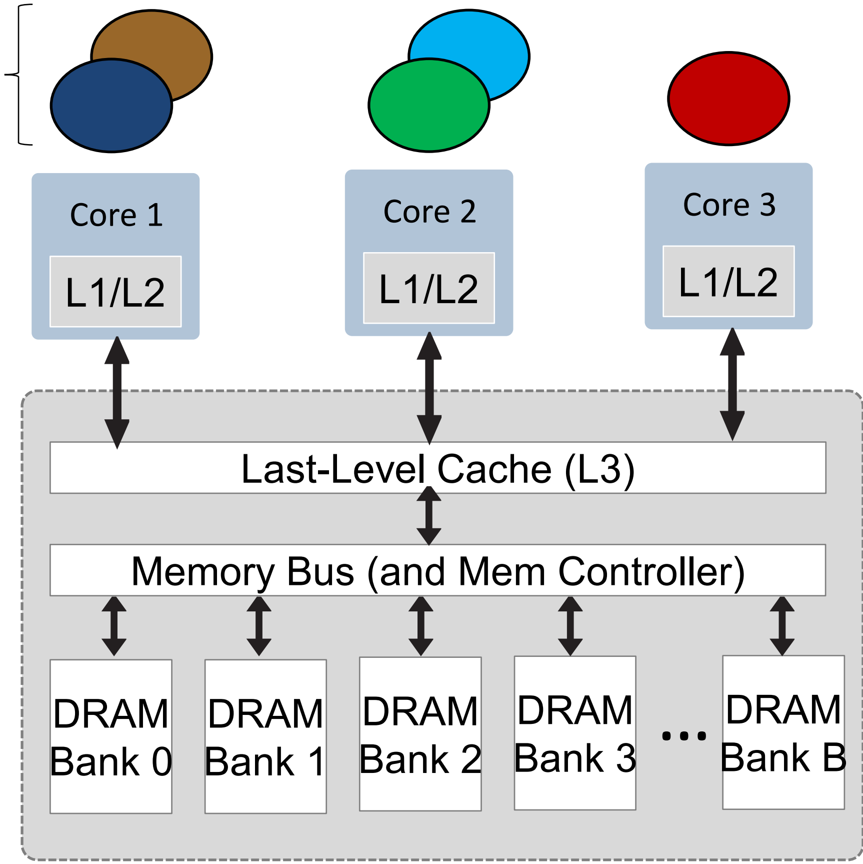


How to
manage inter-core interference?

prove timing correctness
considering inter-core interference?

considering that resources are
complex and often undocumented

Use a schedulability test that can take
this abstraction as input


Tasks

Core 1
L1/L2

Core 2
L1/L2

Core 3
L1/L2

Last-Level Cache (L3)

Memory Bus (and Mem Controller)

DRAM Bank 0
DRAM Bank 1
DRAM Bank 2
DRAM Bank 3
…
DRAM Bank B

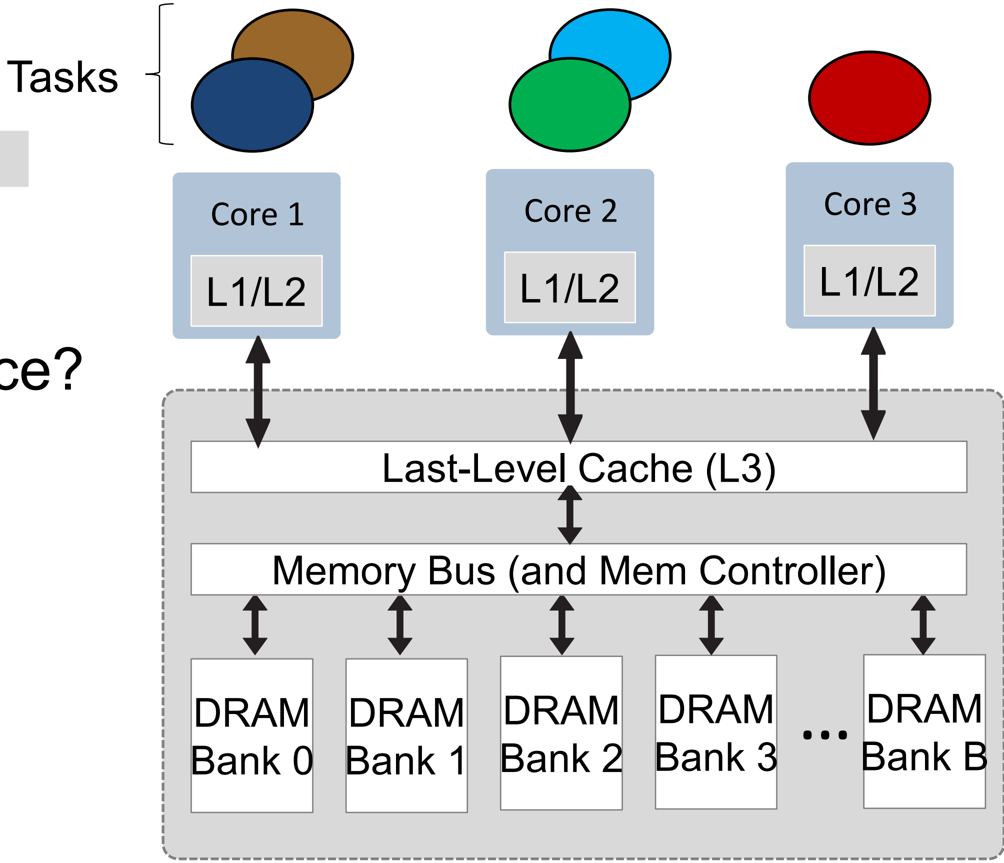
How to

manage inter-core interference?

prove timing correctness
considering inter-core interference?

considering that resources are
complex and often undocumented

Change application software
Change operating system
Change hardware
Change configuration



