PARALLEL COMPUTING SYSTEMS

UNIVERSITEIT VAN AMSTERDAM

TCPS: A Task and Cache-Aware Partitioned Scheduler for Hard Real-Time Multi-core Systems

Yixian Shen, Jun Xiao and Andy D Pimentel

LCTES-2022

Outline

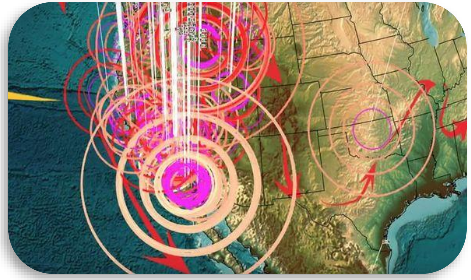
- Research Background
- Start-of-art techniques
- Contribution
- TCPS approach
- Experiments
- Conclusion

Background

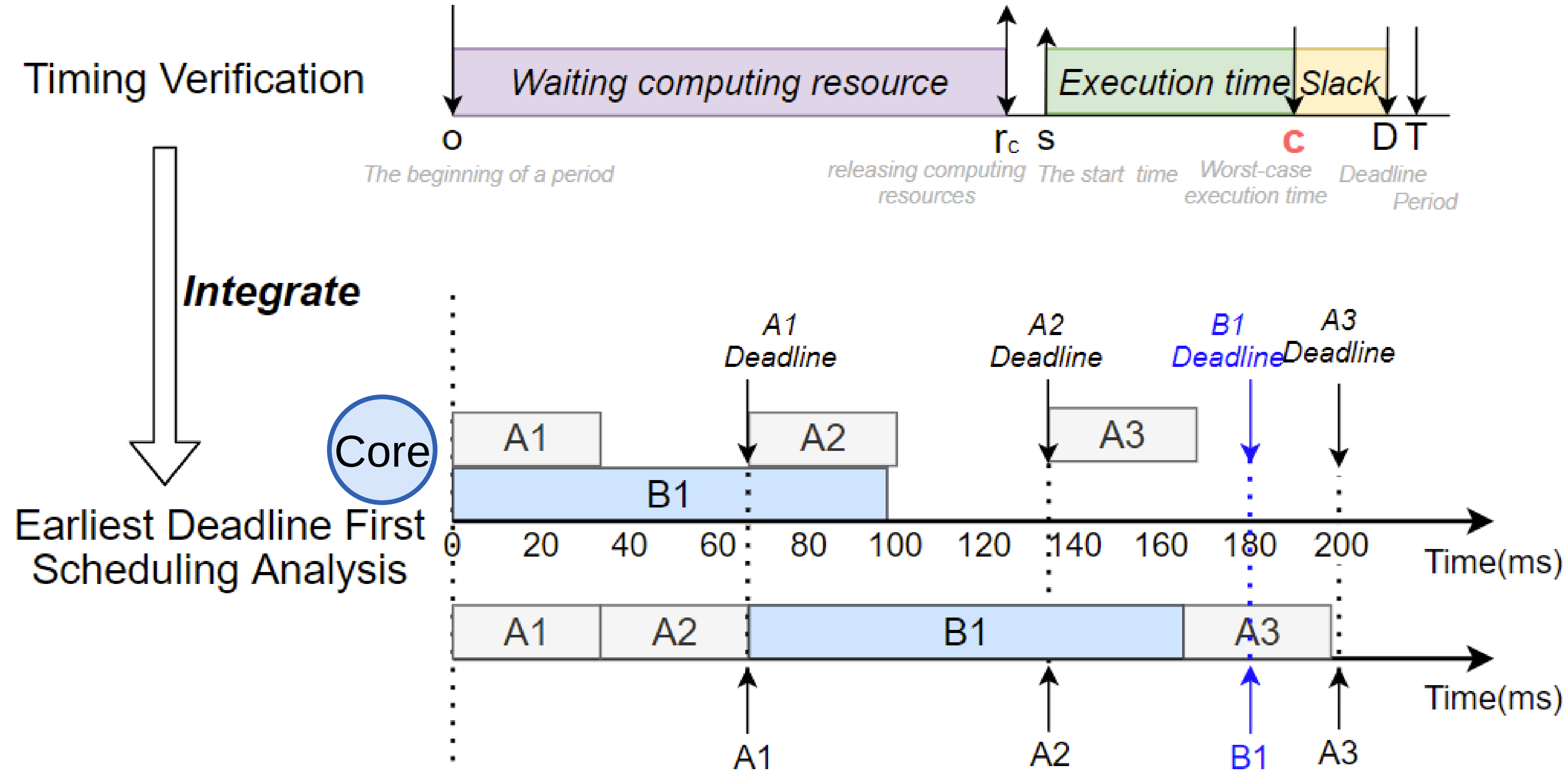
- Time predictability is ***crucial*** for hard latency-critical applications
  - **Airbag restraint systems:** Car airbag, Electronic parachute
  - **Precise instrument:** Leonardo‘s Robot, Microscope
  - **Avionic systems:** Airbus, Boeing Co.
  - **Acute Sensing systems:** Earthquake sensor, Fire alarm sensor
  - **Unmanned aerial vehicles:** Drones





