Washington University in St. Louis

Engineering

Elasticity of Workloads and Periods of Parallel Real-Time Tasks

26th International Conference on Real-Time Networks and Systems (RTNS '18) Poitiers/Futuroscope, France, October 10-12, 2018

[*]James Orr, [*]Chris Gill, [*]Kunal Agrawal,
[*]Sanjoy Baruah, [+]Christian Cianfarani,
[=]Phyllis Ang, and [+]Christopher Wong

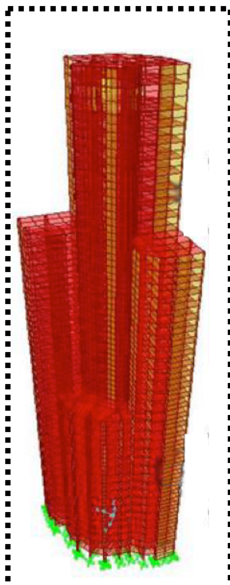
[*]Washington University in St. Louis
[+]Brown University
[=]University of Texas at Austin

Why Elastic Parallel Real-Time Tasks?

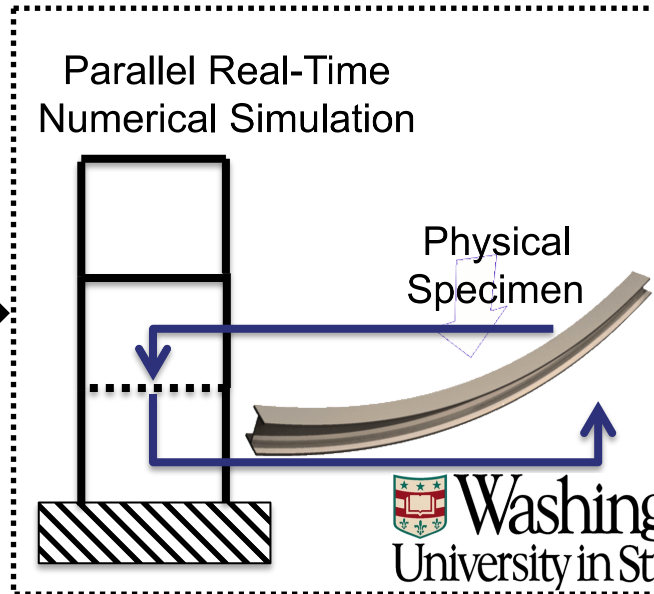
Need to "re-size" either workloads or periods adaptively,
e.g., in a 1000 degree-of-freedom simulation with **real-time** guarantees at periods down to millisecond time-scales, integrated **safely** with control, sensing, actuation

Empirical Evaluation

Theoretical Model



Real-Time Hybrid Simulation

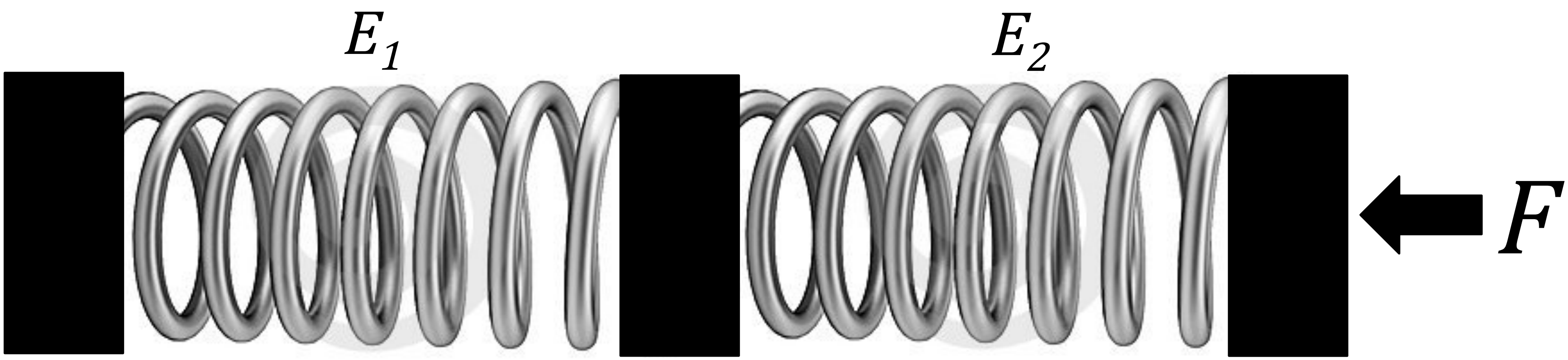


Limitations of the Current State of the Art

- Scheduling theory and concurrency platforms for parallel real-time tasks are mainly static
  - Assume regular release intervals and workloads
  - Limited adaptation to run-time conditions (mixed criticality)
- Elastic scheduling techniques don’t address tasks with both internal parallelism and variable workloads
  - Uniprocessor scheduling of sequential variable-period tasks
  - Elastic scheduling of parallel tasks with variable periods (only)