How to

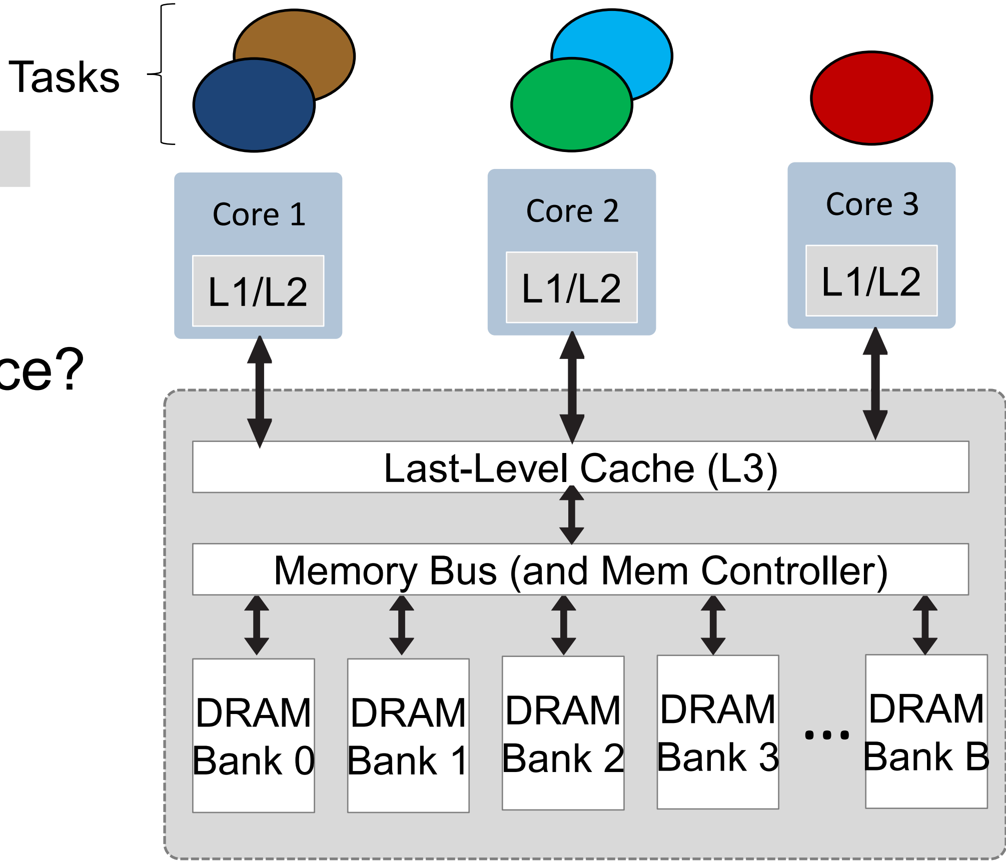
manage inter-core interference?

prove timing correctness
considering inter-core interference?

considering that resources are
complex and often undocumented

Change application software
Change operating system
Change hardware
Change configuration

Expensive



How to

manage inter-core interference?

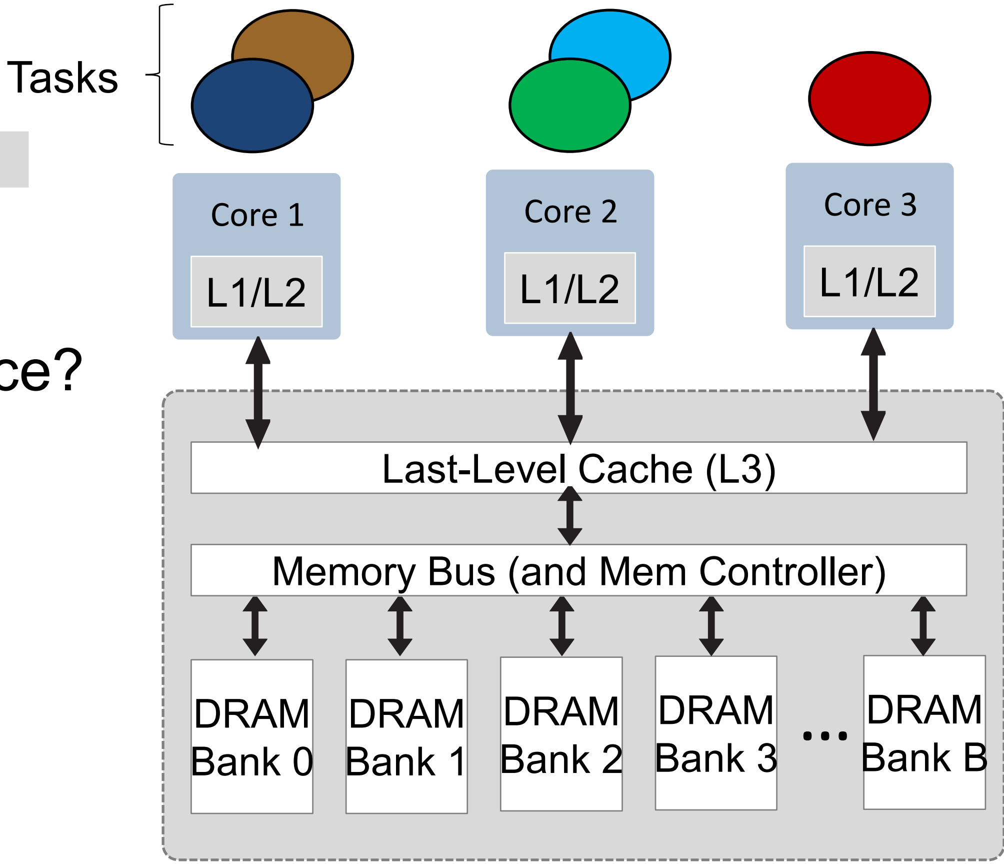
prove timing correctness considering inter-core interference?