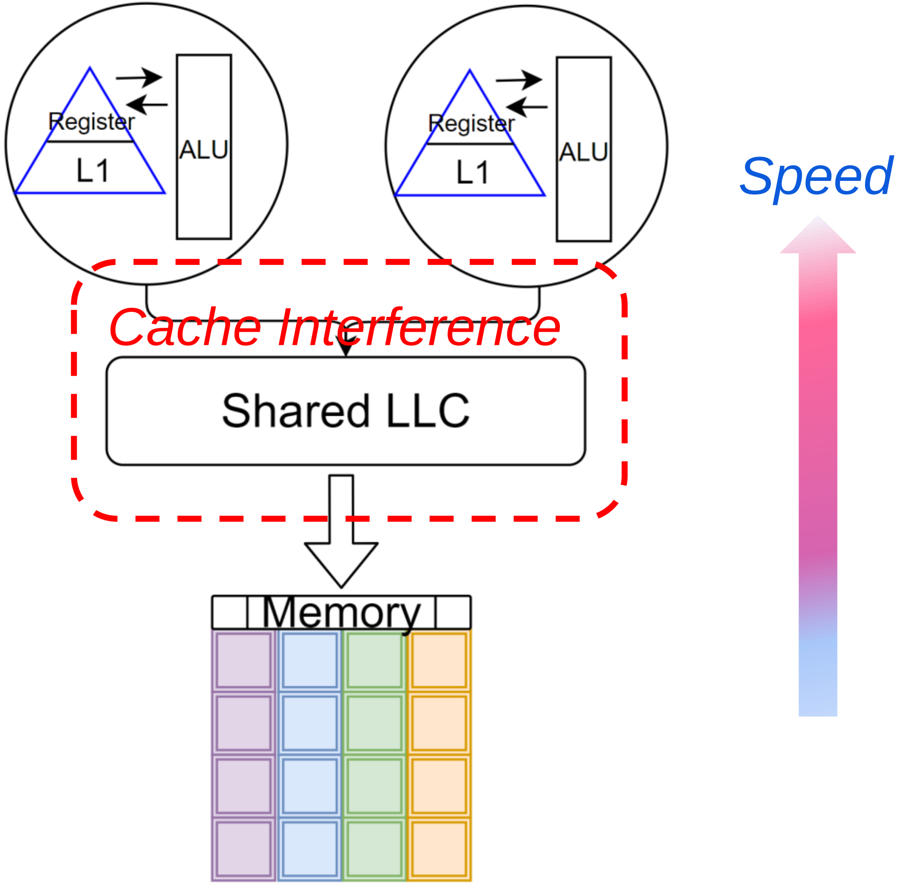



Timing verification in single-core real-time systems



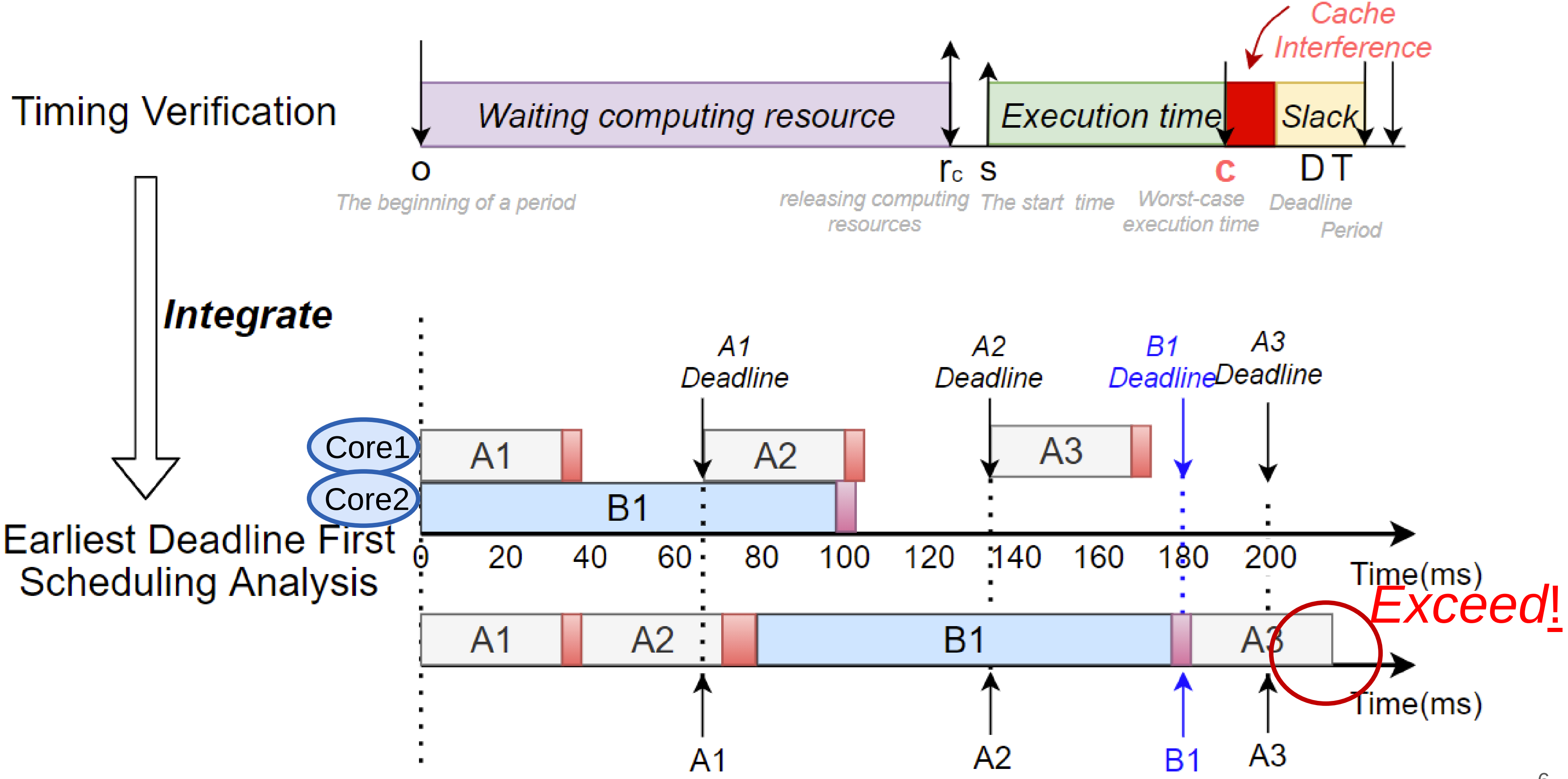
Challenges

- Time accidents degrade time predictability

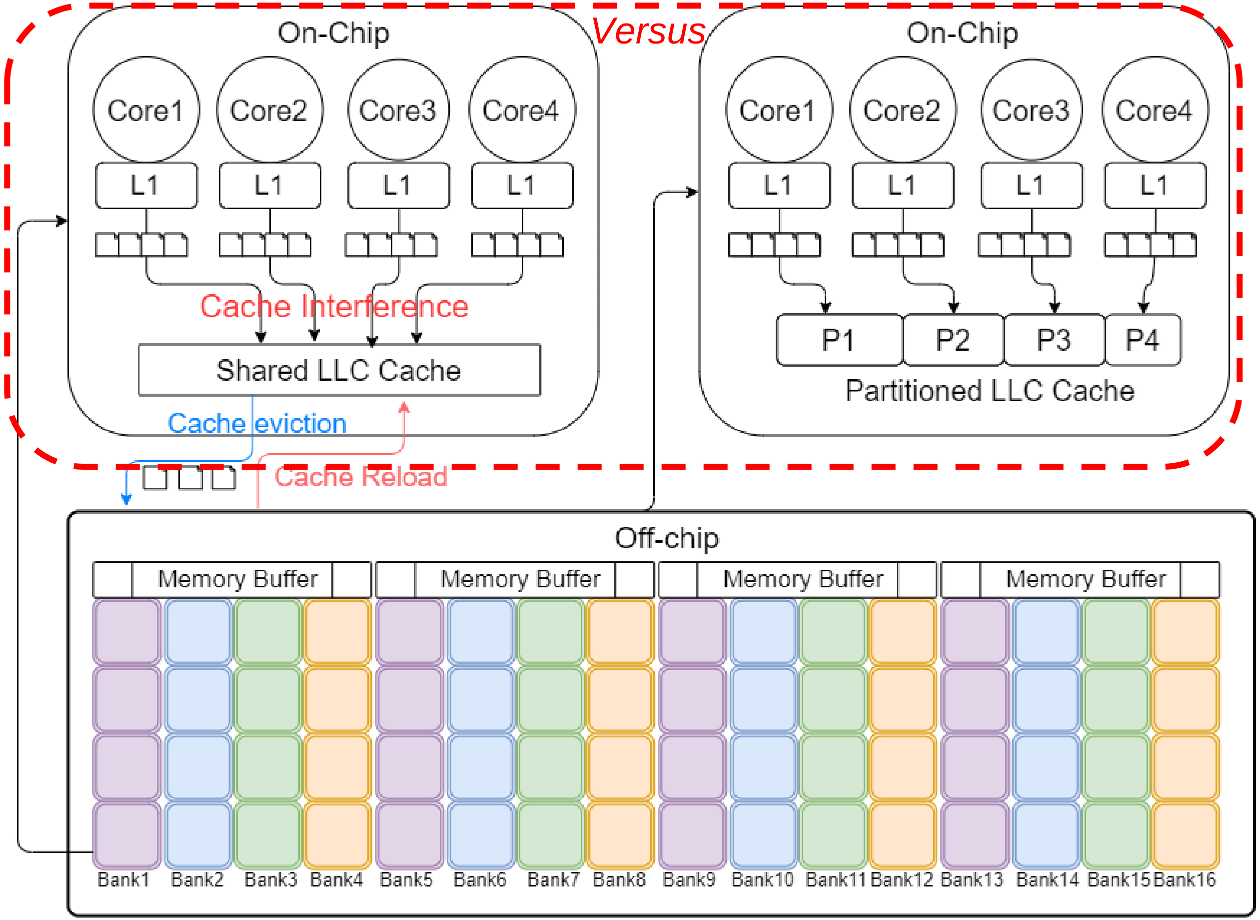


- Types of time accidents
  - Pipeline stalls
  - Branch mispredictions
  - Bus collisions
  - Cache misses
  - TLB misses
  - Memory refresh of DRAM

