INSTITUTE OF COMMUNICATION, INFORMATION AND PERCEPTION TECHNOLOGIES
Scuola Superiore Sant'Anna

Retis Real-Time Systems Laboratory

Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems

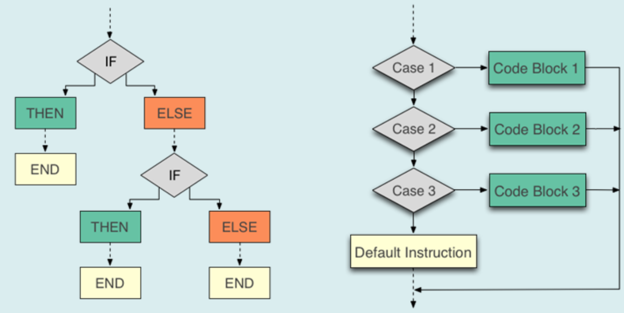
Alessandra Melani

What does it mean?

- « Response-time analysis » ✓
- « conditional »
- « DAG tasks »
- « multiprocessor systems » ✓

What does it mean?

- « Response-time
- **« conditional »**
- « DAG tasks »
- « multiprocessor



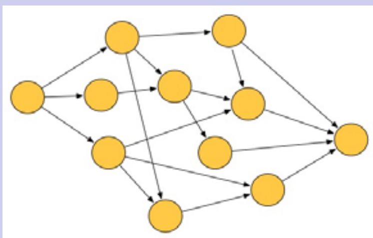
If-then-else statements

Switch statements

What does it mean?

- « Response-time an
- « conditional »
- **« DAG tasks »**
- « multiprocessor sys

DAG: Directed Acyclic Graph



In other words

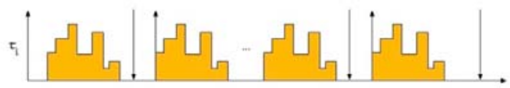
- We will analyze a **multiprocessor** real-time systems…
- … by means of a **schedulability test** based on **response-time analysis**
- … assuming **Global Fixed Priority** or **Global EDF** scheduling policies
- … and assuming a **parallel task model** (i.e., a task is modelled as a **Directed Acyclic Graph - DAG**)

Parallel task models

- Many parallel programming models have been proposed to support parallel computation on multiprocessor platforms (e.g., OpenMP, Cilk, Intel TBB)

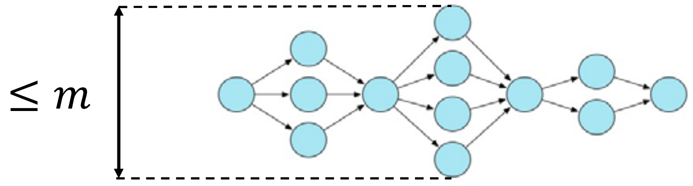
Early real-time scheduling models: each recurrent task is completely sequential

Recently, more expressive execution models allow exploitation of parallelism within tasks



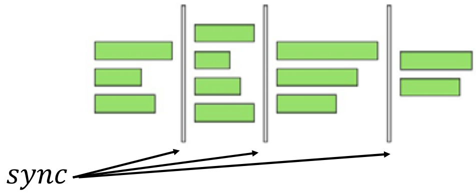
Fork-join

- Each task is an alternating sequence of sequential and parallel segments
- Every parallel segment has a degree of parallelism $\leq m$ (number of processors)



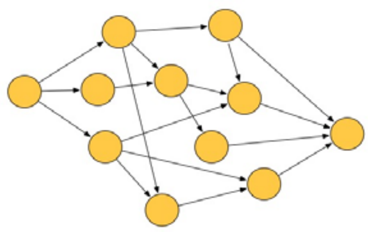
Synchronous-parallel

- Generalization of the fork-join model
- Allows consecutive parallel segments
- Allows an arbitrary degree of parallelism of every segment
- Synchronization at segment boundaries: a sub-task in the new segment may start only after completion of all sub-tasks in the previous segment



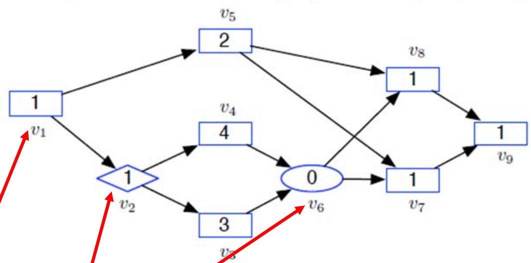
DAG

- Directed acyclic graph (DAG) $G_i = (V_i, E_i)$
- $V_i = \{v_{i,1}, \ldots, v_{i,n_i}\}$; $E_i \subseteq V_i \times V_i$
- Generalization of the previous two models
- Every node is a sequential sub-task
- Arcs represent precedence constraints between sub-tasks