considering that resources are complex and often undocumented

Change operating system

Change configuration

Focus of this talk



How to

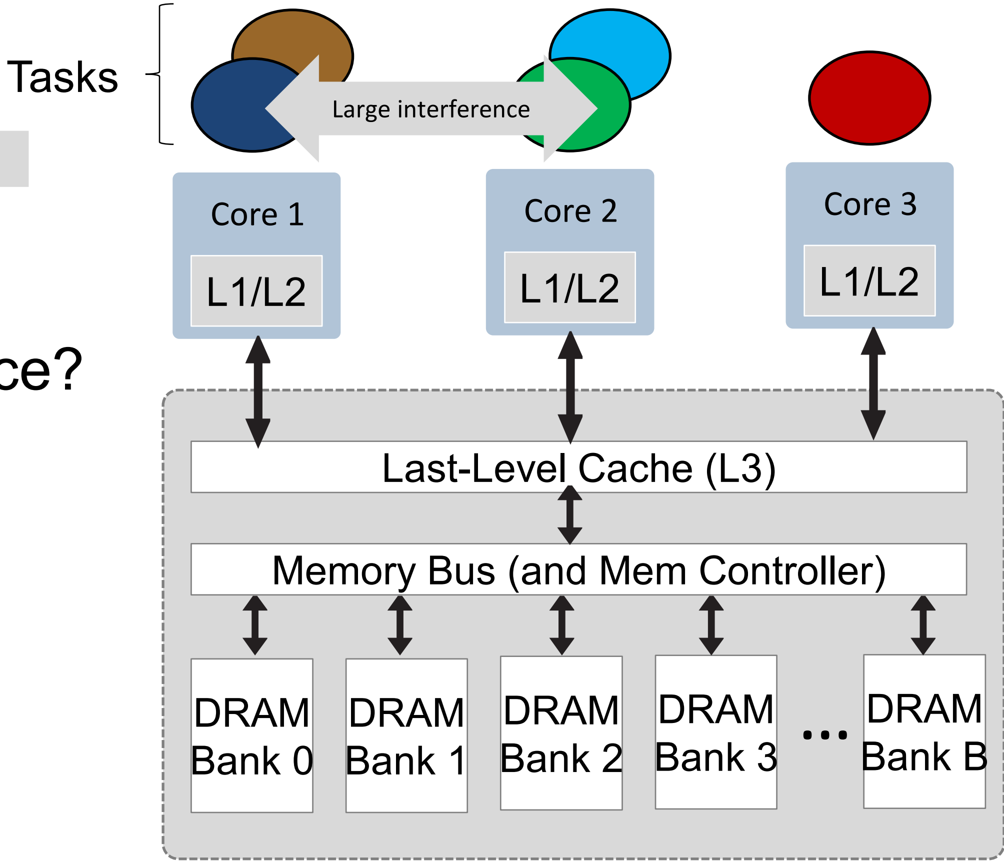
manage inter-core interference?

prove timing correctness
considering inter-core interference?

considering that resources are
complex and often undocumented

Change operating system

Change configuration



How to

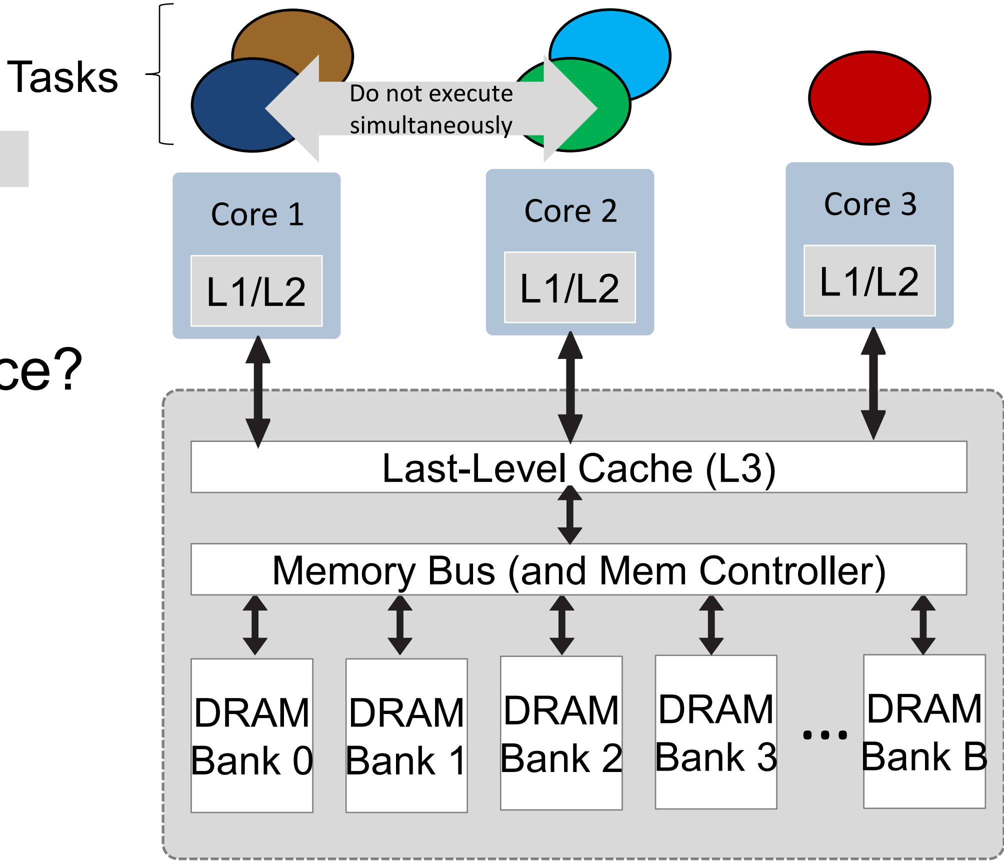
manage inter-core interference?

prove timing correctness considering inter-core interference?

considering that resources are complex and often undocumented

Change operating system

Change configuration



Outline

System model

Co-runner locking scheme

Schedulability analysis

Allocation

Implementation

Conclusions

System Model

Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

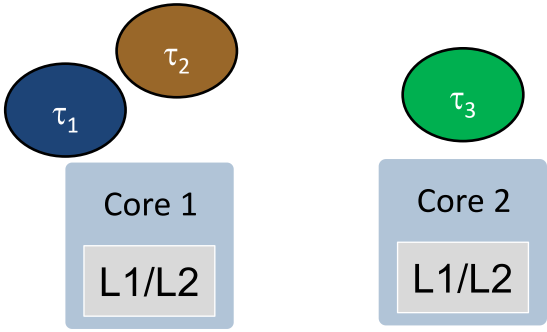
System Model

Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

System Model



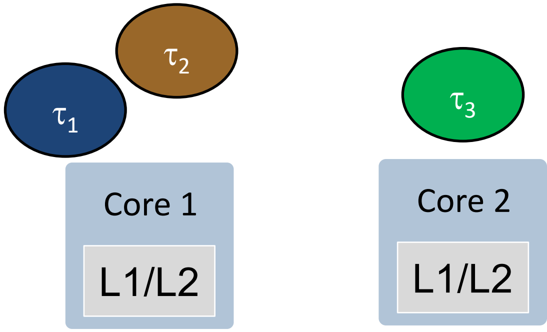
Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