Timing verification in multi-core real-time systems

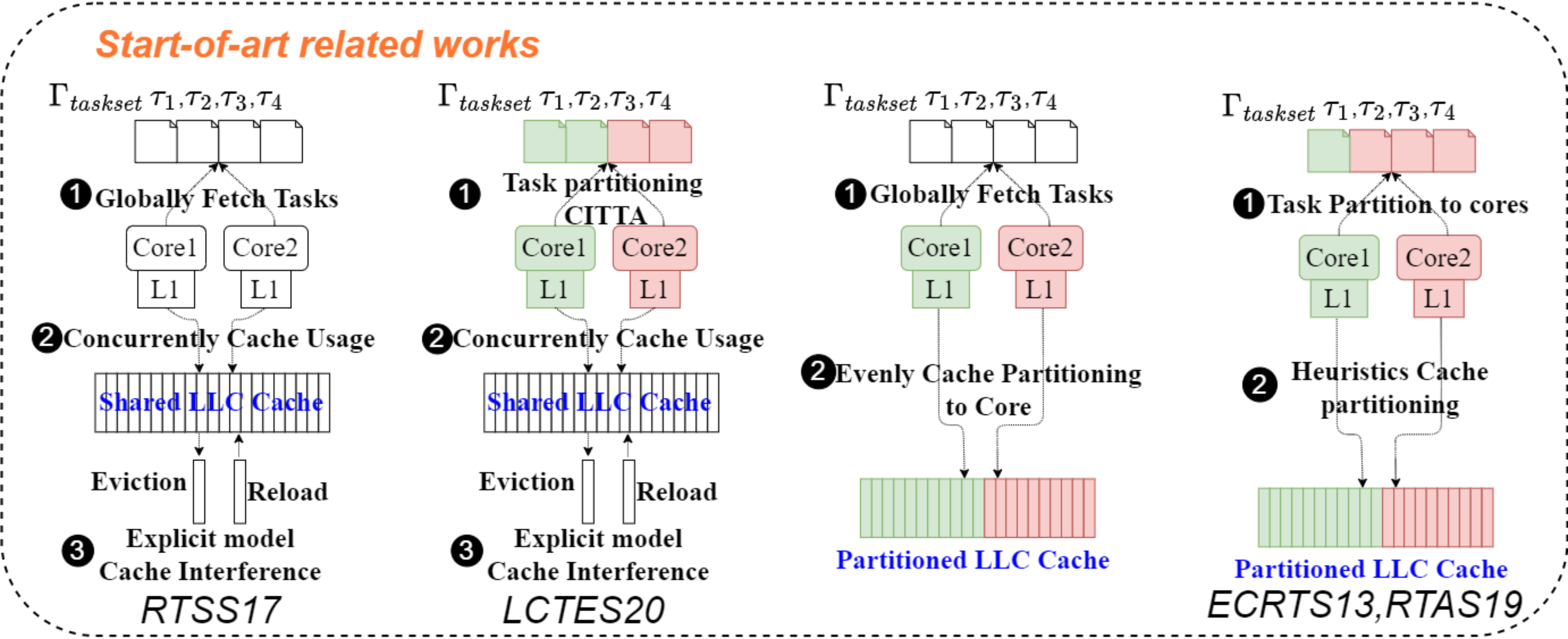


Two mainstream methods to improve the time predictability



- Explicitly calculate the cache inference
- Cache partitioning

Motivation



Start-of-art related works
$\Gamma_{taskset}$ τ_1,τ_2,τ_3,τ_4
❶ Globally Fetch Tasks
Core1
Core2
L1
L1
❷ Concurrently Cache Usage
Shared LLC Cache
Eviction
Reload
❸ Explicit model Cache Interference
RTSS17
$\Gamma_{taskset}$ τ_1,τ_2,τ_3,τ_4
❶ Task partitioning CITTA
Core1
Core2
L1
L1
❷ Concurrently Cache Usage
Shared LLC Cache
Eviction
Reload
❸ Explicit model Cache Interference
LCTES20
$\Gamma_{taskset}$ τ_1,τ_2,τ_3,τ_4
❶ Globally Fetch Tasks
Core1
Core2
L1
L1
❷ Evenly Cache Partitioning to Core
Partitioned LLC Cache
$\Gamma_{taskset}$ τ_1,τ_2,τ_3,τ_4
❶ Task Partition to cores
Core1
Core2
L1
L1
❷ Heuristics Cache partitioning
Partitioned LLC Cache
ECRTS13,RTAS19

Contribution

- TCPS: a new heuristic task and cache-aware partitioned scheduling policy for non-preemptive real-time multi-core systems, which combines the benefits of cache partitioning and partitioned scheduling.

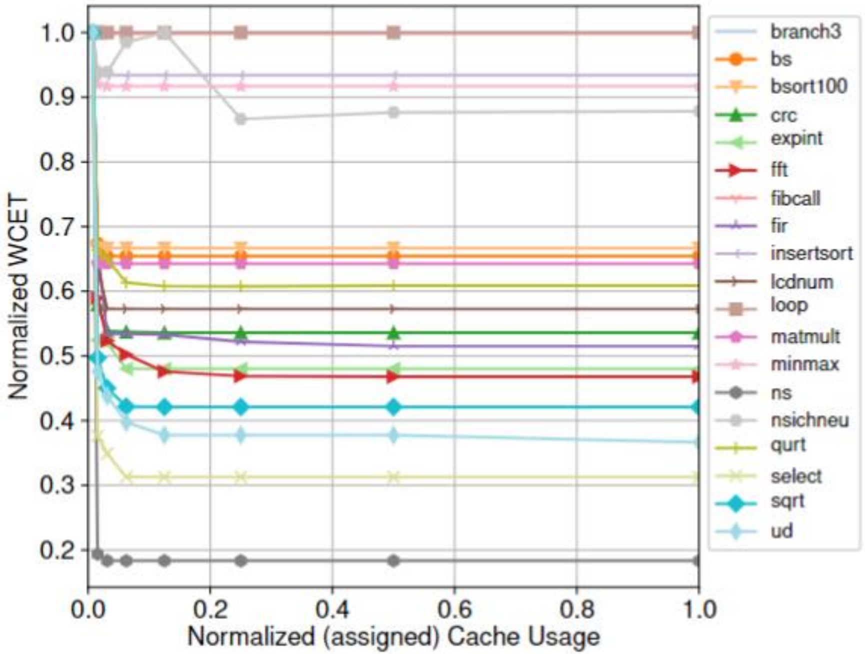
- We evaluate the design choices of our scheduling policy, comparing it with all different combinations of methods to address cache interference and scheduling approaches.

- We conduct comprehensive experiments and identify how different parameter settings affect the relative performance of partitioned and non-partitioned shared caches for different real-time schedulers

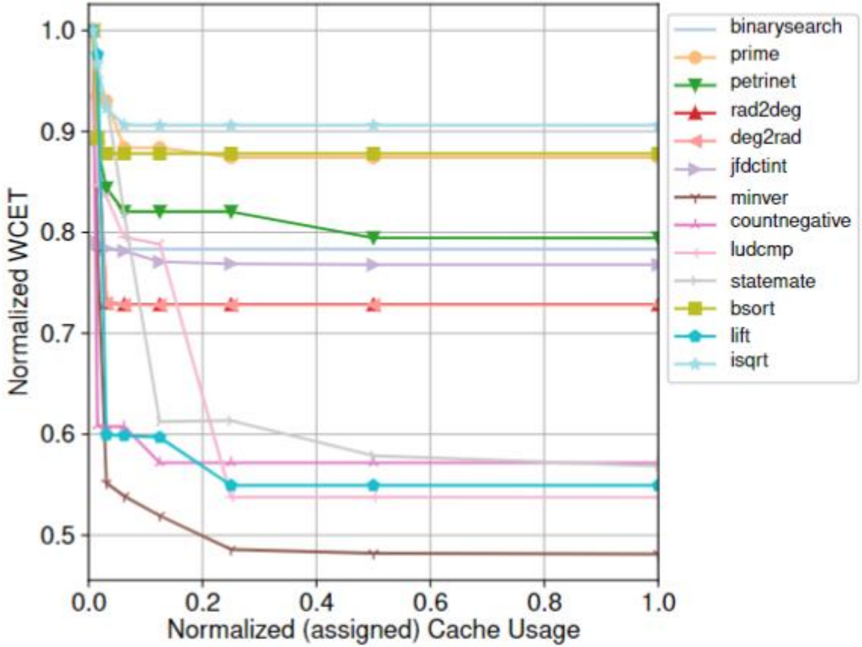
System model

- Real-time task $\tau_k = (C_k, D_k, T_k)$
- Multiple real-time tasks $\tau = (\tau_1, \tau_2, \ldots, \tau_n)$
- Multiple processor with ***m*** cores $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$
  - Shared caches
  - Cache interference
    - Inter-core cache interference
    - Intra-core cache interference
  - C_k does not count cache interference
- Scheduling policy
  - Non-preemptive EDF → no intra-core cache interference