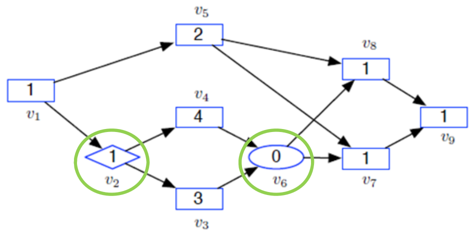
cp-DAG

- Conditional - parallel DAG (cp-DAG) $G_i = (V_i, E_i)$



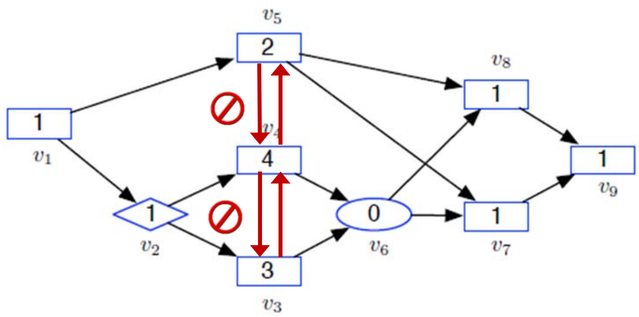
- Two types of nodes
  - **Regular**: all successors must be executed in parallel
  - **Conditional**: to model start/end of a conditional construct (e.g., if-then-else statement)
- Each node has a WCET $C_{i,j}$
- In this lecture, we will focus on **this** task model

Conditional pairs



- (v_2, v_6) form a **conditional pair**
  - v_2 is a starting conditional node
  - v_6 is the joining point of the conditional branches starting at v_2
- **Restriction**: there cannot be any connection between a node belonging to a branch of a conditional statement (e.g., v_4) and nodes outside that branch (e.g., v_5), including other branches of the same statement

Why this restriction?

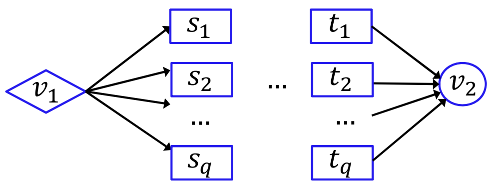


- It does not make sense for v_5 to wait for v_4 if v_3 is executed
- Analogously, v_4 cannot be connected to v_3 since only one is executed
- Violation of the correctness of conditional constructs and the semantics of the precedence relation

Formal definition (1)

Let (v_1, v_2) be a pair of conditional nodes in a DAG $G_i = (V_i, E_i)$. The pair (v_1, v_2) is a conditional pair if the following hold:

- Suppose there are exactly q outgoing arcs from v_1 to the nodes $s_1, s_2, \dots, s_q$, for some $q > 1$. Then there are exactly q incoming arcs into v_2 in E_i, from some nodes $t_1, t_2, \dots, t_q$

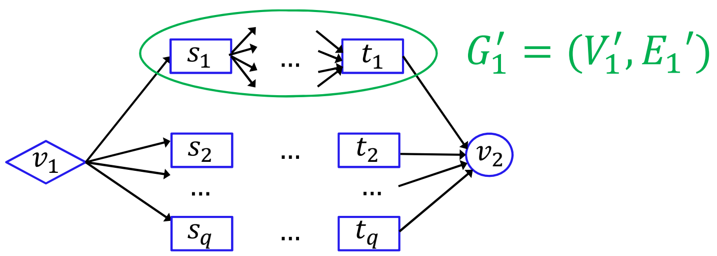


Real-Time Systems Laboratory

Formal definition (2)

- For each $l \in \{1,2,\dots,q\}$, let $V_l' \subseteq V_i$ and $E_l' \subseteq E_i$ denote all the nodes and arcs on paths reachable from s_l that do not include v_2.

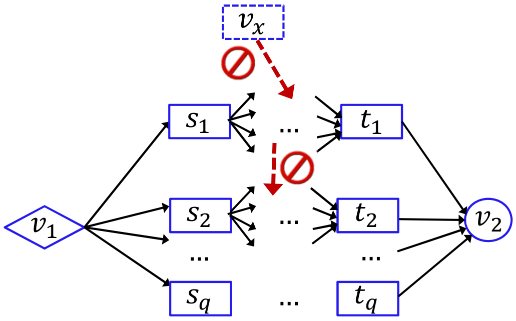
  By definition, s_l is the sole source node of the DAG $G_l' = (V_l', E_l{}')$. It must hold that t_l is the sole sink node of G_l'.



