CISTER - Research Center in
Real-Time & Embedded Computing Systems

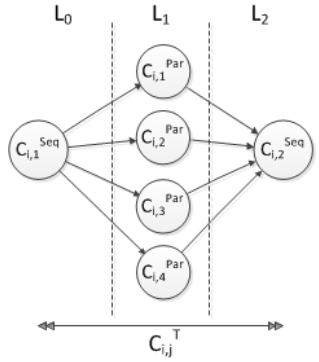
Response-Time Analysis of Synchronous Parallel Tasks in Multiprocessor Systems

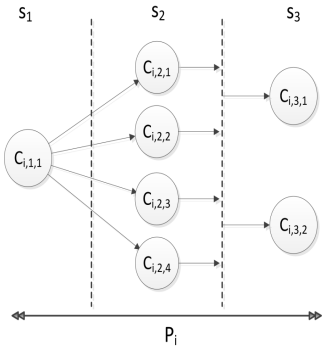
Cláudio Maia, Marko Bertogna, Luís Nogueira, and Luis Miguel Pinho

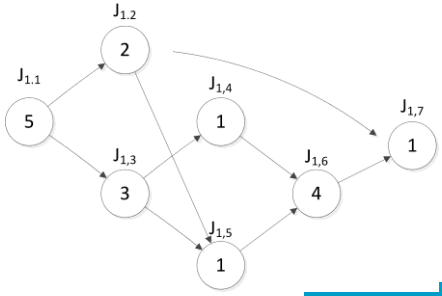
RTNS 2014

Motivation

- How to exploit parallelism provided by homo/heterogeneous platforms?
  - Different models
    - F/J model
    - Synchronous task model
    - DAG-based model
  - Different analysis techniques
    - Decomposition-based techniques
    - Resource-augmentation bound analysis
    - Response-time analysis







Contributions

- Schedulability analysis of synchronous parallel sporadic tasks
  - Existing RT analyses are based on g-EDF or partitioned approaches
  - We consider **global fixed priority** on identical multiprocessors
- Analyse the response-time behaviour of such tasks
  - Carry-out decomposition
  - Sliding window
- Provide a pseudo-polynomial algorithm to compute an upper bound on the RT of each parallel task
- Set of experiments showing that the proposed approach has good results when compared to state-of-the-art techniques

System Model

- System is composed of *m* identical processors with UMA

- Set of synchronous parallel sporadic tasks $\tau = \{\tau_1, ..., \tau_n\}$
- Each task τ_i is composed of a sequence of segments $s_i = \{\sigma_{i,1}, ..., \sigma_{i,si}\}$
- Each segment $\sigma_{i,j}$ is composed of a set of independent $m_{i,j}$ parallel jobs
- The number of parallel jobs of a segment can be greater than the number of cores

- Fixed priority scheduling
- Constrained deadline model $Di \leq Ti$

System Model

WCET of each segment $\sigma_{i,j}$

$$C_{i,j} = \sum_{k=1}^{m_{i,j}} C_{i,j,k}.$$

WCET of a task τ_i

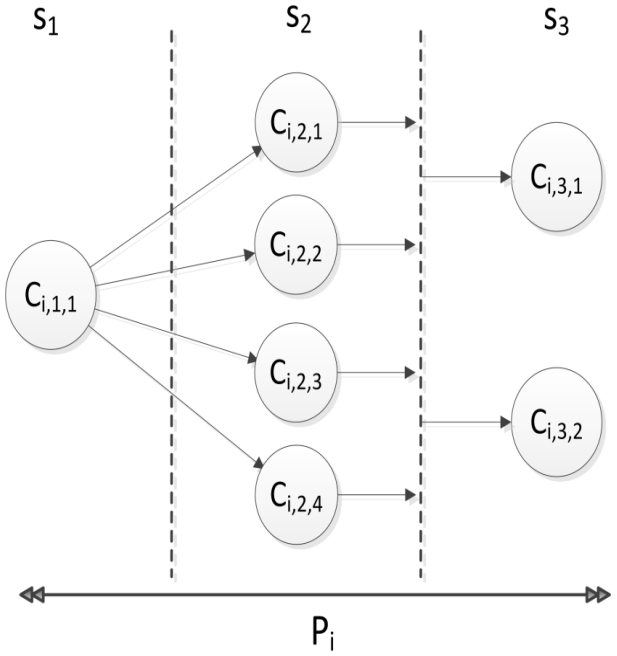
$$C_i = \sum_{j=1}^{s_i} C_{i,j}.$$

Minimum WCET

$$P_i = \sum_{j=1}^{s_i} P_{i,j},$$

WCET of the largest p-job(s) of segment $\sigma_{i,j}$

$$P_{i,j} = \max_{k=1}^{m_{i,j}} \{C_{i,j,k}\}.$$



System Model

- Response Time R_i
  - Maximum amount of time that elapses between the release time (r_i) of any job of task *i* and its completion time