TCPS approach

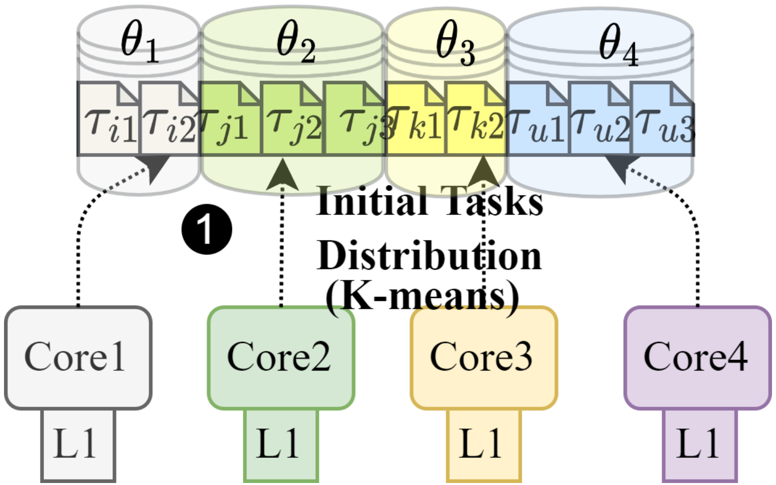


(a) Mälardalen benchmarks



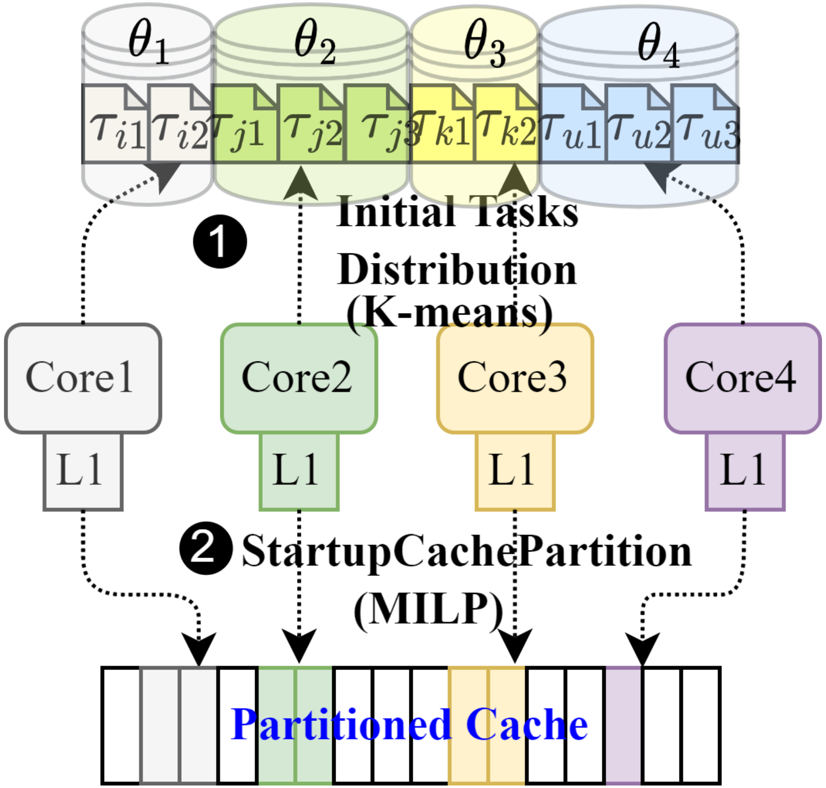
(b) TACLeBench

TCPS approach



$$\underset{\theta}{argmin} \sum_{i=1}^{m} \frac{1}{2|\theta_i|} \sum_{\tau_k, \tau_j \in \theta_i} \| \vec{g}_k - \vec{g}_j \|^2$$

TCPS approach



$$\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$$

$$min\{U_1 + U_2 + \ldots + U_n\}$$

$$min\{\frac{1}{T_1} * c_1 + \frac{1}{T_2} * c_2, \ldots, + \frac{1}{T_n} * c_n\}$$

Constraints:

$$p_1 + p_2 + \ldots + p_n \leq n$$

$$c_1 * A_1 + p_1 * B_1 + k_1 = 0$$

$$c_2 * A_2 + p_2 * B_2 + k_2 = 0$$

$$\ldots$$

$$c_n * A_n + p_n * B_n + k_n = 0$$

The cache partitioning algorithm

$$\tau = \{\tau_1, \tau_2, ..., \tau_n\}$$

$$min\{U_1 + U_2 + ... + U_n\}$$

$$min\{\frac{1}{T_1} * c_1 + \frac{1}{T_2} * c_2, ..., + \frac{1}{T_n} * c_n\}$$

Constraints:

$$p_1 + p_2 + ... + p_n \leq n$$

$$c_1 * A_1 + p_1 * B_1 + k_1 = 0$$

$$c_2 * A_2 + p_2 * B_2 + k_2 = 0$$

$$...$$

$$c_n * A_n + p_n * B_n + k_n = 0$$

$$U_i \leftarrow utilization$$

$$T_i \leftarrow Period$$

$$c_i \leftarrow WCET$$

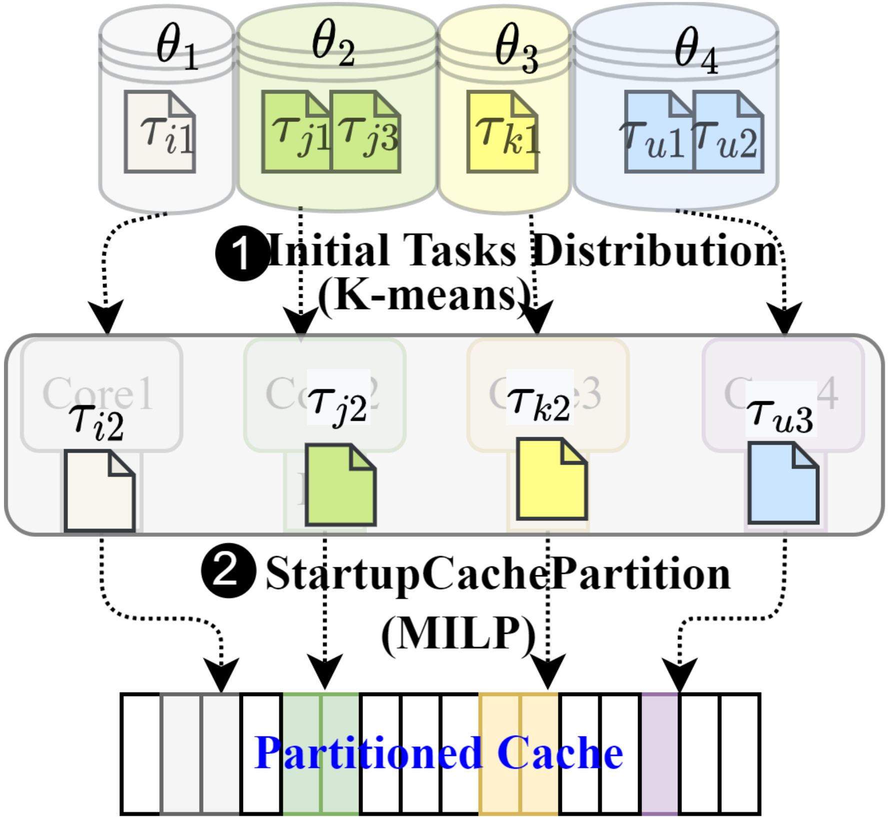
$$U_i = \frac{c_i}{T_i}$$

$$p_i \leftarrow partition\ size$$

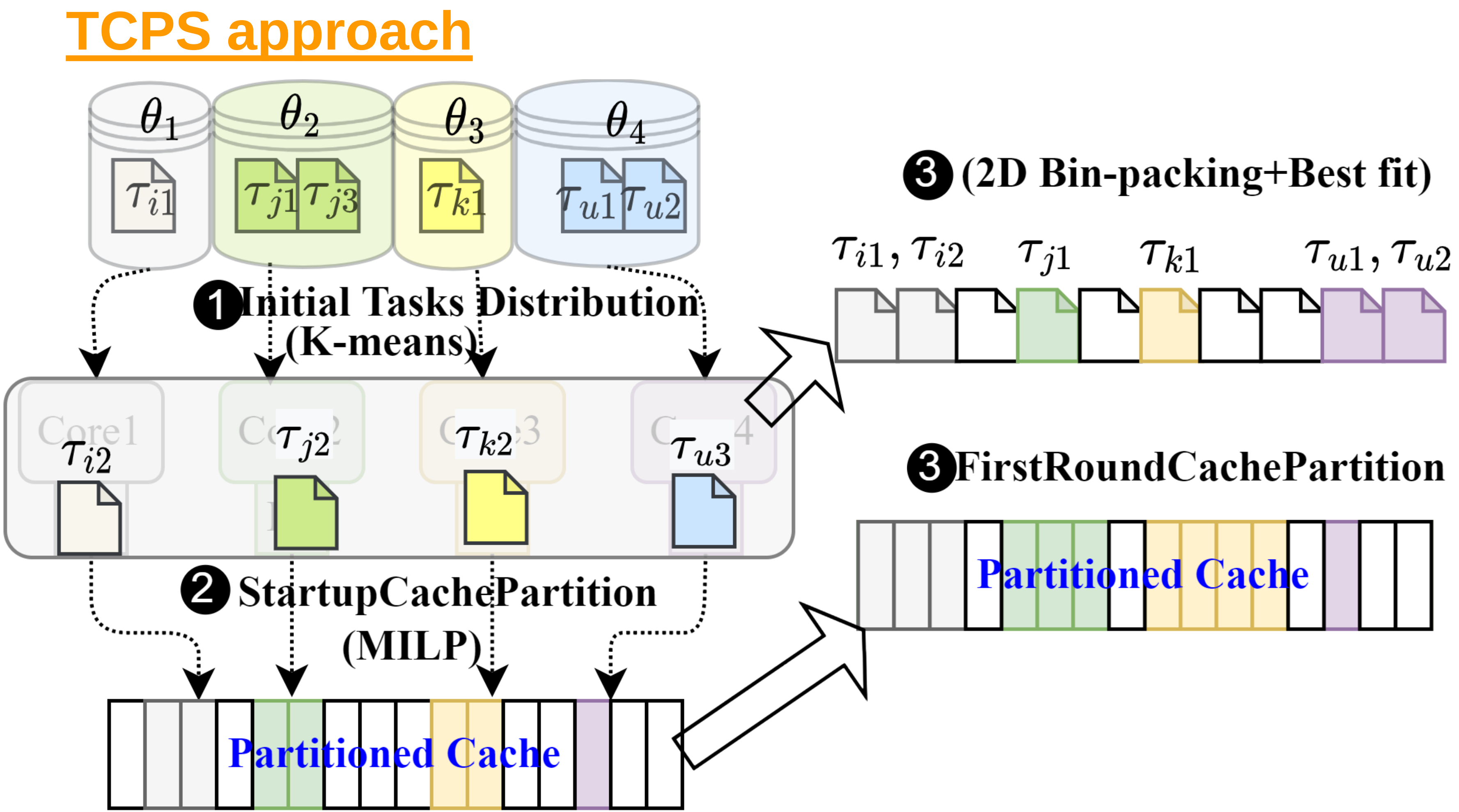
$$A_i\ , B_i \leftarrow coefficient$$

$$k_i \leftarrow offset$$

TCPS approach

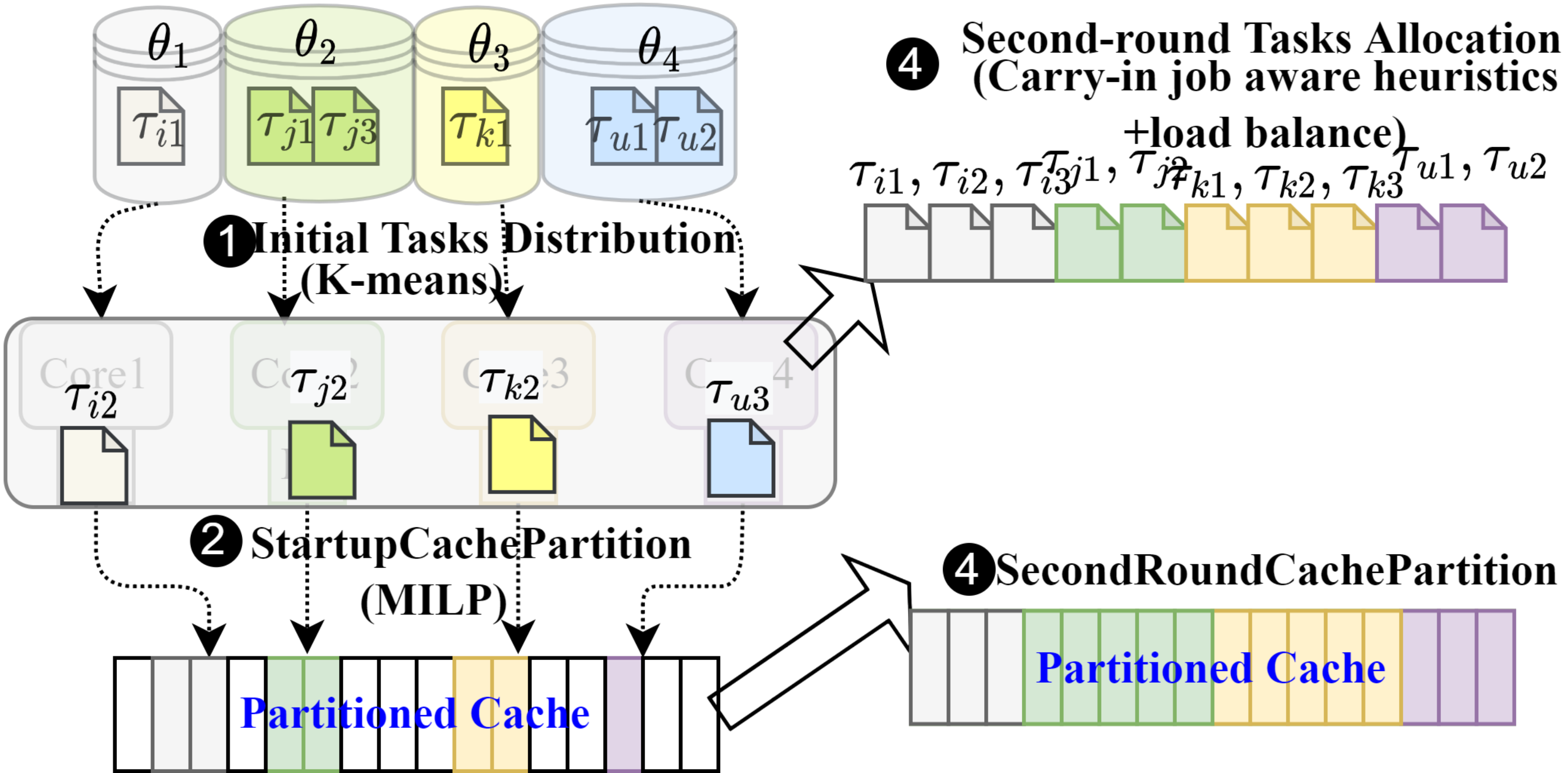


$$D_k \geq \sum_{\substack{\tau_j \in \tau(\pi_i) \cup \{\tau_k\} \\ D_j < D_k}} DBF^*(\tau_j, D_k) + \max_{\substack{\tau_j \in \tau(\pi_i) \cup \{\tau_k\} \\ D_j > D_k}} C_j$$

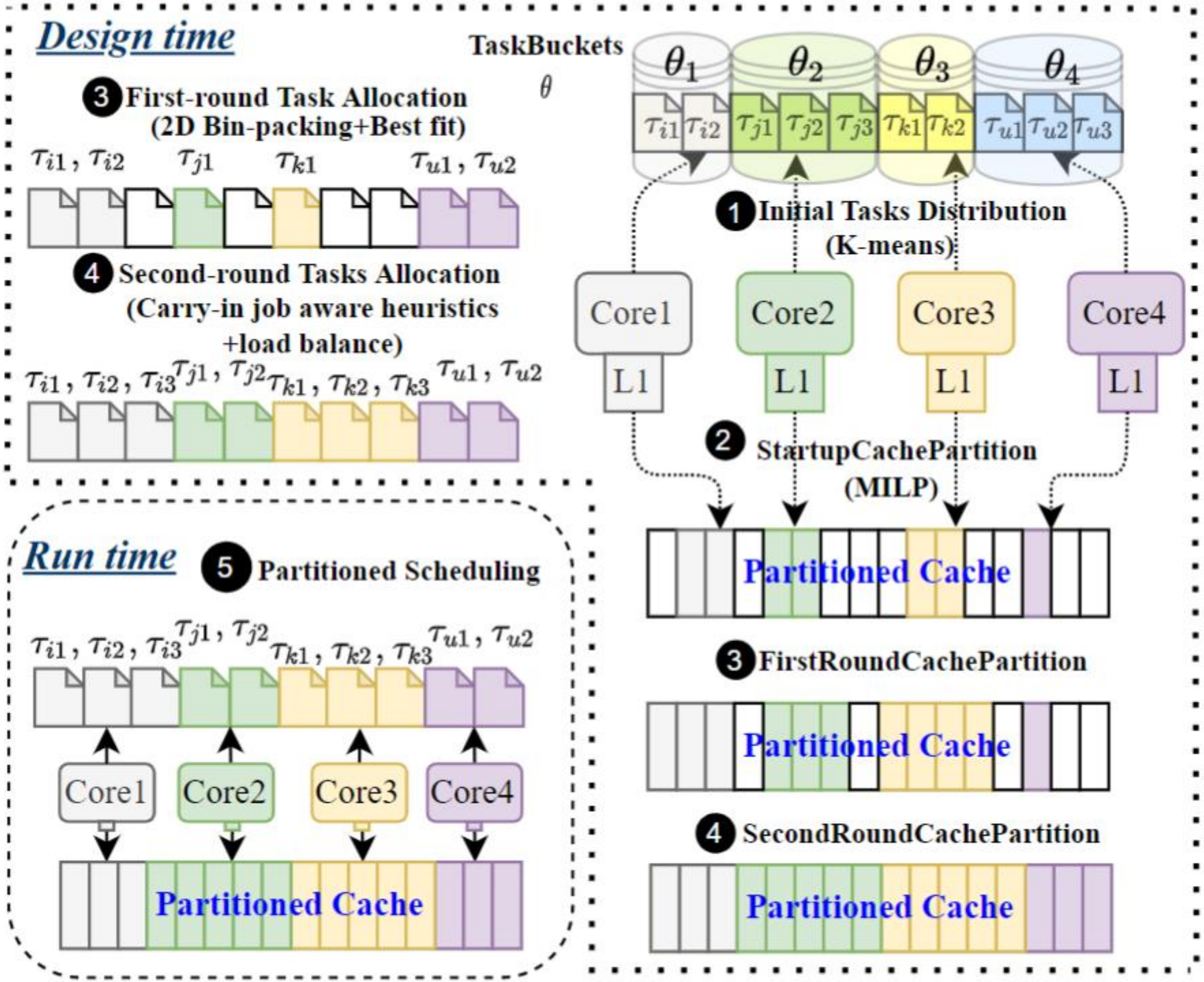


TCPS approach
θ_1
θ_2
θ_3
θ_4
τ_{i1}
τ_{j1} τ_{j3}
τ_{k1}
τ_{u1} τ_{u2}
❶ Initial Tasks Distribution (K-means)
Core1
τ_{i2}
τ_{j2}
τ_{k2}
τ_{u3}
❷ StartupCachePartition (MILP)
Partitioned Cache
❸ (2D Bin-packing+Best fit)
τ_{i1}, τ_{i2}
τ_{j1}
τ_{k1}
τ_{u1}, τ_{u2}
❸ FirstRoundCachePartition
Partitioned Cache