Real-Time Systems Laboratory

Formal definition (3)

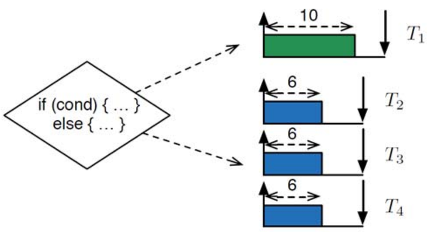
- It must hold that $V_l' \cap V_j' = \emptyset$ for all $l, j, l \neq j$.
  Additionally, with the exception of (v_1, s_l), there should be no arcs in E_i into nodes in V_l' from nodes not in V_l', for each $l \in \{1,2,\dots,q\}$.
  That is, $E_i \cap \left((V_i \backslash V_l') \times V_l'\right) = \{(v_1, s_l)\}$ should hold for all l.



Motivating example (1)

- Why is it important to explicitly model conditional statements?

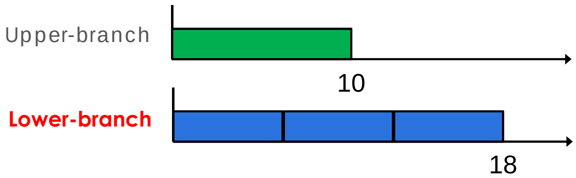
```
#pragma omp parallel num_threads(N)
 #pragma omp master {
 #pragma omp task { // T0
   if (condition) {
     #pragma omp task { // T1 }
   }
   else {
     #pragma omp task { // T2 }
     #pragma omp task { // T3 }
     #pragma omp task { // T4 }
}}}}
```

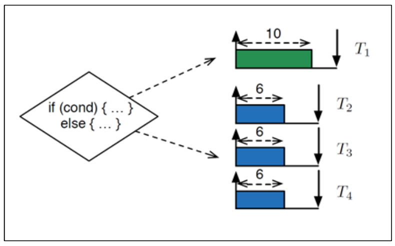


- Which branch leads to the worst-case response-time?

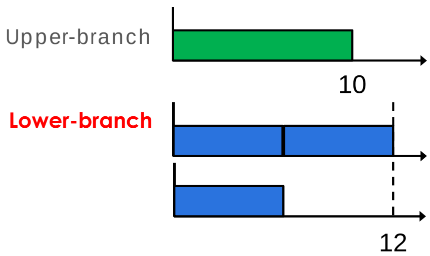
Motivating example (2)

- 1 processor



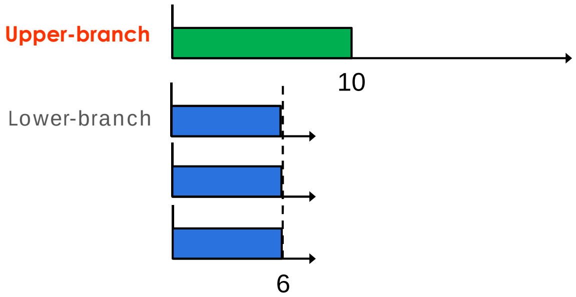


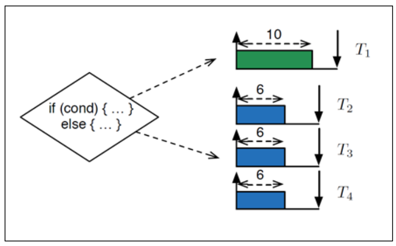
- 2 processors



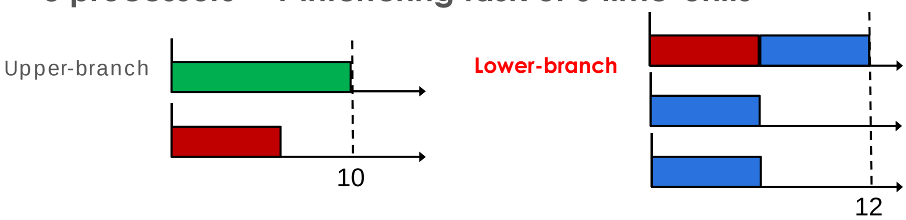
Motivating example (3)

- ≥ 3 processors





- 3 processors + 1 interfering task of 6 time-units



Motivating example (4)