Elastic Scheduling of Sequential Real-Time Tasks

- Buttazzo et al. introduced the elastic scheduling model
  - » Increase tasks' periods to compresses utilizations (RTSS '98)
  - » Analogous to elastic compression of physical springs



  - » Model was also extended to consider blocking terms for critical sections accessed via the Stack Resource Policy (IEEE ToC '02)

Elastic Scheduling as Constrained Optimization

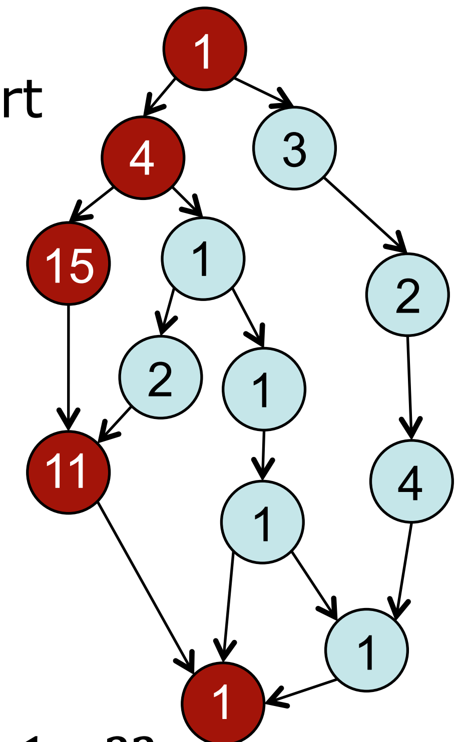
- Chantem et al. defined this as an optimization problem
  - » Minimize a weighted sum of squares of the differences between the chosen utilization for each task and its maximum utilization
  - » Subject to utilizations being between minimum and maximum values and the sum not exceeding the available utilization

$$\textbf{minimize} \sum_{i=1}^{n} \frac{1}{E_i} \left(U_i^{(max)} - U_i \right)^2$$

$$\textbf{such that } \forall_i \left(U_i^{(min)} \le U_i \le U_i^{(max)} \right) \wedge U_d \ge \sum_{i=1}^{n} U_i$$

Key Features of Parallel Real-Time DAG Tasks

- DAG of subtasks and their dependences
  - Predecessor nodes finish before successors start
- Work (computation time) C_i
  - Sequential execution time on 1 core
- Span (critical path length) L_i
  - Least parallel execution time on ∞ cores
- Implicit deadline D_i equals period T_i
  - Task must finish execution before next release



$L_i = 1 + 4 + 15 + 11 + 1 = 32$

$C_i = L_i + 3 + 2 + 4 + 1 + 1 + 1 + 1 + 2 = 47$

Temporally Elastic Parallel Real-Time Tasks

- Federated Scheduling
  - » Utilization of task τ_i is the ratio of its work C_i to its period T_i
  - » Number of (dedicated) cores τ_i needs also considers its span L_i
  - » Schedulable if can dedicate sufficient cores for all tasks' needs
- Extending elastic scheduling to parallel real-time tasks
  - » Semantics of Buttazzo et al. model can be used directly
  - » However, Chantem et al. model offers a more efficient approach based on Federated Scheduling of parallel real-time tasks
  - » Paper submitted to a journal (currently under review)
    - J. Orr, C. Gill, K. Agrawal, J. Li, S. Baruah, "Semantics Preserving Elastic Scheduling for Parallel Real-Time Systems"

Supporting Temporal or Computational Elasticity

- Contributions of this paper
  - » Generalizations of algorithm and task model from LITES paper to support either computational or temporal elasticity
  - » Empirical evaluations to gauge overheads, elastic equivalence
- Temporally elastic tasks
  - » Minimum inter-arrival time (period) can be varied elastically
  - » Task's span and work are fixed
- Computationally elastic tasks
  - » Sum of subtask execution times (work) can be varied elastically
  - » Task's span and period are fixed