TCPS approach



TCPS approach

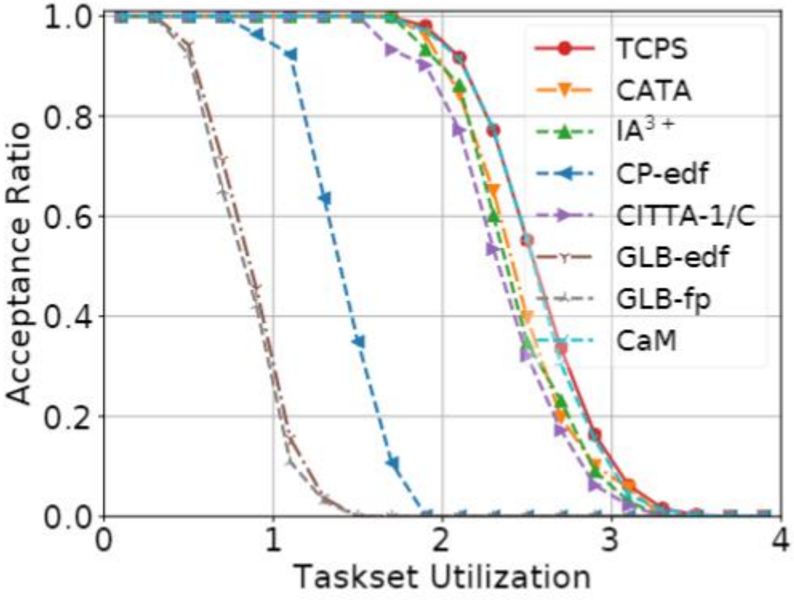


Experiments

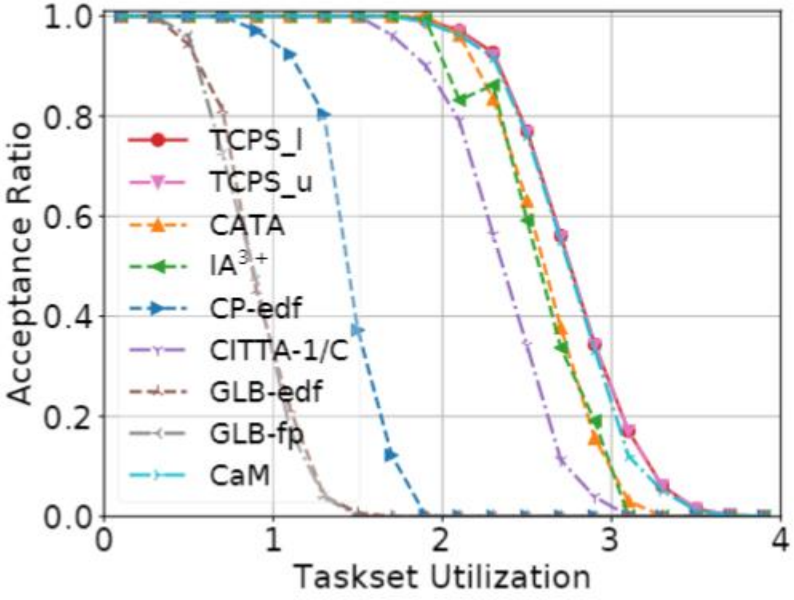
- Workload generation policy
  - The number of tasks ***n***
  - Total task utilization ***U***
  - WCET: measured by Heptane
  - Cache interference: measured by the extended Heptane
- In each experiment, generating totally 20000 taskset
- Evaluation Metric
  - Acceptance ratio: the number of schedulable tasksets divided by the total number of tasksets.
  - Cache Usage
  - Load balance

Experiments

Cache-sensitive tasks : their WCET drops by 40% or more when receiving 12.5% of cache space compared to the minimum allocation space.
cache-insensitive tasks: their WCET drops by 10% or less when receiving 50% cache space.



(a) m=4, n=10, *cache-sensitive* Γ



(b) m=4, n=10, *cache-insensitive* Γ

Experiments

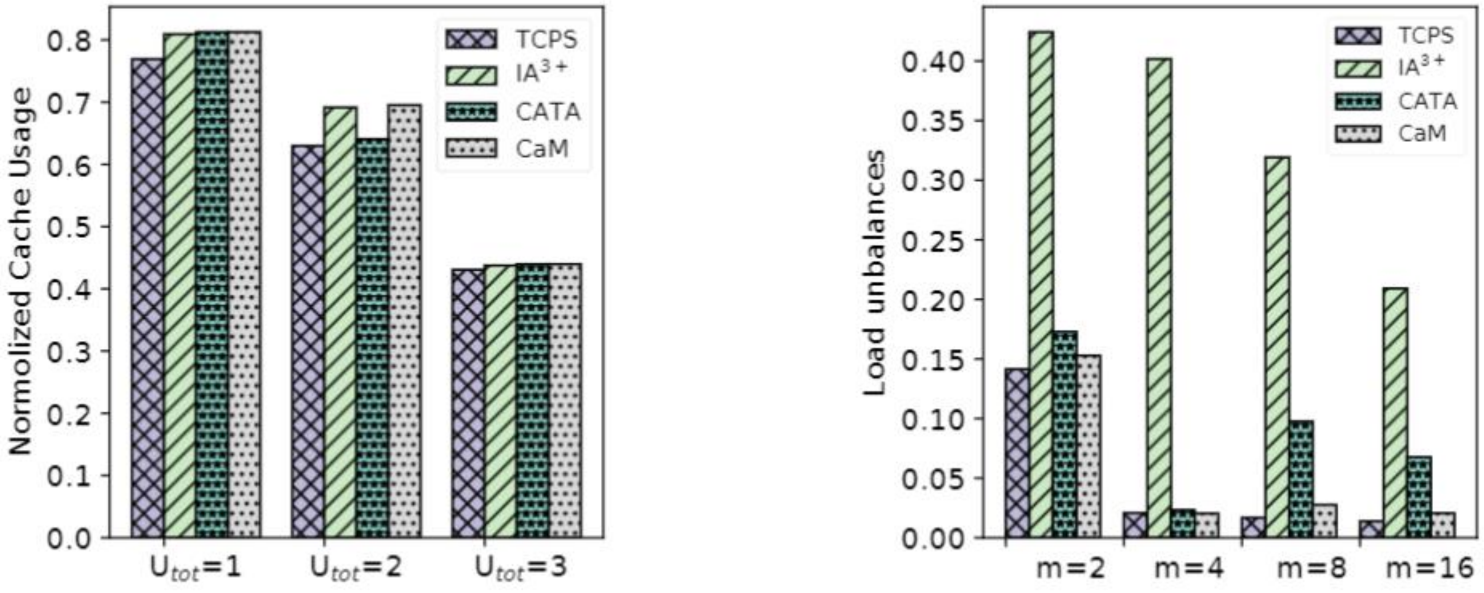
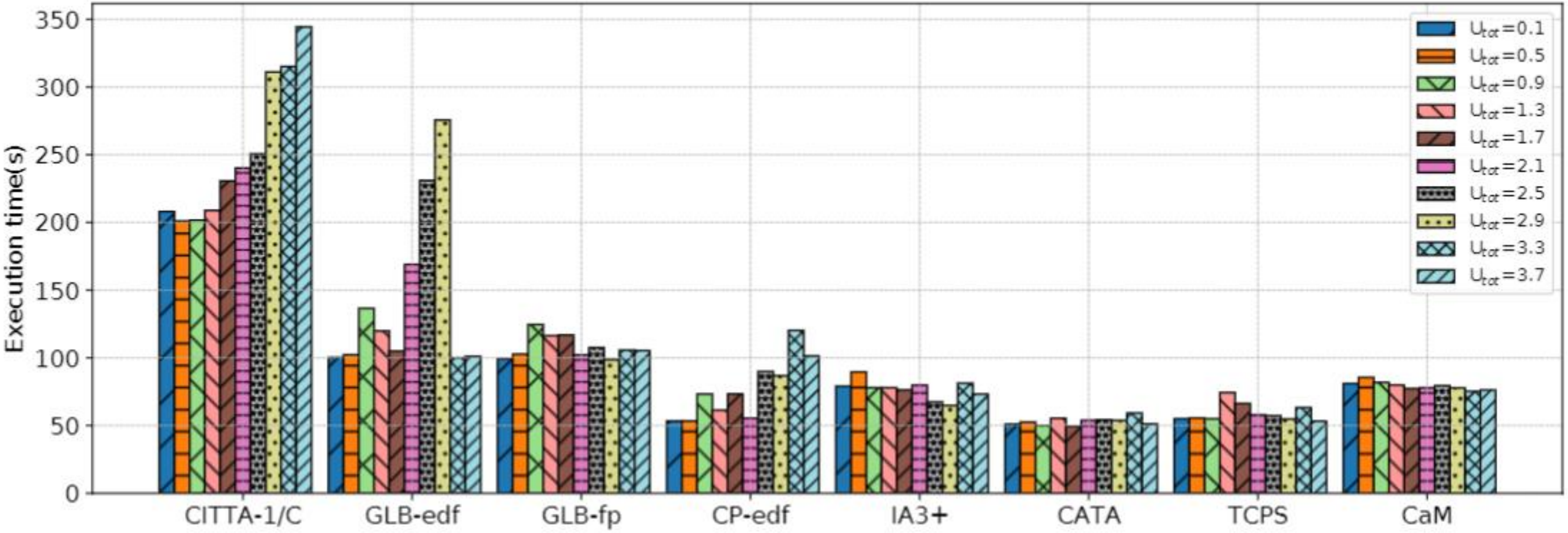