- **Feasibility :** $U(\tau) \leq m$
  - For the synchronous task model this condition is necessary

- **Schedulability :** $Pi \leq Di$

Interference – Parallel Tasks

- Inter-task interference
  - Standard notion of interference in sequential models
- Intra-task interference
  - Self-interference of parallel jobs of the same task instance
  - Particular to parallel task systems
- Critical thread
  - The last thread to complete among the threads belonging to the same segment

Interference – Parallel Tasks

- Worst-case response time of a task
  - Sufficient to characterize the interference imposed to its critical threads
- Critical Interference in any interval of length L ($I_k(L)$)
  - Cumulative time in which a critical thread of a task *k* is ready to execute but it cannot due to the execution of higher priority parallel jobs
- Critical interference $I_{i,k}(L)$ by task *i* on task *k* in *L*
  - same def. as above but just considering task *i*

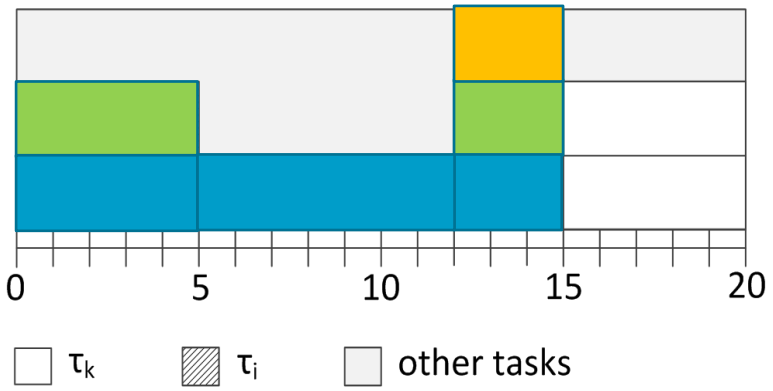
Interference – Parallel Tasks

- Theorem 1: $P_k + I_k(R_k^{ub}) \leq R_k^{ub}$.
  - Scheduling window spans [rk; rk +R_k^{ub}]
- Computing the exact interference imposed on a task is difficult
  - Upper bound each task contribution with WC workload executed in the considered window
- Each task *i* may contribute with different p-jobs at the same time

Interference – Parallel Tasks

- *at least p-depth critical interference*
  - To capture how many parallel jobs of *i* may **simultaneously** interfere with task *k*

- *at least p-depth critical interference* of *i* on *k*, in any interval of length L ($I_{i,k}^{p}(L)$)
  - total amount of time in which a critical thread of *k* is ready to execute but cannot execute while there are at least *p* threads of task *i* **simultaneously** executing in the system

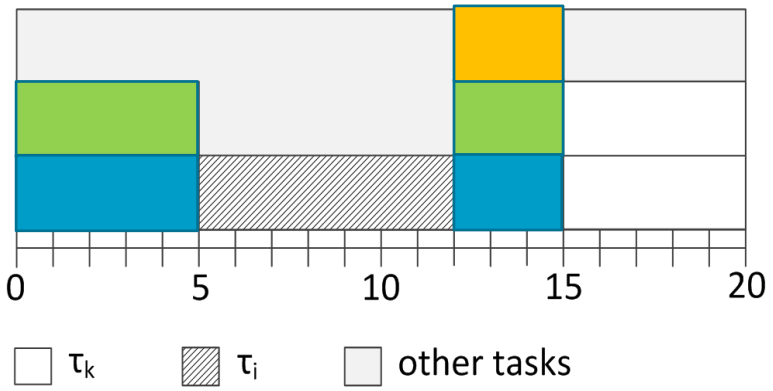


$I_{i,k}^{1}(L) = 15$

Interference – Parallel Tasks

- *at least p-depth critical interference*
  - To capture how many parallel jobs of *i* may simultaneously interfere with task *k*

- *at least p-depth critical interference* of *i* on *k*, in any interval of length L ($I_{i,k}^{p}(L)$)
  - total amount of time in which a critical thread of *k* is ready to execute but cannot execute while there are at least *p* threads of task *i* simultaneously executing in the system