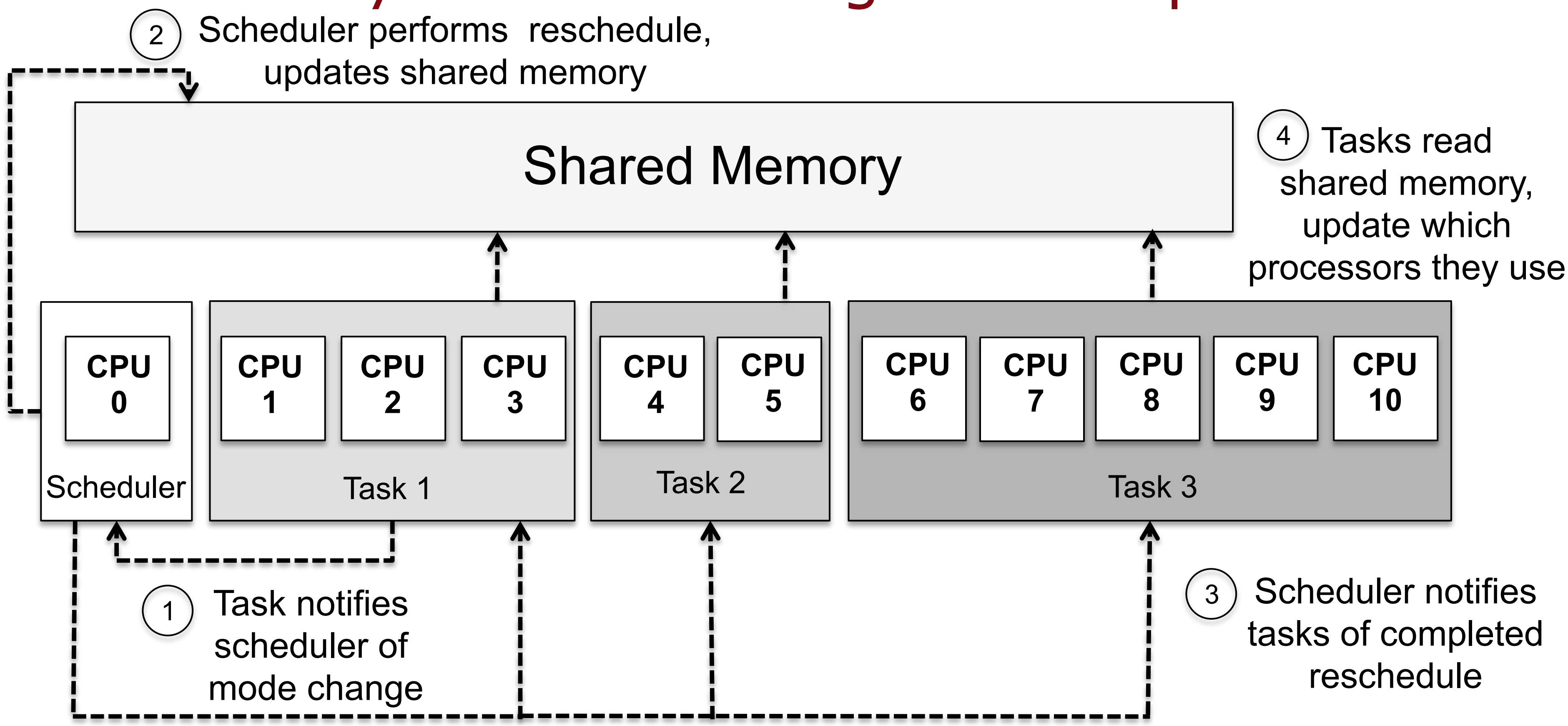
Elastic Compression of Parallel Real-Time Tasks

$$\textbf{minimize} \sum_{i=1}^{n} \frac{1}{E_i}\left(U_i^{(max)} - U_i\right)^2$$

$$\textbf{such that } \forall_i \left(U_i^{(min)} \leq U_i \leq U_i^{(max)}\right) \wedge m \geq \sum_{i=1}^{n} \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil$$