System Model



Partitioned priority-based preemptive scheduling

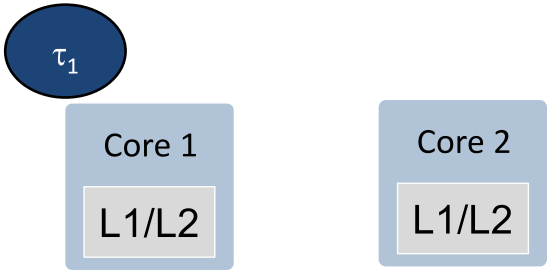
A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{ \qquad \}$

System Model



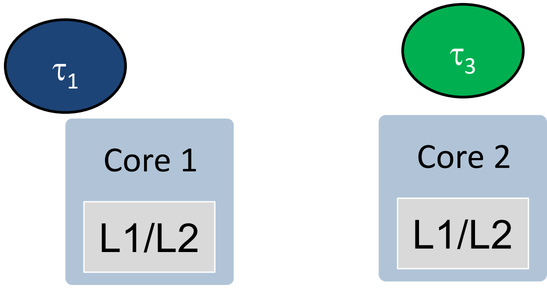
Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset \quad \}$



System Model

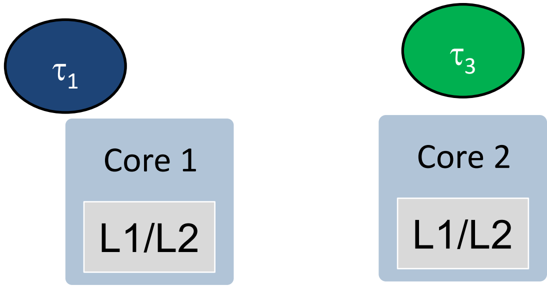
Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\quad \{\tau_3\}\}$



System Model

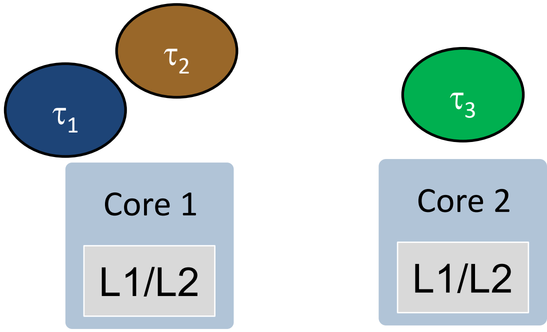
Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset, \{\tau_3\}\}$



System Model

Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset, \{\tau_3\}\}$, $S_2 = \{\emptyset, \{\tau_3\}\}$
- $S_3 = \{\emptyset, \{\tau_1\}, \{\tau_2\}\}$

System Model

Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset, \{\tau_3\}\}$, $S_2 = \{\emptyset, \{\tau_3\}\}$
- $S_3 = \{\emptyset, \{\tau_1\}, \{\tau_2\}\}$

$\sigma_{i,s}$: Slowdown factor for a task τ_i due to a co-runner set s

System Model

Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

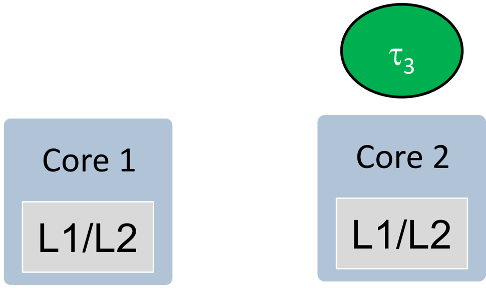
Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset, \{\tau_3\}\}$, $S_2 = \{\emptyset, \{\tau_3\}\}$
- $S_3 = \{\emptyset, \{\tau_1\}, \{\tau_2\}\}$

$\sigma_{i,s}$: Slowdown factor for a task τ_i due to a co-runner set s

- $\sigma_{3,\emptyset} = 1$, $\sigma_{3,\{\tau_1\}} = 3$, $\sigma_{3,\{\tau_2\}} = 2$

System Model



Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

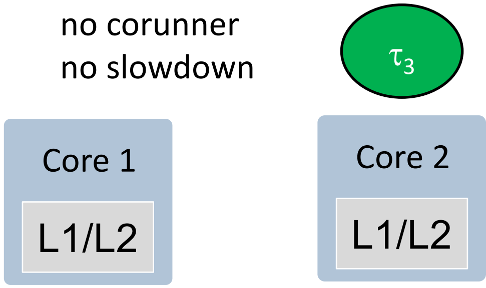
Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset, \{\tau_3\}\}$, $S_2 = \{\emptyset, \{\tau_3\}\}$
- $S_3 = \{\emptyset, \{\tau_1\}, \{\tau_2\}\}$

$\sigma_{i,s}$: Slowdown factor for a task τ_i due to a co-runner set s

- $\sigma_{3,\emptyset} = 1$

System Model



no corunner
no slowdown

Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

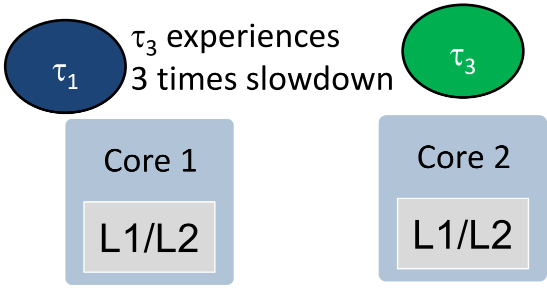
Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset, \{\tau_3\}\}$, $S_2 = \{\emptyset, \{\tau_3\}\}$
- $S_3 = \{\emptyset, \{\tau_1\}, \{\tau_2\}\}$

$\sigma_{i,s}$: Slowdown factor for a task τ_i due to a co-runner set s

- $\sigma_{3,\emptyset} = 1$

System Model



Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset, \{\tau_3\}\}$, $S_2 = \{\emptyset, \{\tau_3\}\}$
- $S_3 = \{\emptyset, \{\tau_1\}, \{\tau_2\}\}$

$\sigma_{i,s}$: Slowdown factor for a task τ_i due to a co-runner set s

$$\sigma_{3,\{\tau_1\}} = 3$$

System Model

Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset, \{\tau_3\}\}$, $S_2 = \{\emptyset, \{\tau_3\}\}$
- $S_3 = \{\emptyset, \{\tau_1\}, \{\tau_2\}\}$