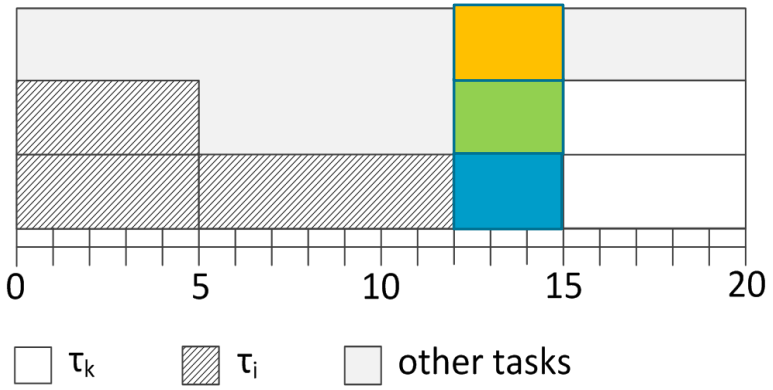


$I_{i,k}^{2}(L) = 8$

Interference – Parallel Tasks

- *at least p-depth critical interference*
  - To capture how many parallel jobs of *i* may simultaneously interfere with task *k*

- *at least p-depth critical interference* of *i* on *k*, in any interval of length L ($I_{i,k}^{p}(L)$)
  - total amount of time in which a critical thread of *k* is ready to execute but cannot execute while there are at least *p* threads of task *i* simultaneously executing in the system



$I_{i,k}^{3}(L) = 3$

Interference – Parallel Tasks

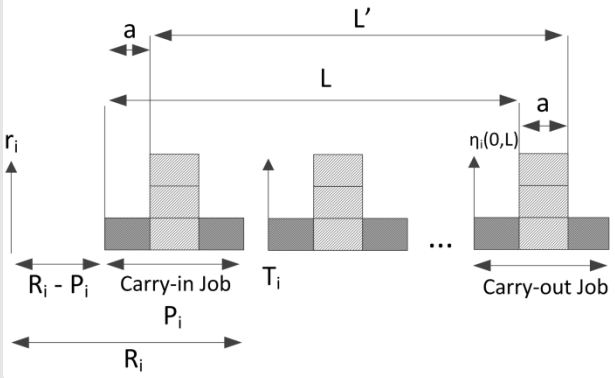
- **Lemma 2:** $I_k(L) = \frac{1}{m} \sum_{\forall \tau_i} \sum_{p=1}^{m} I_{i,k}^{p}(L).$
  - Task *k* interference is equivalent to the sum of the at least p-depth critical interferences of higher priority tasks
- **Theorem 2:** $\sum_{\forall \tau_i} \sum_{p=1}^{m} \min\left(I_{i,k}^{p}(R_k^{ub}), R_k^{ub} - P_k + 1\right) < m(R_k^{ub} - P_k + 1)$
  - If it holds then the WCRT is upper-bounded by R_k^{ub}

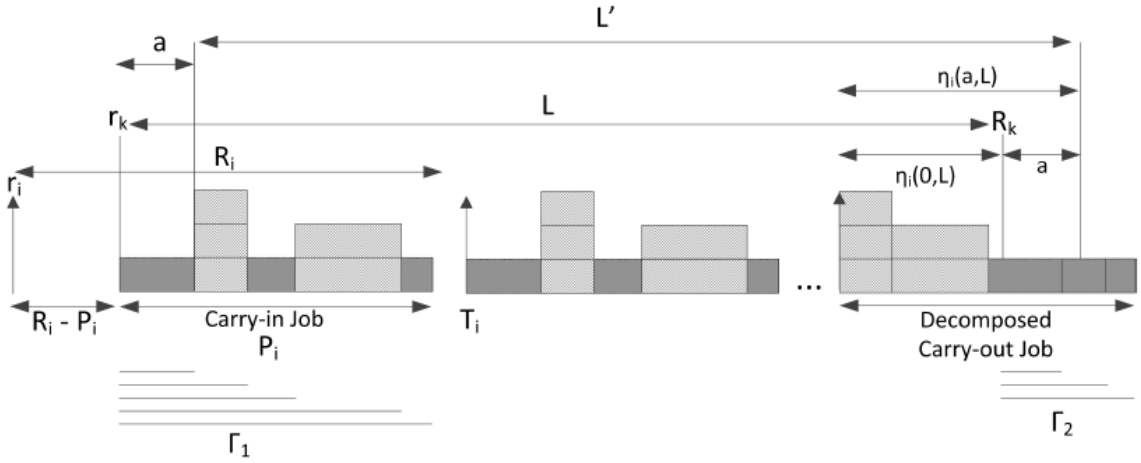
Response Time Analysis

- Provide an upper bound on the execution time of an interfering task *i* in a window of length *L* ($[r_k, r_k+L]$)
  - Contributions of **Carry-in** job, **carry-out** job, **body** jobs