- This example shows that it makes sense to enrich the task model with conditional statements when dealing with **parallel task models**
- Depending on the number of processors and on the other tasks, not always the same branch leads to the worst-case response-time
- Why we do not model conditional statements also with sequential task models?
  - Conditional branches are incorporated in the notion of WCET (longest chain of execution)
  - The only parameters needed to compute the response-time of a task are the WCETs, periods and deadlines of each task in the system

System model

- n conditional-parallel tasks (cp-tasks) τ_i, expressed as cp-DAGs in the form $G_i = (V_i, E_i)$
- platform composed of m identical processors
- **sporadic** arrival pattern (minimum inter-arrival time T_i between jobs of task τ_i)
- **constrained** relative deadline $D_i \leq T_i$

Problem: compute a **safe upper-bound** on the response-time of each cp-task, with any work-conserving algorithm (including Global FP and Global EDF)

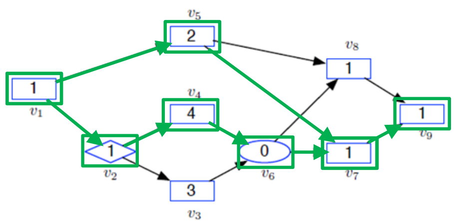
Quantities of interest

1. Chain (or path) of a cp-task
2. Longest path
3. Volume
4. Worst-case workload
5. Critical chain

Retis
Real-Time Systems Laboratory

1. Chain (or path)

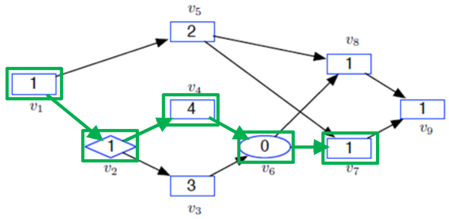
A chain (or path) of a cp-task τ_i is a sequence of nodes $\lambda = (v_{i,a}, \dots, v_{i,b})$ such that $(v_{i,j}, v_{i,j+1}) \in E_i, \forall j \in [a, b)$.



Retis
Real-Time Systems Laboratory

1. Chain (or path)

A chain (or path) of a cp-task τ_i is a sequence of nodes $\lambda = (v_{i,a}, \dots, v_{i,b})$ such that $(v_{i,j}, v_{i,j+1}) \in E_i, \forall j \in [a, b)$.

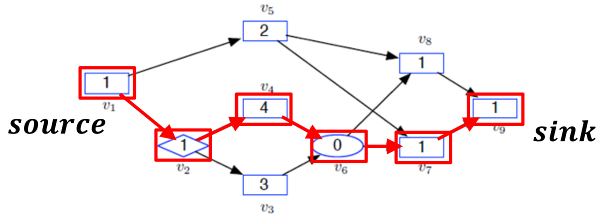


The length of the chain, denoted by $len(\lambda)$, is the sum of the WCETs of all its nodes:

$$len(\lambda) = \sum_{j=a}^{b} C_{i,j}$$

2. Longest path

The longest path L_i of a cp-task τ_i is any source-sink chain of the task that achieves the longest length



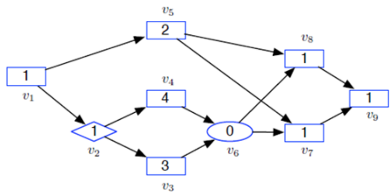
L_i also represents the time required to execute it when the number of processing units is infinite (large enough to allow maximum parallelism)

Necessary condition for feasibility: $L_i \le D_i$

2. Longest path

How to compute the longest path?

1. Find a topological order of the given cp-DAG
   - A topological order is such that of there is an arc from u to v in the cp-DAG, then u appears before v in the topological order → can be done in $O(n)$
   - Example: for this cp-DAG possible topological orders are
     - $(v_1, v_2, v_5, v_3, v_4, v_6, v_8, v_7, v_9)$
     - $(v_1, v_5, v_2, v_3, v_4, v_6, v_7, v_8, v_9)$
     - $(v_1, v_2, v_4, v_3, v_6, v_5, v_8, v_7, v_9)$



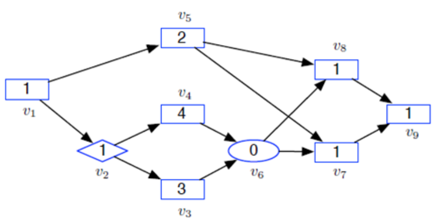
2. Longest path

How to compute the longest path?