$\sigma_{i,s}$: Slowdown factor for a task τ_i due to a co-runner set s

- $\sigma_{3,\emptyset} = 1$, $\sigma_{3,\{\tau_1\}} = 3$, $\sigma_{3,\{\tau_2\}} = 2$

System Model

Partitioned priority-based preemptive scheduling

A task τ_i is characterized by its minimum inter-arrival time (T_i), relative deadline (D_i), and execution requirement (C_i)

S_i: Set of potential co-runner sets of a task τ_i

Example: a dual-core system w/ τ_1 & τ_2 on core 1 and τ_3 on core 2

- $S_1 = \{\emptyset, \{\tau_3\}\}$, $S_2 = \{\emptyset, \{\tau_3\}\}$
- $S_3 = \{\emptyset, \{\tau_1\}, \{\tau_2\}\}$

$\sigma_{i,s}$: Slowdown factor for a task τ_i due to a co-runner set s

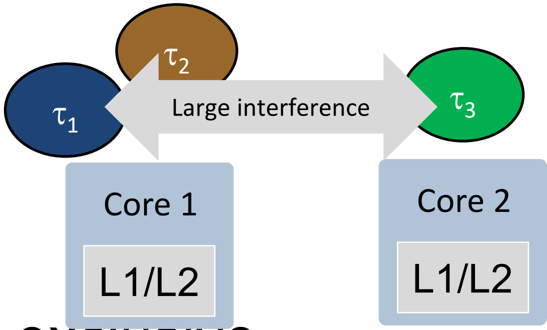
- $\sigma_{3,\emptyset} = 1$, $\sigma_{3,\{\tau_1\}} = 3$, $\sigma_{3,\{\tau_2\}} = 2$

$C_i \cdot \sigma_{i,s}$: Worst-case execution time of τ_i with a co-runner set s

Co-runner locking scheme

Key idea: representing and enforcing the mutually-exclusive conditions for selected co-runner tasks

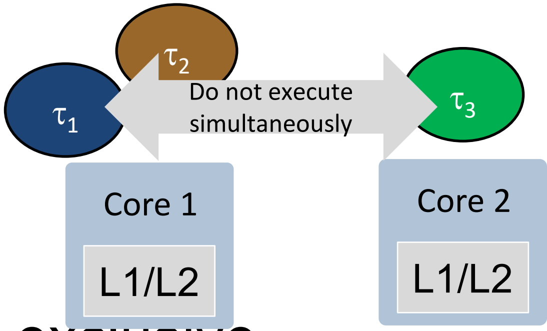
Co-runner exclusion set ϵ_i: co-runner tasks that are not allowed to execute in parallel with τ_i



Co-runner locking scheme

Key idea: representing and enforcing the mutually-exclusive conditions for selected co-runner tasks

Co-runner exclusion set ϵ_i: co-runner tasks that are not allowed to execute in parallel with τ_i



Co-runner locking scheme

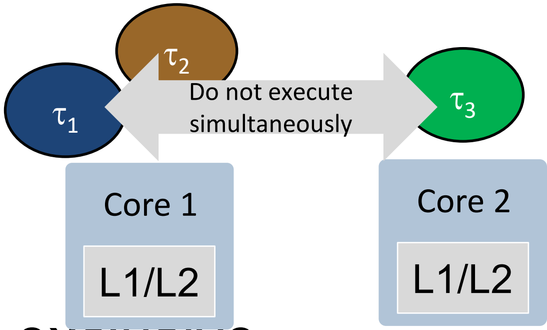
Key idea: representing and enforcing the mutually-exclusive conditions for selected co-runner tasks

Co-runner exclusion set ϵ_i: co-runner tasks that are not allowed to execute in parallel with τ_i

$\epsilon_1 = \{\tau_3\}$

$\epsilon_3 = \{\tau_1\}$

Co-runner locking scheme



Key idea: representing and enforcing the mutually-exclusive conditions for selected co-runner tasks

Co-runner exclusion set ϵ_i: co-runner tasks that are not allowed to execute in parallel with τ_i

Co-runner locking scheme

Key idea: representing and enforcing the mutually-exclusive conditions for selected co-runner tasks

Co-runner exclusion set ϵ_i: co-runner tasks that are not allowed to execute in parallel with τ_i

With ϵ_i, *true* co-runners that execute in parallel with τ_i at runtime are determined: $G_i = \{s | s \in S_i \land \nexists(\tau_j \in s \land \tau_j \in \epsilon_i)\}$

Co-runner locking scheme

Key idea: representing and enforcing the mutually-exclusive conditions for selected co-runner tasks

Co-runner exclusion set ϵ_i: co-runner tasks that are not allowed to execute in parallel with τ_i

With ϵ_i, *true* co-runners that execute in parallel with τ_i at runtime are determined: $G_i = \{s | s \in S_i \wedge \nexists(\tau_j \in s \wedge \tau_j \in \epsilon_i)\}$

Slowdown factors with co-runner locking: $\rho_i = \{\sigma_{i,s} | s \in G_i\}$

Maximum slowdown: $\theta_i = \max \sigma_{i,s}$

Co-runner locking scheme

Key idea: representing and enforcing the mutually-exclusive conditions for selected co-runner tasks

Co-runner exclusion set ϵ_i: co-runner tasks that are not allowed to execute in parallel with τ_i

With ϵ_i, *true* co-runners that execute in parallel with τ_i at runtime are determined: $G_i = \{s | s \in S_i \land \nexists(\tau_j \in s \land \tau_j \in \epsilon_i)\}$

Slowdown factors with co-runner locking: $\rho_i = \{\sigma_{i,s} | s \in G_i\}$

Maximum slowdown: $\theta_i = \max \sigma_{i,s}$

Possible to use with real-time synchronization protocols (see paper)

Execution Control Policy

Co-runner exclusion set ϵ_i does not determine the execution order of co-runners when they compete

Possible approaches:

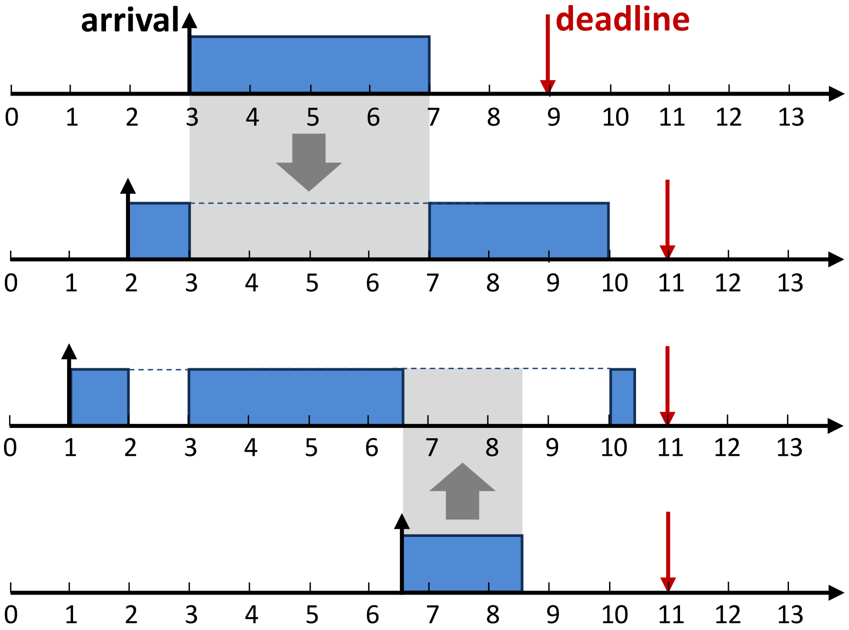
- Priority-based (this paper)
- First-come, first-served
- Other separate ordering different from task priority

Priority-based Execution Control

Core 1 $\tau_1 : \epsilon_1 = \{\tau_3\}$

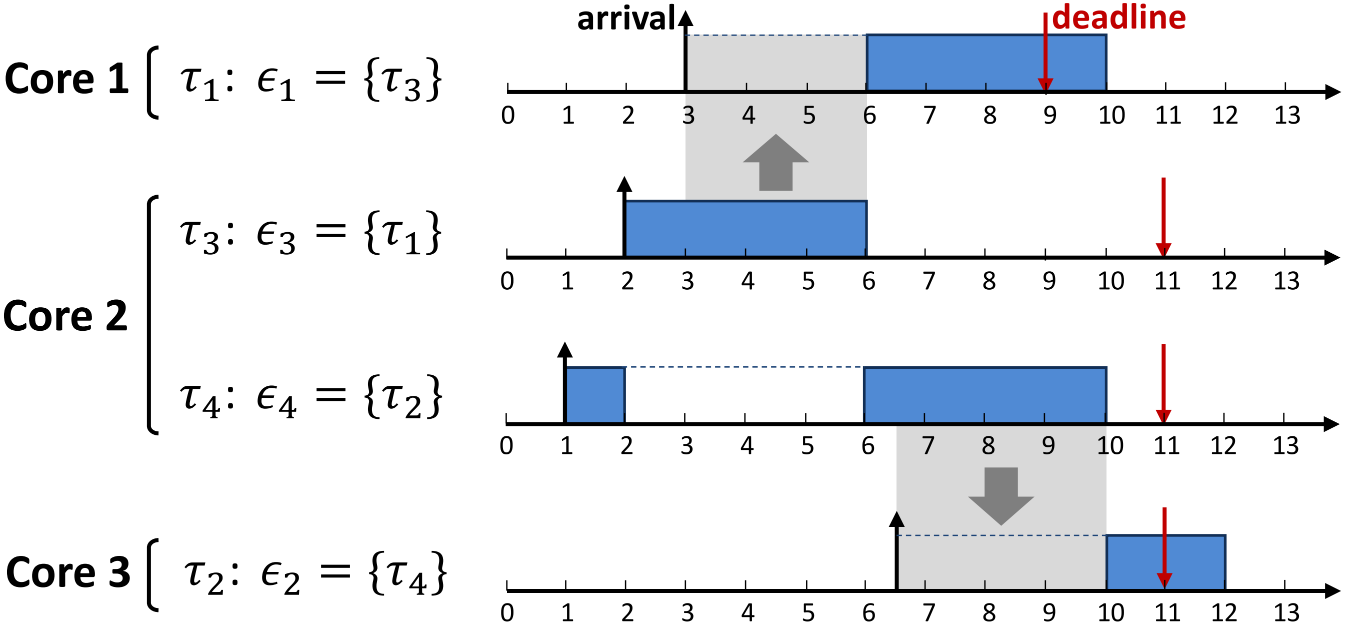
Core 2 $\tau_3 : \epsilon_3 = \{\tau_1\}$, $\tau_4 : \epsilon_4 = \{\tau_2\}$

Core 3 $\tau_2 : \epsilon_2 = \{\tau_4\}$



Tasks in a mutual-exclusive relationship (ϵ_i) behave as if they were preemptively scheduled on the same core based on their priorities (called “co-runner preemption”)

Other Control Policies: FCFS



FCFS takes away the opportunity for higher-priority tasks to preempt lower-priority tasks, possibly resulting in poor schedulability

Schedulability Analysis

Pessimistic. It assumes tasks always get the worst-case slowdown throughout their execution

Baseline analysis:

$$R_i = C_i \cdot \theta_i + \sum_{\tau_j \in hpp(\tau_i)} \left\lceil \frac{R_i + I_j(C_j \cdot \theta_j)}{T_j} \right\rceil C_j \cdot \theta_j + \sum_{\tau_k \in \epsilon_i \wedge \tau_k \in hp(\tau_i)} \left\lceil \frac{R_i + I_k(C_k \cdot \theta_k)}{T_k} \right\rceil C_k \cdot \theta_k$$

Co-runner preemption

$$where$$

$$I_j(x) = \begin{cases} \max(R_j - x, 0) & , \exists \tau_y : \tau_y \in \epsilon_j \wedge \tau_y \in hp(\tau_j) \\ 0 & , otherwise \end{cases}$$

Two more advanced approaches:
- Job-oriented Slowdown Analysis
- Load-oriented Slowdown Analysis

Job-oriented Slowdown Analysis

Focuses on how long interfering co-runners can actually execute during the response time of a task under analysis

Find the maximum cumulative execution time of a co-runner τ_k:

$$\zeta_{i,k} = \min\Bigg(\bigg\lfloor \frac{R_i + I_k(C_k \cdot \theta_k)}{T_k} \bigg\rfloor C_k \cdot \theta_k + \min\Big((R_i + I_k(C_k \cdot \theta_k)) \bmod T_k, C_k \cdot \theta_k\Big), R_i\Bigg)$$