- Densest possible packing:
  - A job starts executing at the beginning of the window of interest, and completes as close as possible to its response time
  - All subsequent jobs of *i* are executed as soon as possible after being released

Sliding Window



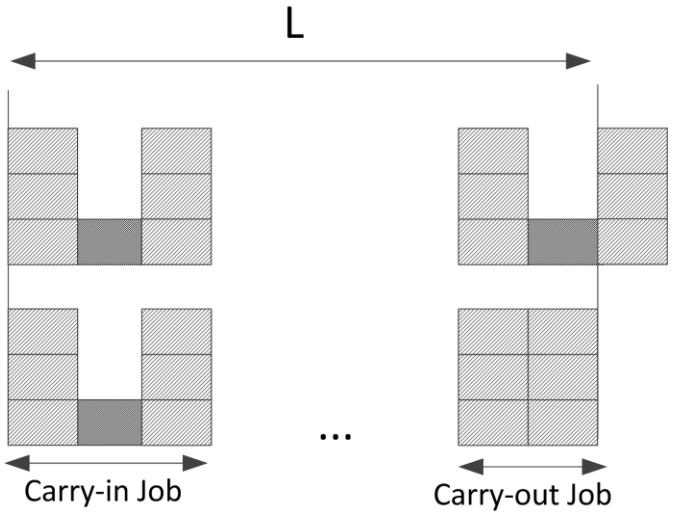


- Parallel structure affects the densest possible packing of jobs
  - Check all different alignments of the window *L*
  - Use the points coinciding with a segment boundary
- Worst-case workload of a task *i* in window *L*
  - Max. workload of *i* over all possible configurations

Decomposing the carry-out

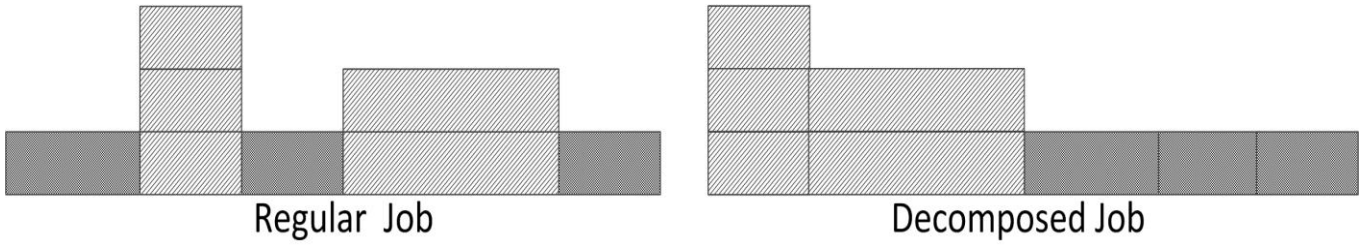
- Predictability and robustness of the schedulability test
  - Consider all possible execution requirements of the given tasks
    - Some segment requires less than C_{ij} time-units or
    - A task may skip some of the segments.

- Predictability problem exists when a larger workload enters the considered window if the carry-out skips some segment



Decomposing the carry-out

- Re-alignment of parallel segments
  - Shift segments with higher parallelism to the beginning
- Replace the original carry-out job by a decomposed job



Workload

- Derive an upper bound on W_i^p
  - At least p-depth contributions of **carry-in**, **body** and **decomposed carry-out** of each task *i* in the worst-case scenario

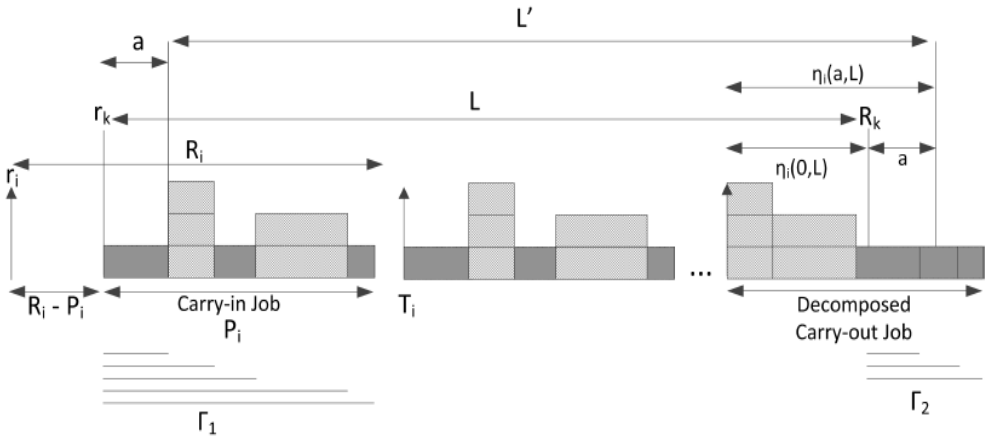
$$\widehat{W}_i^p(L,a) = f_i^p(\alpha_i(a,L)) + b_i^p(L) + \tilde{g}_i^p(\eta_i(a,L)),$$