2. For each vertex $v_{i,j}$ of the cp-DAG in the topological order, compute the length of the longest path ending at $v_{i,j}$ by looking at its incoming neighbors and adding $C_{i,j}$ to the maximum length recorded for those neighbors

   If $v_{i,j}$ has no incoming neighbors, set the length of the longest path ending at $v_{i,j}$ to $C_{i,j}$

   Example:
   - For v_1, record 1
   - For v_2, record 2
   - For v_3, record 5
   - For v_4, record 6
   - For v_5, record $\max(5, 6) = 6$

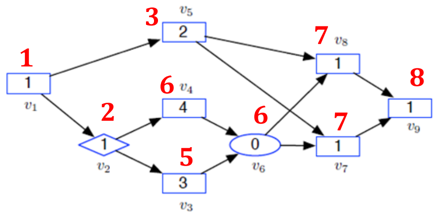


2. Longest path

How to compute the longest path?

3. Finally, the longest path in the cp-DAG may be obtained by starting at the vertex $v_{i,j}$ with the largest recorded value, then repeatedly stepping backwards to its incoming neighbor with the largest recorded value, and reversing the sequence found in this way

Example: **recorded values**



- Starting at v_9 and stepping backward we find the sequence $(v_9, v_7, v_6, v_4, v_2, v_1)$
- The longest path is then $(v_1, v_2, v_4, v_6, v_7, v_9)$

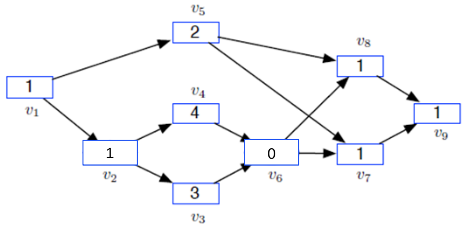
Complexity of the longest path computation: $O(n)$

3. Volume

In the **absence** of conditional branches, the volume of a task is the worst-case execution time needed to complete it on a dedicated single-core platform

It can be computed as the sum of the WCETs of all its vertices:

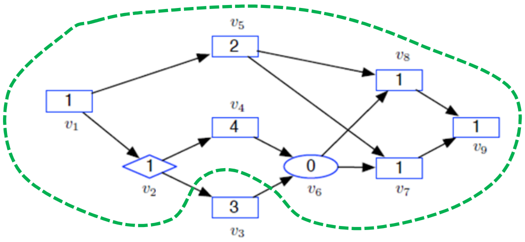
$$vol_i = \sum_{v_{i,j} \in V_i} C_{i,j}$$



It also represents the maximum amount of workload generated by a single instance of a DAG-task

4. Worst-case workload

In the **presence** of conditional branches, the worst-case workload of a task is the worst-case execution time needed to complete it on a dedicated single-core platform, *over all combination of choices for the conditional branches*

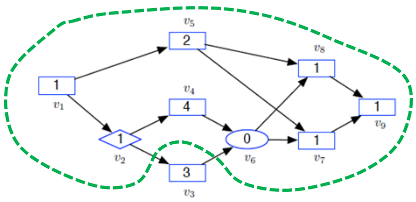


It also represents the maximum amount of workload generated by a single instance of a cp-task

In this example, the worst-case workload is given by all the vertices except v_3, since the branch corresponding to v_4 yields a larger workload

4. Worst-case workload

How can it be computed?



Algorithm 1 Worst-Case Workload Computation

procedure WCW(G)
  $\sigma \leftarrow$ TopologicalOrder(G)
  for $z = |V|$ down to 1 **do** — reverse topological order
    $i \leftarrow \sigma(z)$ — i takes the z^{th} element of the permutation
    $S(v_i) \leftarrow \{v_i\}$ — S takes the accumulated worst-case workload from v_i till the end of the cp-DAG
    if $\text{SUCC}(v_i) \neq \emptyset$ **then** — if the vertex has some successors
      if IsBeginCond(v_i) **then** — if the vertex is the head node of a conditional pair
        $v^* \leftarrow \text{argmax}_{v \in \text{SUCC}(v_i)} C(S(v))$ — v^* is the successor of v_i achieving the largest partial workload
        $S(v_i) \leftarrow S(v_i) \cup S(v^*)$ — $S(v^*)$ is merged into $S(v_i)$
      else — if instead the vertex is a regular one
        $S(v_i) \leftarrow S(v_i) \cup \bigcup_{v \in \text{SUCC}(v_i)} S(v)$ — the workload of all successors is merged into $S(v_i)$
      end if
    end if
  end for
  return $C(S(v_{\sigma(1)}))$ — the worst-case workload accumulated by the source vertex is returned as output
end procedure

4. Worst-case workload

- What is the complexity of this algorithm?