Maximize the execution time of a job of τ_i ($C_i^* \leq C_i \cdot \theta_i$):

$$C_i^* = \sum_{0<k\leq|\rho_i|} V_{i,k} \cdot \phi_{i,k} \qquad \phi_{i,k} = \min\bigg(C_i - \sum_{0<l<k} \phi_{i,l},\ \frac{\xi_{i,X_{i,k}}}{V_{i,k}}\bigg)$$

Calculate the response time of τ_i:

$$R_i = C_i^* + \sum_{\tau_j \in hpp(\tau_i)} \bigg\lceil \frac{R_i + I_j(C_j^*)}{T_j} \bigg\rceil C_j^* + \sum_{\tau_k \in \epsilon_i \wedge \tau_k \in hp(\tau_i)} \bigg\lceil \frac{R_i + I_k(C_k^*)}{T_k} \bigg\rceil C_k^*$$

Solvable by fixed-point iteration

Example 1

Core 1 $\tau_1: (C_1 = 4, T_1 = 10)$

- $\rho_{1,\{\tau_2\}} = 4$
- $\rho_{1,\{\tau_3\}} = 2$
- $\rho_{1,\{\tau_4\}} = 1$

Core 2 $\tau_2: (C_2 = 1, T_2 = 10)$

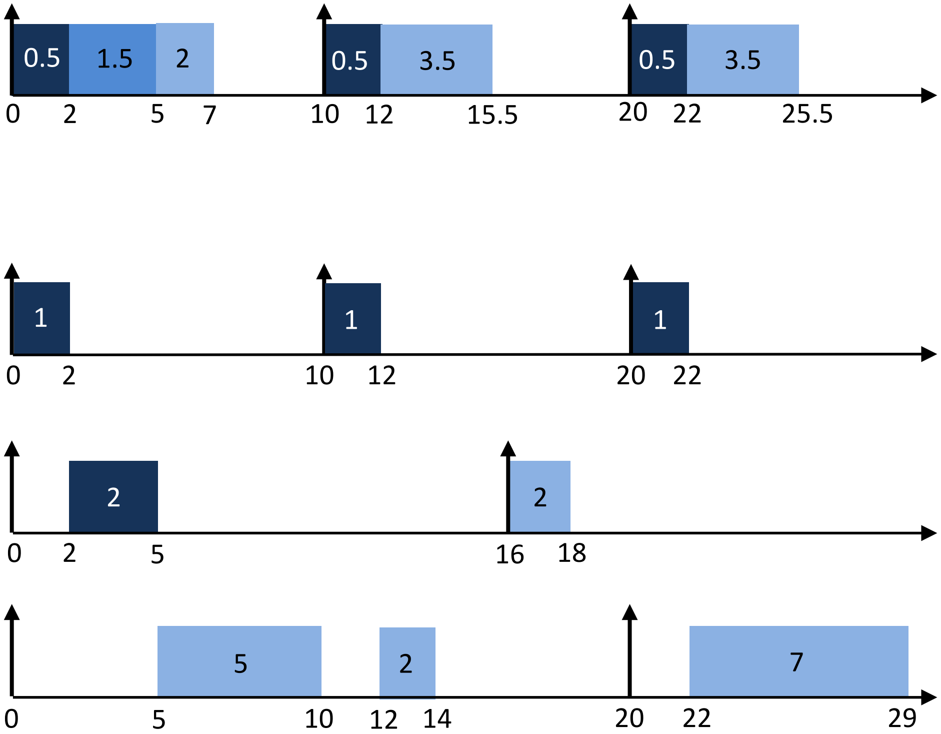
- $\rho_{2,\{\tau_1\}} = 2$

$\tau_3: (C_3 = 2, T_3 = 16)$

- $\rho_{3,\{\tau_1\}} = 1.5$

$\tau_4: (C_4 = 7, T_4 = 20)$

- $\rho_{4,\{\tau_1\}} = 1$



Response time of τ_1:

- Baseline analysis: 16 → Misjudging that τ_1 misses the deadline
- Job-oriented slowdown analysis: 7 → Same as in the figure

Load-oriented Slowdown Analysis

Job-oriented analysis computes each job’s slowdown separately and then analyzes the response time

Instead, load-oriented analysis bounds the cumulative slowdown that can be possibly imposed on all execution requirements during the response time of a given task

Find the execution requirements E_i during the response time of τ_i:

$$E_i = C_i + \sum_{\tau_j \in H_i \setminus \{\tau_i\}} \left\lceil \frac{R_i + I_j(C_j)}{T_j} \right\rceil C_j$$

Maximize the response time considering possible slowdowns for E_i:

$$R_i = \sum_{0<k\leq|\rho_i^*|} V^*_{i,k} \cdot \phi^*_{i,k} \qquad \phi^*_{i,k} = \min\left(E_i - \sum_{0<l<k} \phi^*_{i,l},\ \frac{\xi_{i,X^*_{i,k}}}{V^*_{i,k}}\right)$$

Solvable by fixed-point iteration

Example 2

Core 1 $\tau_1: (C_1 = 1, T_1 = 10)$

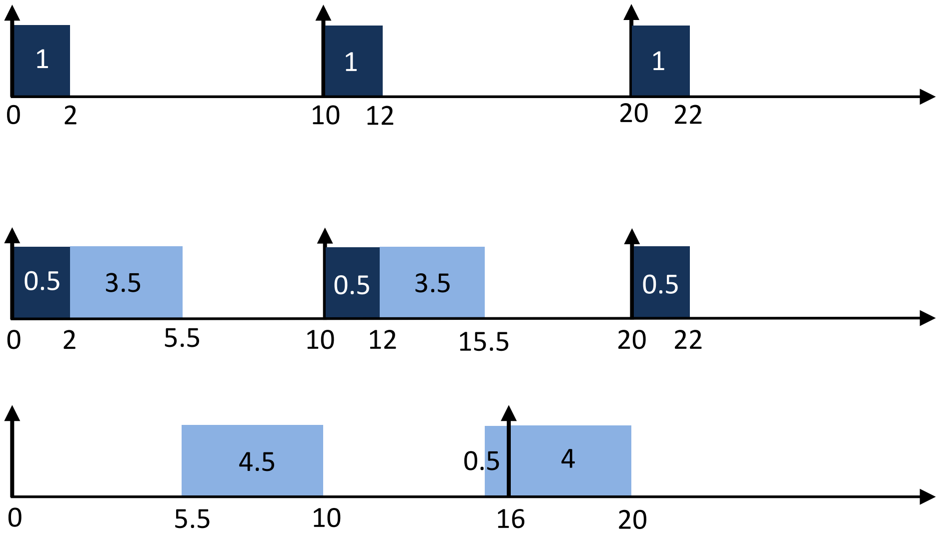
- $\rho_{1,\{\tau_2\}} = 2$
- $\rho_{1,\{\tau_3\}} = 2$