$$\widehat{W}_i^p(L) = \max_{a \in \Gamma_1 \cup \tilde{\Gamma}_2} \{W_i^p(L,a)\},$$

Inter-task workload

$$\widehat{W}_k^p = \sum_{\forall j : m_{k,j} \geq p+1} P_{k,j},$$

Intra-task workload



Schedulability Condition

- **Theorem 3**: An upper bound on the WCRT of a task k can be derived by the fixed-point iteration, starting with $R^{ub}_k = P_k$

$$R_k^{ub} \leftarrow P_k + \left\lfloor \frac{1}{m} \left(\sum_{\forall i<k} \sum_{p=1}^{m_i} \min\left(\widehat{W}_i^p(R_k^{ub}), R_k^{ub} - P_k + 1\right) + \sum_{p=1}^{m_k} \min\left(\widehat{W}_k^p, R_k^{ub} - P_k + 1\right) \right) \right\rfloor .$$

Evaluation

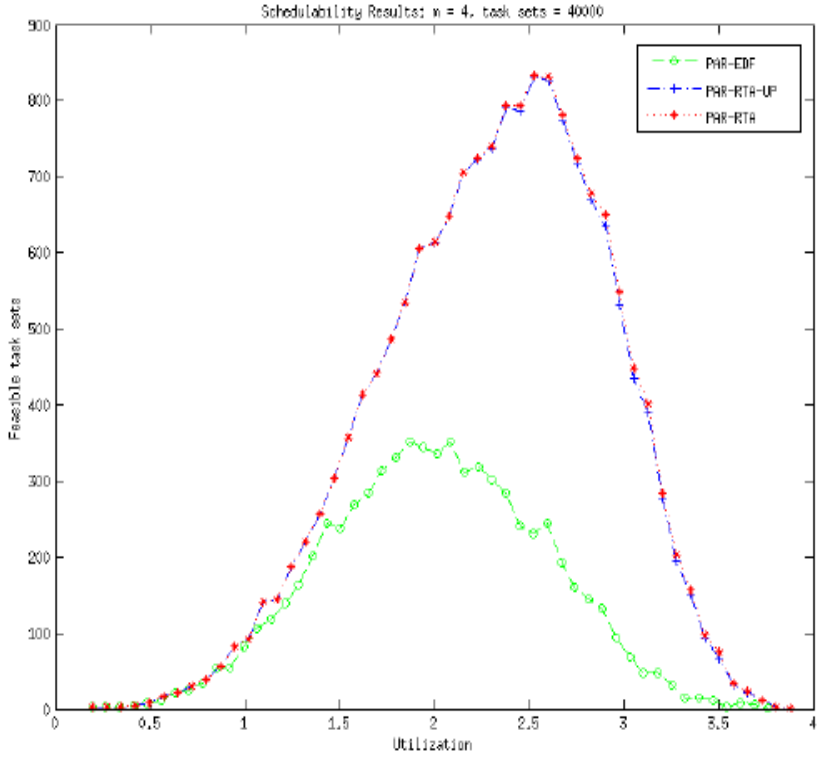
- Simulation Environment
  - Iteratively add new tasks to a task set and form new task sets until U exceeds m
  - 40,000 task sets are generated
  - Parameters are randomly generated following a uniform distribution over certain intervals

- We compare the number of schedulable task sets detected by our analysis (PAR-RTA) with (PAR-EDF)[1]