Algorithm 1 Worst-Case Workload Computation

procedure WCW(G)
  $\sigma \leftarrow$ TOPOLOGICALORDER(G)
  for $z = |V|$ down to 1 **do**
    $i \leftarrow \sigma(z)$
    $S(v_i) \leftarrow \{v_i\}$
    if SUCC(v_i) $\neq \emptyset$ **then**
      if ISBEGINCOND(v_i) **then**
        $v^* \leftarrow \text{argmax}_{v \in \text{succ}(v_i)} C(S(v))$
        $S(v_i) \leftarrow S(v_i) \cup S(v^*)$
      else
        $S(v_i) \leftarrow S(v_i) \cup \bigcup_{v \in \text{succ}(v_i)} S(v)$
      end if
    end if
  end for
  return $C(S(v_{\sigma(1)}))$
end procedure

- $O(|E|)$ set operations
- Any of them may require to compute $C(S(v_i))$, which has cost $O(|V|)$

The time complexity is then $O(|E||V|)$

5. Critical chain

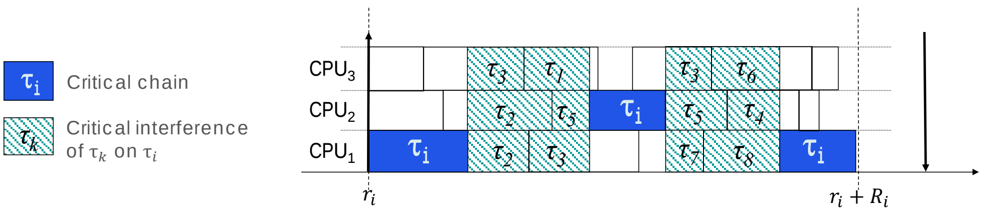
- Given a set of cp-tasks and a (work-conserving) scheduling algorithm, the **critical chain** λ_i^* of a cp-task τ_i is the chain of vertices of τ_i that leads to its worst-case response-time R_i
- How can it be identified?
  - We should know the worst-case instance of τ_i (i.e., the job of τ_i that has the largest response-time in the worst-case scenario)
  - Then we should take its sink vertex v_{i,n_i} and recursively pre-pend the last to complete among the predecessor nodes, until the source vertex $v_{i,1}$ has been included in the chain

Key observation: the critical chain is unknown, but is always upper-bounded by the longest path of the cp-task!

Critical interference

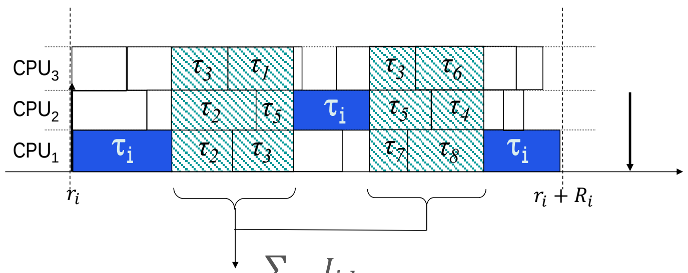
To find the response-time of a cp-task, it is sufficient to characterize the maximum interference suffered by its critical chain

The **critical interference** $I_{i,k}$ imposed by task τ_k on task τ_i is the cumulative workload executed by vertices of τ_k while a node belonging to the critical chain of τ_i is ready to execute but is not executing



Critical interference

- I_i: total interference suffered by task τ_i
- $I_{i,k}$: total interference of task τ_k on task τ_i



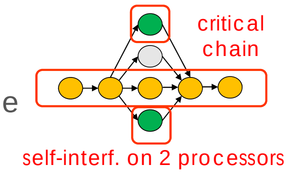
$$I_i = \frac{\sum_{\tau_k} I_{i,k}}{m}$$

For any work-conserving algorithm!