Fig. 6. The comparisons of cache usage and load unbalance for partitioned scheduling policies

Experiments

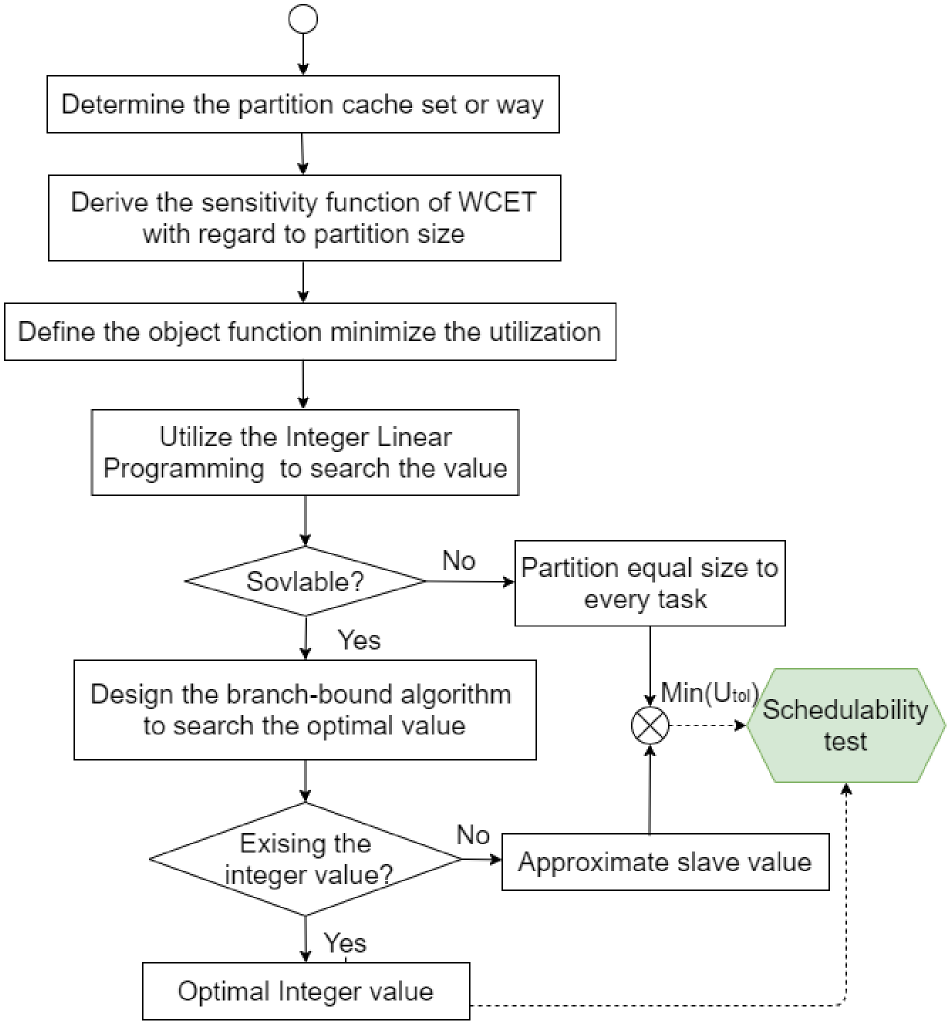


Conclusion

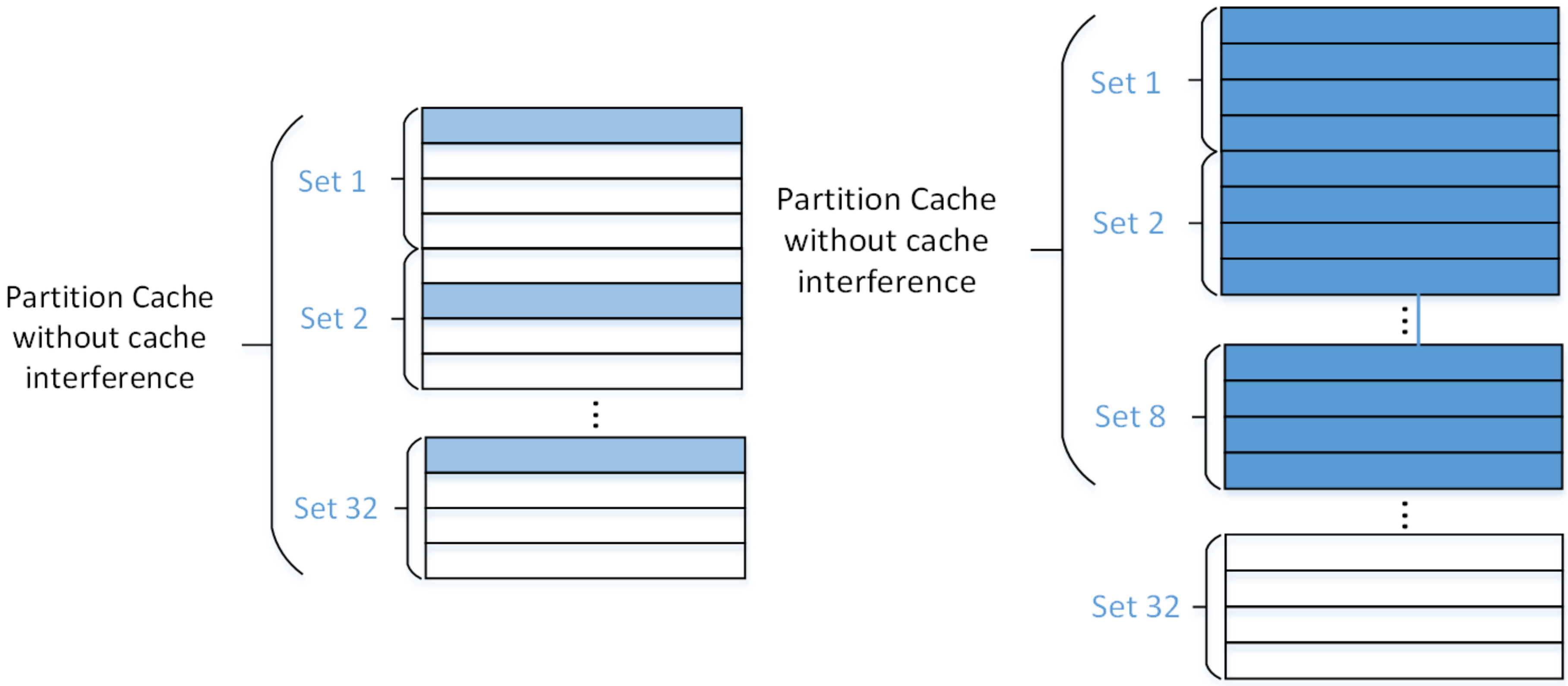
- TCPS: a non-preemptive partitioned scheduling algorithm called TCPS that aims at improving system schedulability through awareness of the cache sensitivity characteristics of applications.

- We have compared the schedulability performance of TCPS with a large range of state-of-the-art approaches, covering all possible scheduling/cache partitioning combinations.

The procedure of cache partition



Partitioning cache way or cache set?