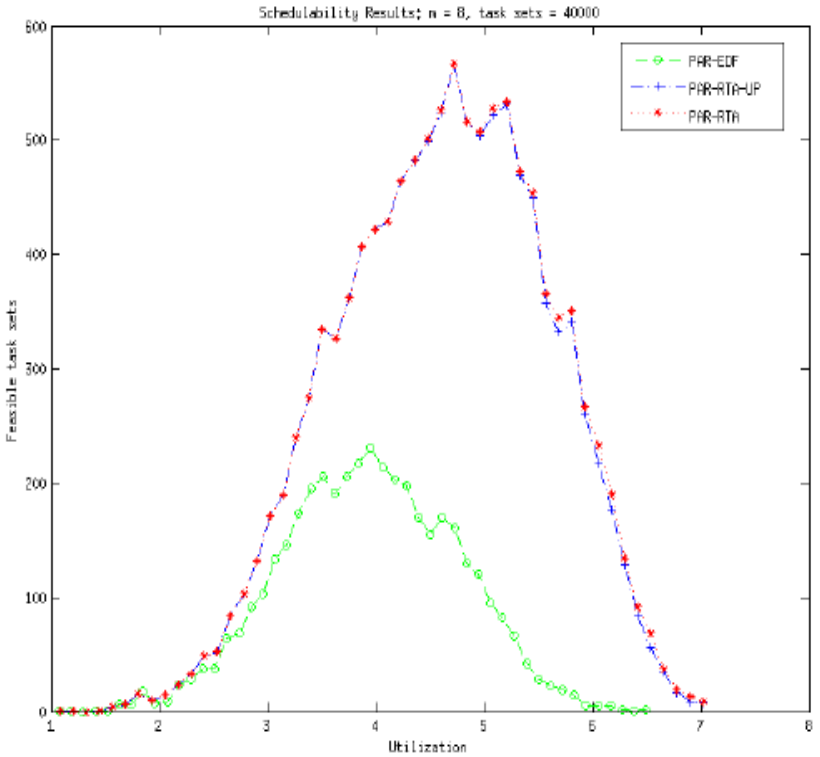
- We show the performance of a faster method (PAR-RTA-UP) that uses the workload upper bound (Eq. 15)

[1]Proposed in [7] by Chwa et al.

Evaluation



(a) m = 4



(b) m = 8

- Our approach detects 230% more schedulable task sets
- Faster method using the simplified upper bound
  - Performance very similar to the complete method (within 1%)

Conclusion

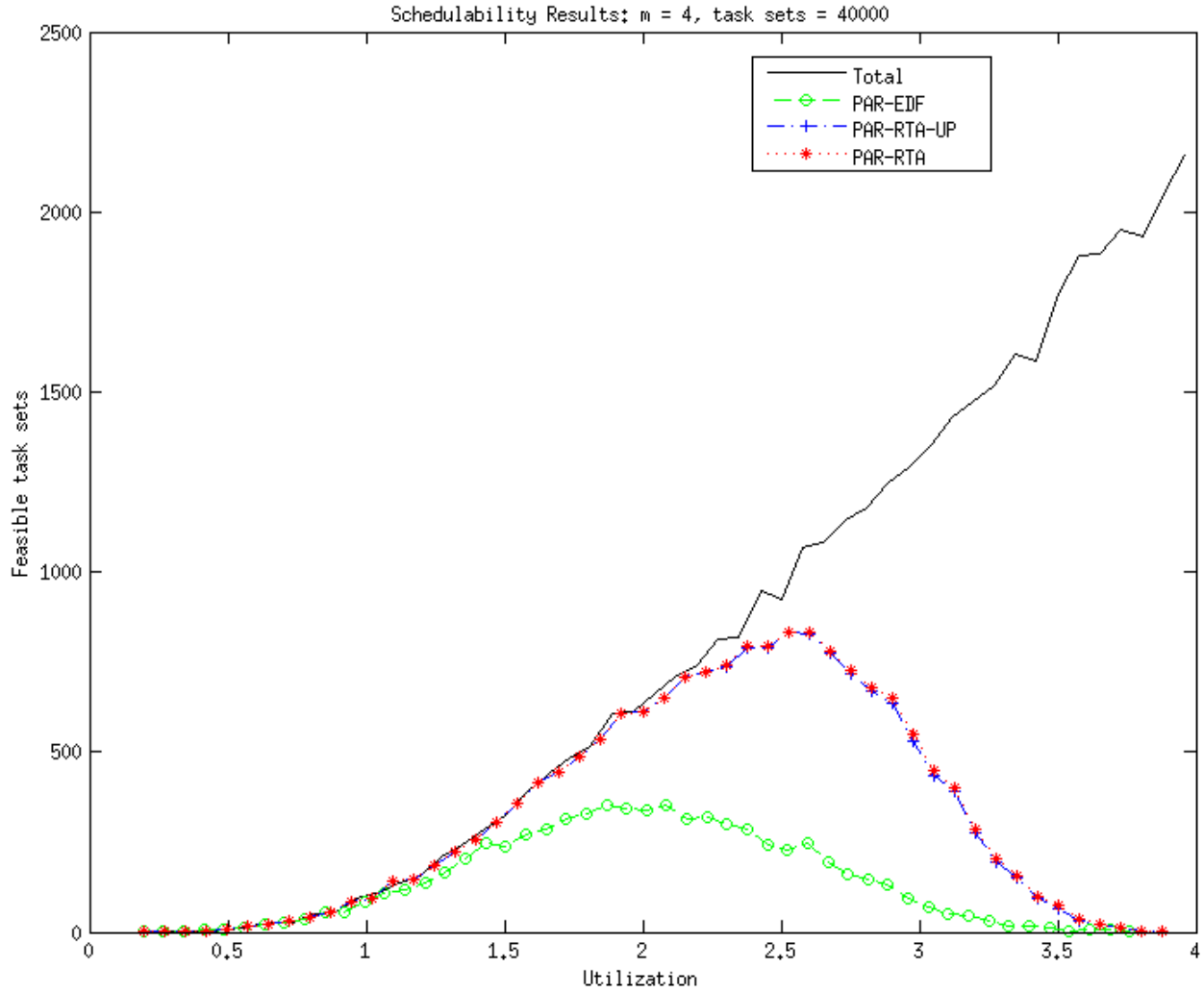
- Recent shift to multi/manycore architectures
  - Brings the need to new real-time parallel task models
  - Schedulability analisys techniques for such models