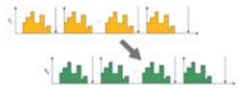
$$R_i = len(\lambda_i^*) + I_i = len(\lambda_i^*) + \frac{\sum_{\tau_k} I_{i,k}}{m}$$

Types of interference

- In the particular case when $i = k$, the critical interference $I_{i,i}$ includes interfering contributions of vertices of the same task (not belonging to the critical chain) on τ_i itself



- This type of interference is called **self-interference** (or *intra-task interference*) and is **peculiar to parallel tasks** only



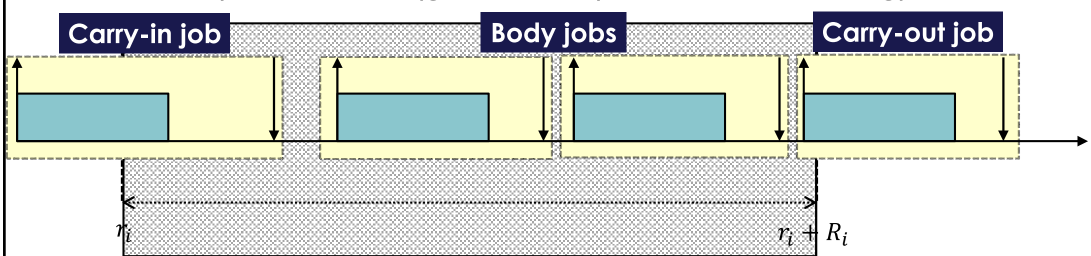
- The interference from other tasks in the system is called **inter-task interference**

$$R_i = len(\lambda_i^*) + I_i = len(\lambda_i^*) + \frac{\sum_{\tau_k} I_{i,k}}{m} = len(\lambda_i^*) + \frac{1}{m} I_{i,i} + \frac{\sum_{\tau_{k \neq i}} I_{i,k}}{m}$$

self-int. | inter-task int.

Inter-task interference

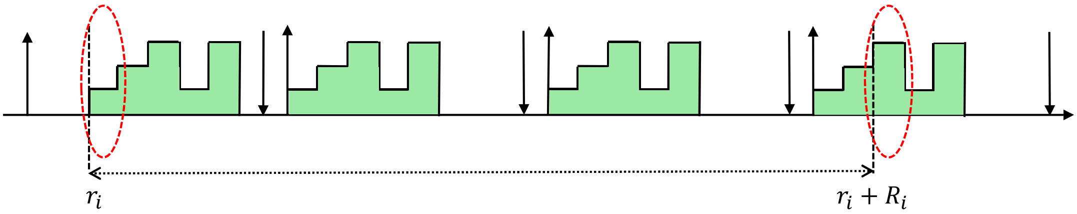
- Caused by other cp-tasks executing in the system
- Finding it exactly is difficult
- We need to find an **upper-bound on the workload** of an interfering task in the scheduling window $[r_i, r_i + R_i]$
- In the sequential case (global multiprocessor scheduling):



What is the scenario that maximizes the interfering workload?

Inter-task interference

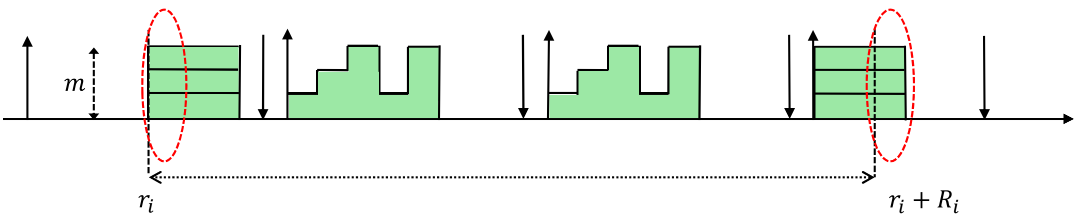
- **Sequential case**
  - The first job of τ_k starts executing as late as possible, with a starting time aligned with the beginning of the scheduling window
  - Later jobs are executed as soon as possible
- **Parallel case**
  - This scenario may not give a safe upper-bound on the interfering workload. Why?



Shifting right the scheduling window may give a larger interfering workload!

Inter-task interference

- **Pessimistic assumption**
  - Each interfering job of task τ_k executes for its worst-case workload W_k
  - The carry-in and carry-out contributions are evenly distributed among all m processors
  - Distributing them on less processors cannot increase the workload within the window
  - Other task configurations cannot lead to a higher workload within the window



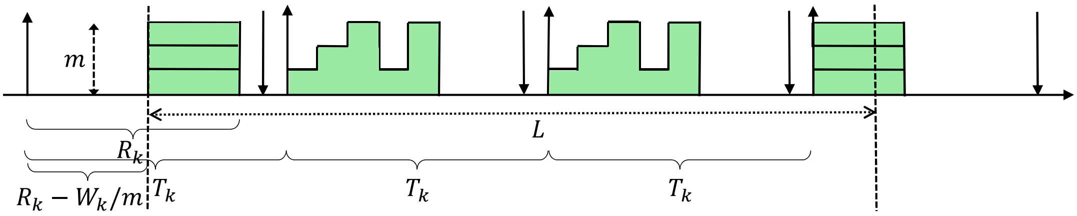
Inter-task interference

- **Lemma:** An upper-bound on the workload of an interfering task τ_k in a scheduling window of length L is given by

$$\mathcal{W}_k(L) = \left\lfloor \frac{L + R_k - W_k/m}{T_k} \right\rfloor W_k + \min\left(W_k, m \cdot \left(\left(L + R_k - \frac{W_k}{m}\right) mod\ T_k\right)\right)$$