Experiment configuration

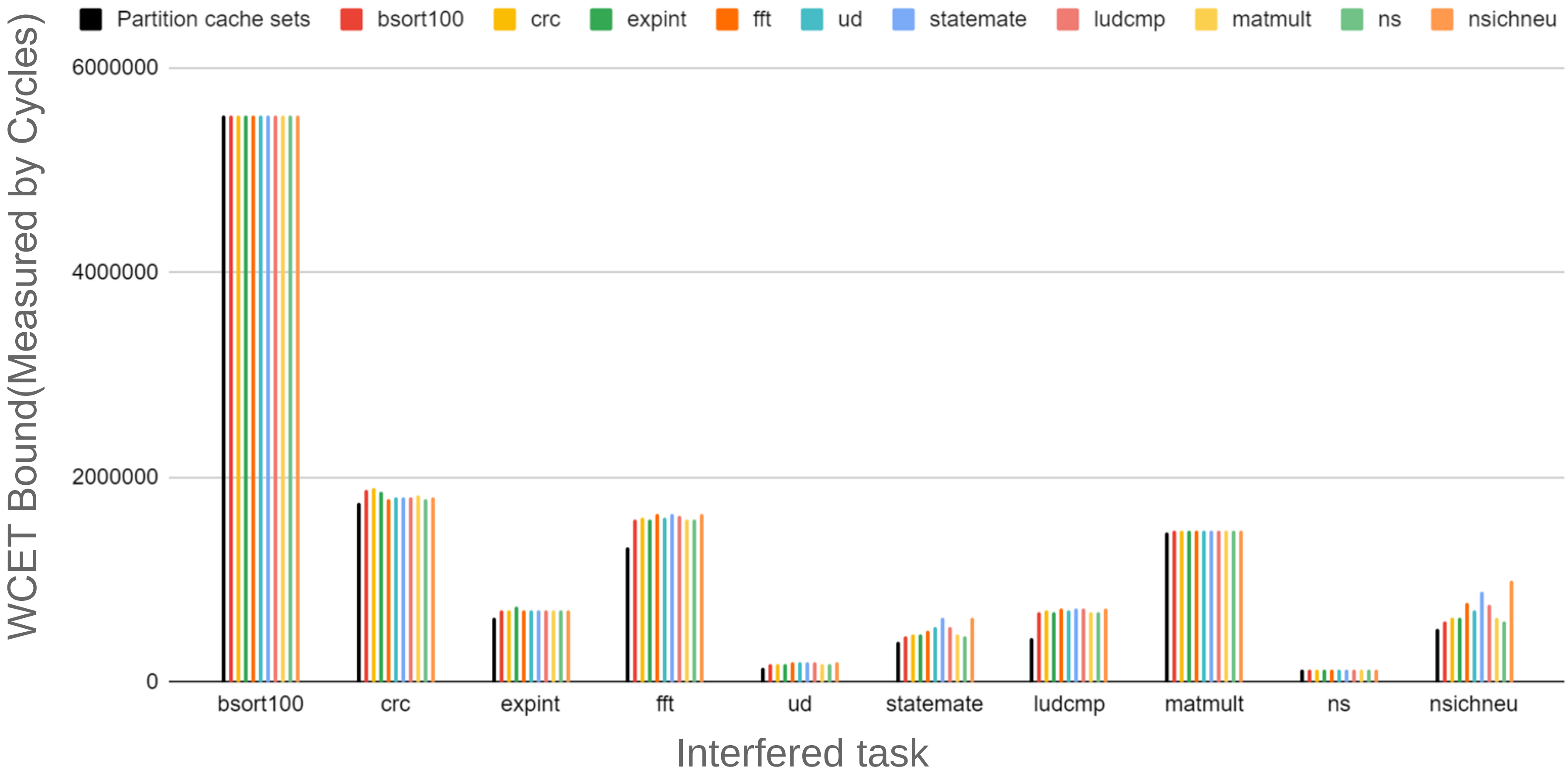
Execution platform: an embedded ARM processor with 4 cores.

Cache capacity: L1 private cache--cache line size 16B--512B
L2 shared cache--cache line size 32B--4KB

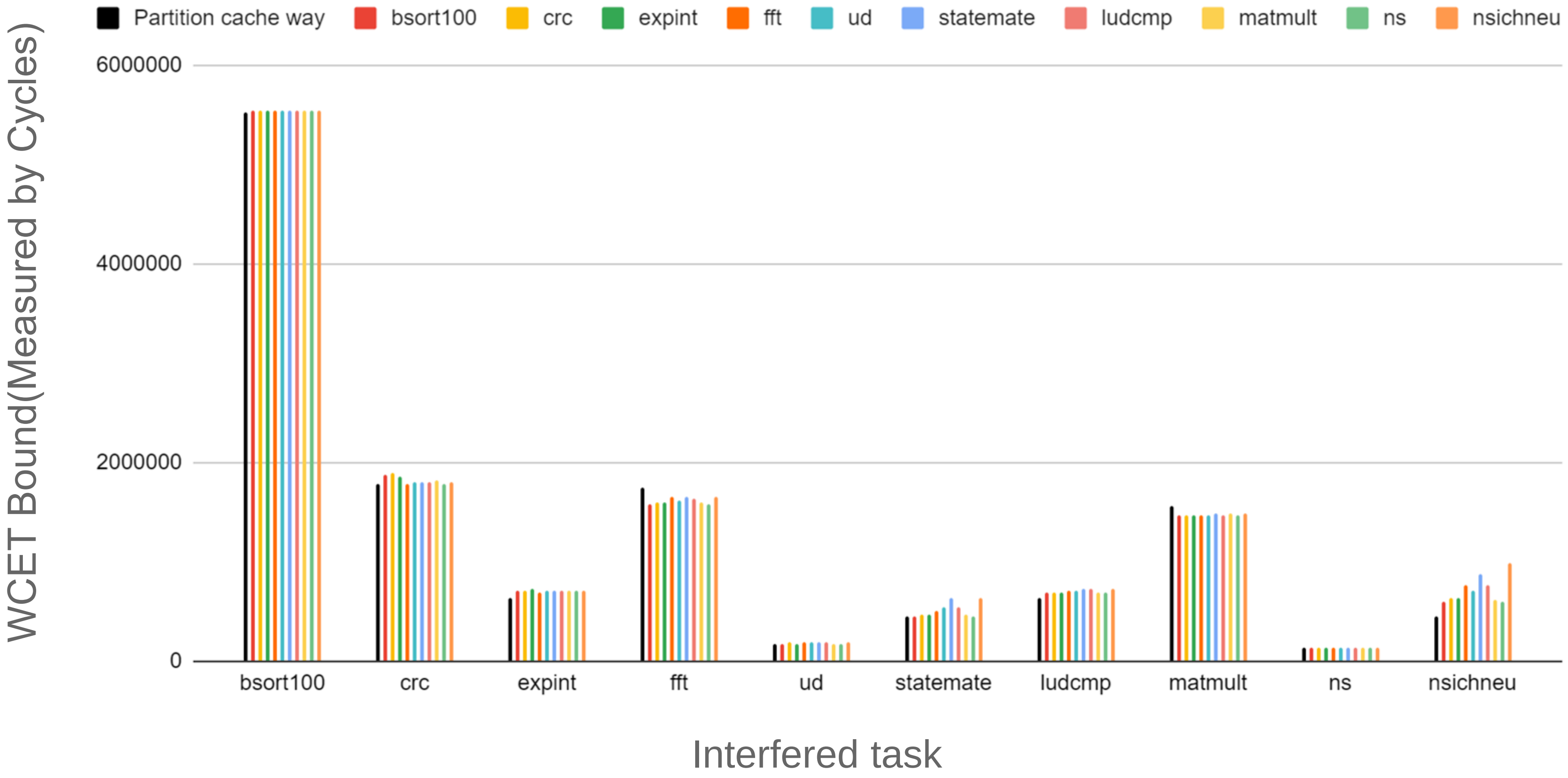
Benchmarks: Mälardalen WCET benchmarks

Name	Description
bsort100	bubblesort 100 Integer numbers
crc	Cyclic redundancy check computation
expint	Series expansion for computing an exponential
fft	Fast Fourier Transform
jfdctint	Fast Discrete Cosine Transform
ludcmp	Simultaneous Linear Equations by LU Decomposition
ns	Search in a multi-dimensional array
statemate	Automatically generated code by STARC
matmult	Product of two 20x20 integer matrixes
nsichneu	Simulate an extended Petri net

Partitioning at the granularity of cache set



Partitioning at the granularity of cache way



Experiment configuration

Experiment testbed: an embedded ARM processor with 4 cores.

Cache mode: Direct-mapped Cache

Cache capacity: L1 private cache--cache line size 16B--512B
L2 shared cache--cache line size **32B--8KB**

Benchmarks: Mälardalen WCET benchmarks

Name	Description
bsort100	bubblesort 100 Integer numbers
crc	Cyclic redundancy check computation
expint	Series expansion for computing an exponential
fft	Fast Fourier Transform
jfdctint	Fast Discrete Cosine Transform
ludcmp	Simultaneous Linear Equations by LU Decomposition
ns	Search in a multi-dimensional array
statemate	Automatically generated code by STARC
matmult	Product of two 20x20 integer matrixes
nsichneu	Simulate an extended Petri net

The cache partitioning algorithm

$$min\{U_1 + U_2 + ... + U_n\}$$

$$min\{\frac{1}{T_1} * c_1 + \frac{1}{T_2} * c_2, ..., + \frac{1}{T_n} * c_n\}$$

Constraints:

$$p_1 + p_2 + ... + p_n \leq n$$

$$c_1 * A_1 + p_1 * B_1 + k_1 = 0$$

$$c_2 * A_2 + p_2 * B_2 + k_2 = 0$$

$$...$$

$$c_n * A_n + p_n * B_n + k_n = 0$$