Core 2 $\tau_2: (C_2 = 4, T_2 = 10)$

- $\rho_{2,\{\tau_1\}} = 4$

$\tau_3: (C_3 = 5, T_3 = 16)$

- $\rho_{3,\{\tau_1\}} = 3$



Response time of τ_3:

- Job-oriented: Fail
- Load-oriented: 16 → Same as in the figure

Job-oriented and load-oriented analyses do not dominate each other

- Load oriented analysis fails for Example 1
- Use them jointly by taking the minimum between the two

How to decide co-runner exclusion sets?

We propose two heuristics:

1) SA: Simulated annealing

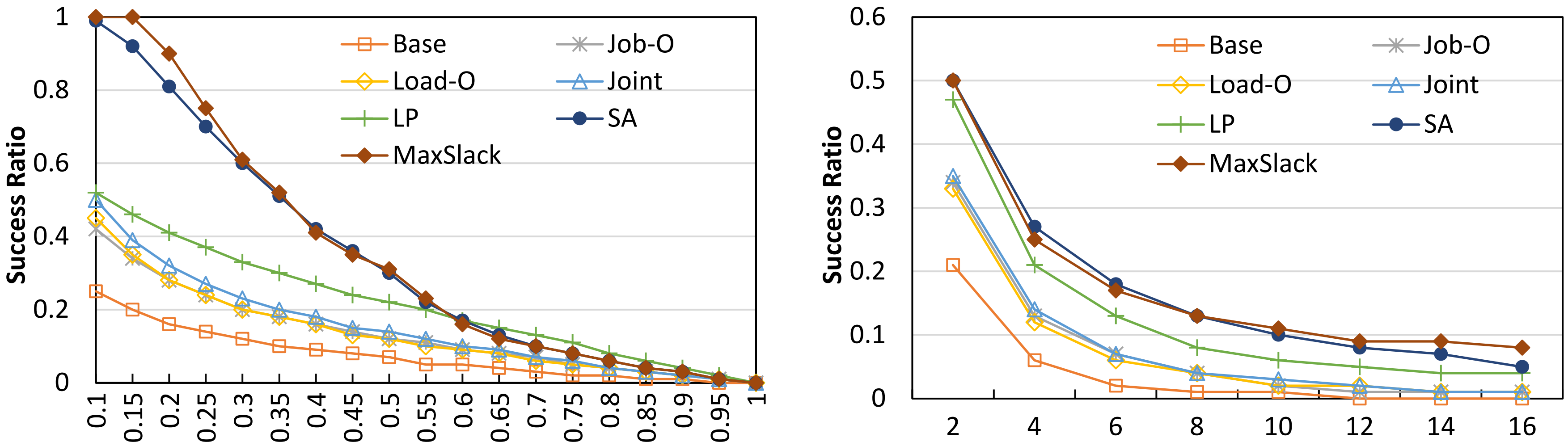
   Cost for SA: # of schedulable tasks in a given taskset

2) MaxSlack: Maximize relative slack

   Adds a co-runner task to the exclusion set if doing so increases the cumulative sum of slack for all tasks

Evaluation

Schedulability success ratio (SR) of random tasksets



SR as a function of *mul*
(per-core utilization)

SR as a function of *n*
(# of tasks)

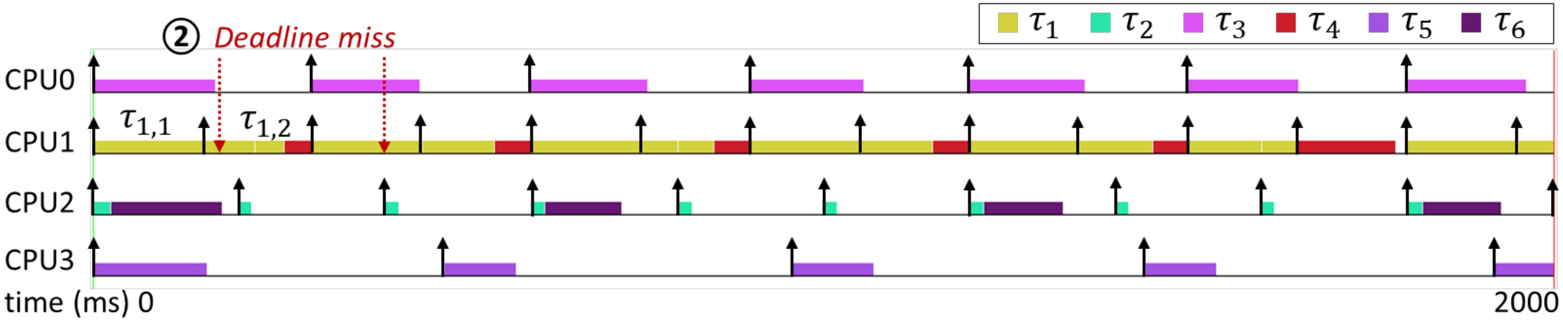
Co-runner locking (Joint analysis + MaxSlack/SA)
- Almost 2x of improvement over the “no locking” case until mul > 0.3

Case Study: Nvidia Xavier AGX

Inter-core interference (slowdown): 1.01 – 1.81x (much smaller than the worst cases reported in the literature)

Scheduling with co-runner locking:

Scheduling without co-runner locking:



Conclusions

Co-runner locking scheme:

- New way to prevent excessive slowdown scenarios
- Applicable to priority-based preemptive scheduling
- No extra restrictions that related prior work requires:
  - PREM: serializes memory phases of all tasks in the system
  - Non-preemptive time-triggered scheduling
  - RT-Gang: tasks in each gang have the same release offset and period
- Effective alternative when cache/DRAM partitioning methods are not available or they cannot eliminate all the interference penalties

Thanks!