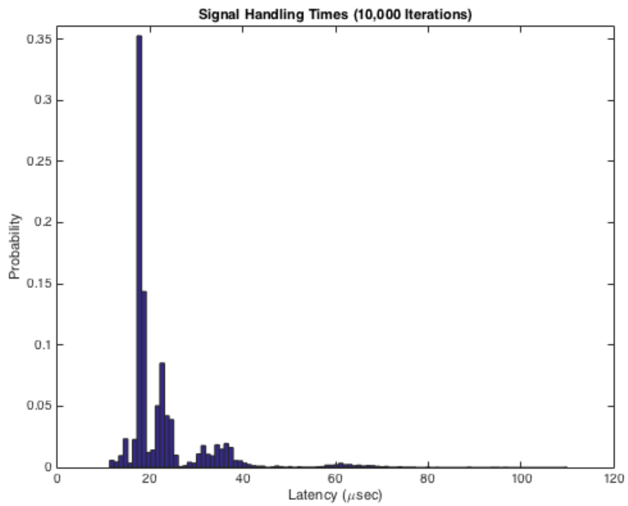
- Updates optimization from Chantem et al. (RTSS 2006)
  - Uses utilization definition for parallel real-time tasks
  - Allows either period or work to be compressed elastically
  - Checks schedulability under Federated Scheduling on m cores

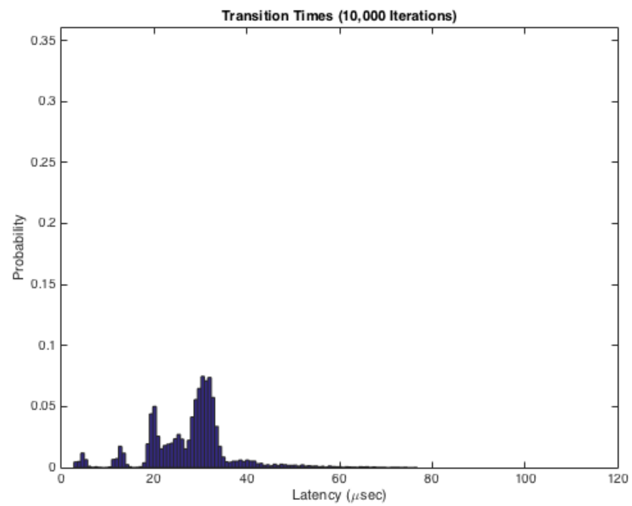
Concurrency Platform Design and Implementation



Adaptation Mechanism Overheads are Acceptable

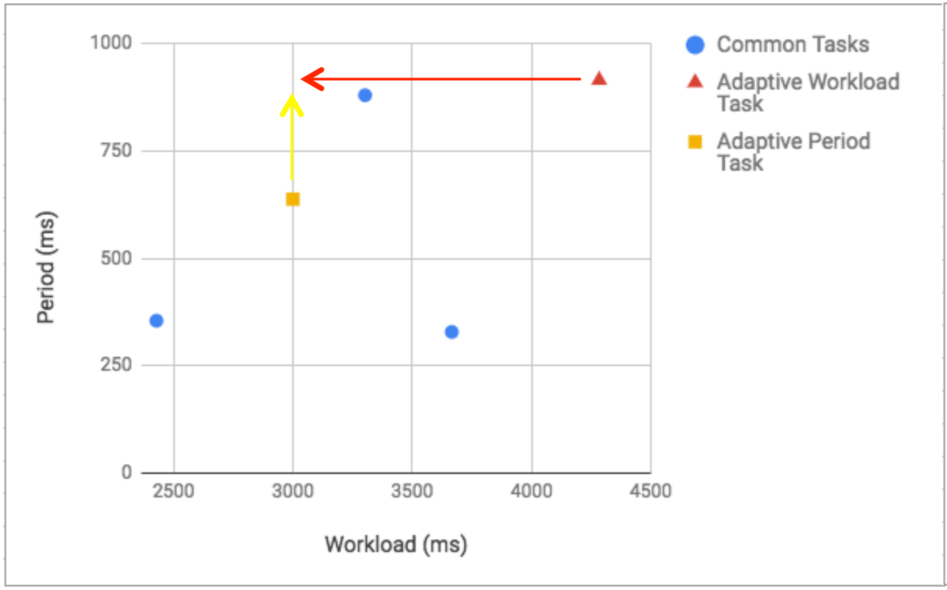
- Task notification via POSIX RT signals
  - » Ranged from 11.23 μsec to 110.03 μsec, often around 18 μsec
- Thread priority change (and possible core migration)
  - » Ranged from 2.67 μsec to 76.77 μsec, often around 30 μsec