- We propose a response-time analysis for work-conserving schedulers
  - Worst-case scenarios leading to the largest possible interference
- Two tests are presented:
  - Sliding window technique and carry-out decomposition
  - a simplified test with a smaller computational complexity and performance
  - Both tests significantly improve over the state of the art

- Extend the response-time analysis framework presented in this paper to other task models

Questions

Extra results



Overview

- Response-time analysis of synchronous parallel tasks in identical multiprocessors

- Built upon the work of Chwa et al. ([7])
  - Derive tighter upper-bounds on the workload
  - Compute the response-time upper bounds similarly to Bertogna and Cirinei ([8]) for sequential task sets

Interference - Sequential

- Interference a task *k* suffers in an interval of length *L* ($I_k(L)$)
  - Interference of a higher priority task *i* over task *k* in L is denoted as $I_{i,k}(L)$

- Interference a task suffers cannot be greater than the total workload of higher priority jobs

Interference – Parallel Tasks

- Each task *i* may contribute with different p-jobs at the same time to the individual interference on a task *k*
  - when *i* = *k*, the critical interference $I_{k,k}(L)$, may include intra-task interference

- **Lemma 1:** $$I_k(L) = \frac{1}{m} \sum_{\forall \tau_i} I_{i,k}(L).$$
  - For any work-conserving algorithm, all *m* cores are busy when a critical thread of *k* is interfered

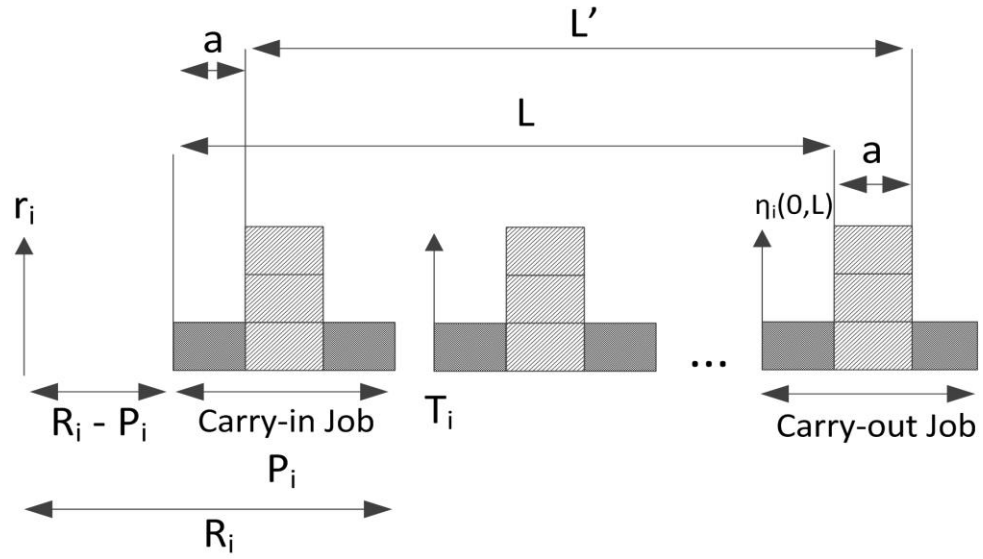
Response Time Analysis

- Finding the critical interference is a difficult problem for multiprocessor systems