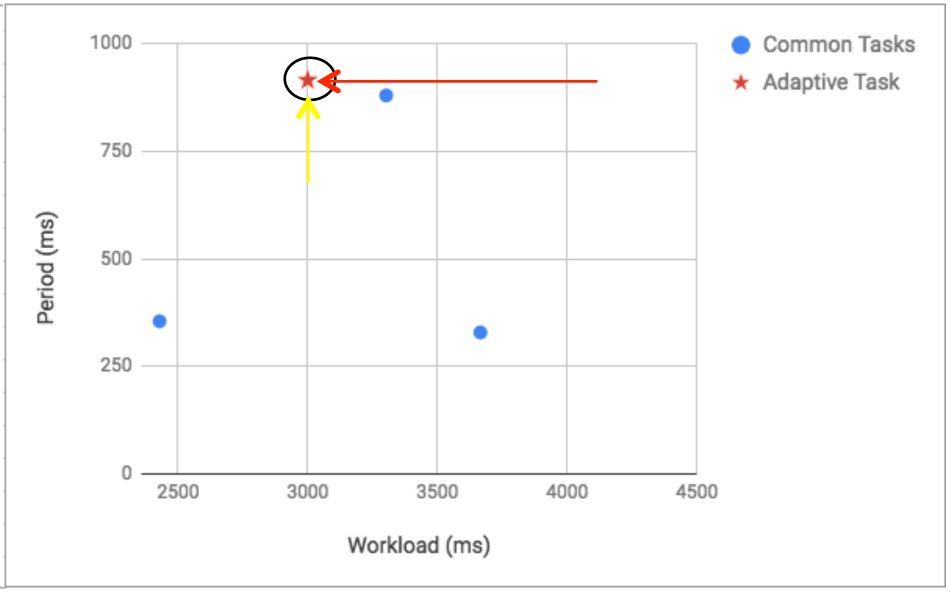




Evaluation Experiments Demonstrate Equivalence

- Experiments compared varying a task's D_i vs. its C_i
- Comparable tasks compressed to the same utilization
  - » Temporally vs. computationally elastic tasks reached same point





Conclusions and Future Work

- Contributions of this research
  - Scheduling of ***computationally elastic*** parallel real-time tasks
  - Equivalence of utilization compression when tasks are computationally vs. temporally elastic
  - Efficient implementation using OpenMP atop Linux

- Future research directions
  - Allowing a task's span and work and period to change at once
    - Schedulability analysis, optimization problem for elastic compression
    - Thread prioritization, core bindings, synchronization, notification
  - Elastic compression of tasks with only discrete utilization values

Thanks!