- A common approach is to use upper bounds
  - e.g. upper bound the interference of task *i* in a window of length *L* is given by the maximum workload *i* can execute within the considered window
    - Which is also difficult to obtain

- **Alternative**: consider pessimistic scenarios
  - workload in a given window cannot be smaller than the worst-case situation

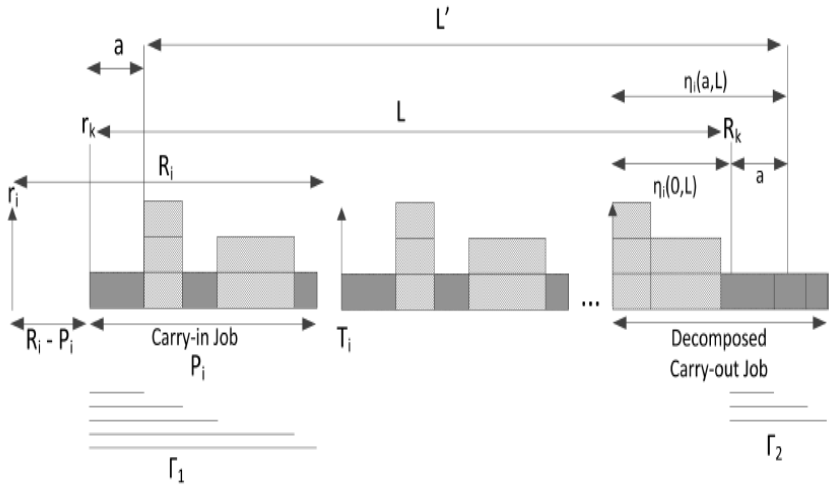
Window of Interest



- Densest possible packing:
  - A job starts executing at the beginning of the window of interest, and completes as close as possible to its response time
  - All subsequent jobs of *i* are executed as soon as possible after being released

Sliding Window

- Worst-case workload of a task *i* in window *L*
  - Max. workload of *i* over all possible configurations in which the window is shifted right from the original configuration
- Consider the meaningful offsets in each scenario



$$\Gamma_1 \doteq \left\{ \sum_{x=1}^{j} P_{i,x} \leq P_i - \eta_i(0,L), \forall j \in [1, s_i] \right\}.$$

$$\Gamma_2 \doteq \left\{ \max\left(0, \sum_{x=1}^{j} P_{i,x} - \eta_i(0,L)\right), \forall j \in [1, s_i] \right\}.$$

Workload within Window

Number of body Jobs

$$\beta_i(L) = \left\lfloor \frac{L + R_i - P_i}{T_i} \right\rfloor - 1.$$

p-depth workload of body jobs

$$b_i^p(L) = \beta_i(L) \sum_{\forall j : m_{i,j} \geq p} P_{i,j}.$$

$$f_i^p(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ \sum_{j=h:m_{i,j}\geq p}^{s_i} P_{i,j} + (x - \sum_{j=h}^{s_i} P_{i,j}), & \text{if } 0 < x \leq P_i \text{ and } m_{i,h-1} \geq p \\ \sum_{j=h:m_{i,j}\geq p}^{s_i} P_{i,j}, & \text{if } 0 < x \leq P_i \text{ and } m_{i,h-1} < p \\ \sum_{\forall j:m_{i,j}\geq p} P_{i,j}, & \text{otherwise,} \end{cases}$$

Carry-in contribution

Decomposed carry-out length

$$g_i^p(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ \sum_{j=1:m_{i,j}\geq p}^{z} P_{i,j} + (x - \sum_{j=1}^{z} P_{i,j}), & \text{if } 0 < x \leq P_i \text{ and } m_{i,z+1} \geq p \\ \sum_{j=1:m_{i,j}\geq p}^{z} P_{i,j}, & \text{if } 0 < x \leq P_i \text{ and } m_{i,z+1} < p \\ \sum_{\forall j:m_{i,j}>p} P_{i,j}, & \text{otherwise,} \end{cases}$$

Carry-in length

$$\alpha_i(a, L) = L - \eta_i(a, L) - \beta_i(L) T_i.$$

Future Work

- Refine the analysis by reducing the number of carry-in instances
  - as [22] *Guan et al.* did for the sequential case

- Extend the response-time analysis framework presented in this paper to other task models
  - DAG-based, arbitrary deadlines, etc.
  - Scheduling policies, including global EDF