Washington University in St.Louis

James Orr
Chris Gill
Kunal Agrawal
Sanjoy Baruah

BROWN UNIVERSITY

Christian Cianfarani
Christopher Wong

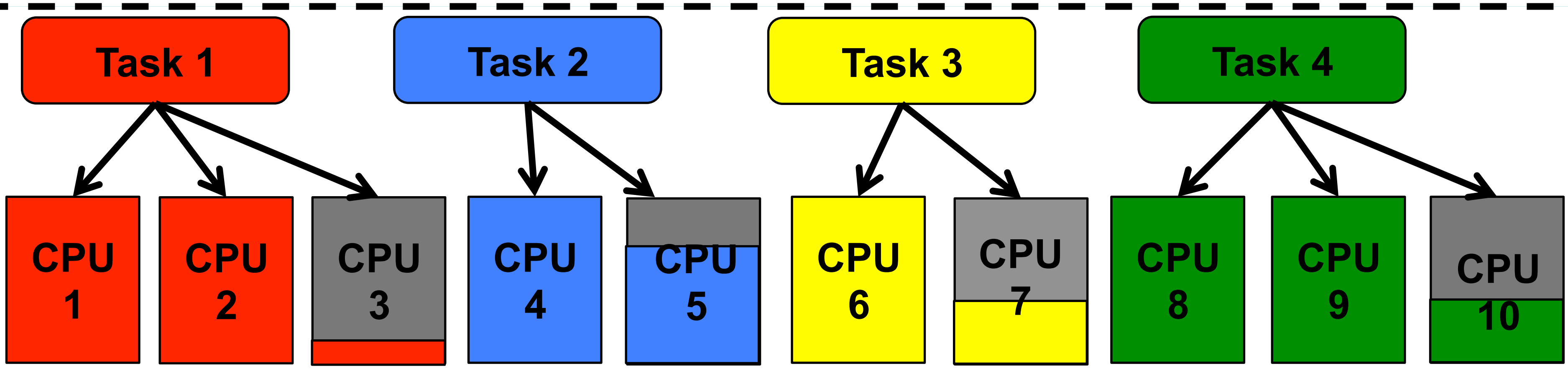
TEXAS
The University of Texas at Austin

Phyllis Ang

Work supported in part by NSF grant CCF-1337218 (XPS: FP)

Backup Slide: Federated Scheduling

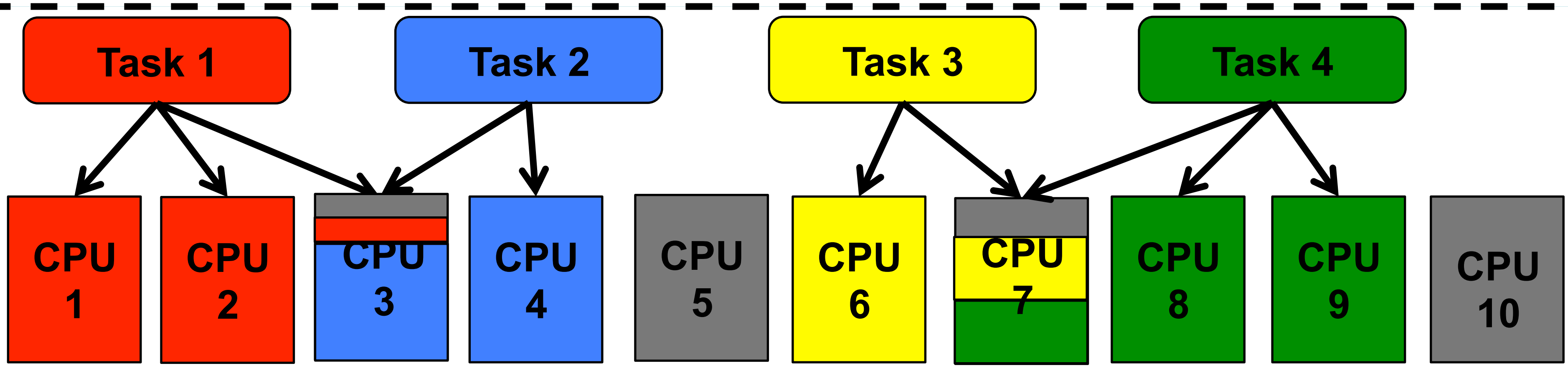
- In general a parallel task requires $\frac{C_i - L_i}{D_i - L_i}$ = $\mathbf{A_i} + \boldsymbol{\varepsilon_i}$ (A_i is integer, $0 \le \varepsilon_i < 1$) CPUs to guarantee completion
- Federated scheduling allocates $\left\lceil \frac{C_i - L_i}{D_i - L_i} \right\rceil = \begin{cases} A_i & \varepsilon_i = 0 \\ 1 + A_i & \varepsilon_i > 0 \end{cases}$ CPUs



J. Li et al., "Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks," 2014 26th Euromicro Conference on Real-Time Systems, Madrid, 2014, pp. 85-96.

Backup Slide: Semi-Federated Scheduling

- In general a parallel task requires $\frac{C_i - L_i}{D_i - L_i} = \mathbf{A_i} + \boldsymbol{\varepsilon_i}$ (A_i is integer, $0 \leq \varepsilon_i < 1$) CPUs to guarantee completion
- Semi-federated scheduling first allocates $\left\lfloor \frac{C_i - L_i}{D_i - L_i} \right\rfloor = \boldsymbol{A_i}$ CPUs
  - Remaining ε_i scheduled as sequential tasks on remaining CPUs (e.g. via partitioned EDF)



Jiang, Xu & Guan, Nan & Long, Xiang & Yi, Wang. (2017). Semi-Federated Scheduling of Parallel Real-Time Tasks on Multiprocessors. RTSS 2017.