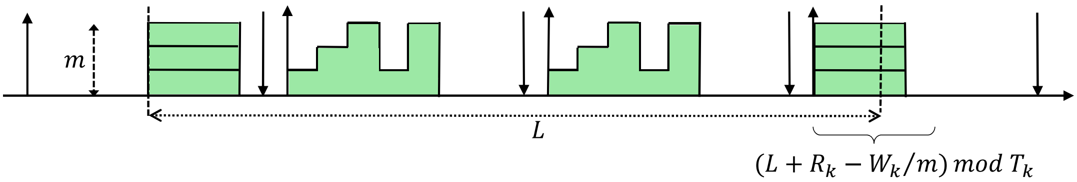
- **Proof:**
  - The maximum number of carry-in and body instances within the window is

$$\left\lfloor \frac{L + R_k - W_k/m}{T_k} \right\rfloor$$



Inter-task interference

- **Proof (continued):** $\mathcal{W}_k(L) = \left\lfloor \frac{L + R_k - W_k/m}{T_k} \right\rfloor W_k + \min\left(W_k, m \cdot \left(\left(L + R_k - \frac{W_k}{m}\right) mod\ T_k\right)\right)$
  - Each of the $\left\lfloor \frac{L+R_k-W_k/m}{T_k} \right\rfloor$ instances contributes for W_k
  - The portion of the carry-out job included in the window is $\left(L + R_k - \frac{W_k}{m}\right) mod\ T_k$



  - At most m processors may be occupied by the carry-out job
  - The carry-out job cannot execute for more than W_k units

Intra-task interference

- Simple upper-bound

$$I_{i,k}(L) \le \mathcal{W}_k(L)$$

$$R_i = len(\lambda_i^*) + I_i = len(\lambda_i^*) + \frac{1}{m} I_{i,i} + \frac{\sum_{\tau_{k \ne i}} I_{i,k}}{m}$$

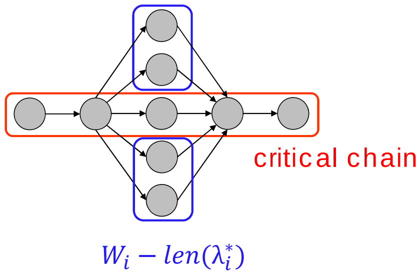
?

$$Z_i \overset{\text{def}}{=} len(\lambda_i^*) + \frac{1}{m} I_{i,i}$$

$$\le len(\lambda_i^*) + \frac{1}{m}(W_i - len(\lambda_i^*))$$

$$\le L_i + \frac{1}{m}(W_i - L_i)$$

Length of the longest path



Real-Time Systems Laboratory

Putting things together

- Schedulability condition

Given a cp-task set globally scheduled on m processors, an upper-bound R_i^{ub} on the response-time of a task τ_i can be derived by the fixed-point iteration of the following expression, starting with $R_i^{ub} = L_i$:

$$R_i^{ub} = L_i + \frac{1}{m}(W_i - L_i) + \left\lfloor \frac{1}{m} \sum_{\forall k \ne i} x_k^{ALG} \right\rfloor$$

- Global FP

$$x_k^{ALG} = x_k^{FP} = \begin{cases} \mathcal{W}_k(R_i^{ub}), & \forall\, k < i \\ 0, & otherwise \end{cases}$$

Decreasing priority order

- Global EDF

$$x_k^{ALG} = x_k^{EDF} = \mathcal{W}_k(R_i^{ub}), \forall\, k \ne i$$

$$\mathcal{W}_k(L) = \left\lfloor \frac{L + R_k - W_k/m}{T_k} \right\rfloor W_k + \min\left(W_k, m \cdot \left(\left(L + R_k - \frac{W_k}{m}\right) mod\ T_k\right)\right)$$

Real-Time Systems Laboratory

Putting things together

$$R_i^{ub} = L_i + \frac{1}{m}(W_i - L_i) + \left\lfloor \frac{1}{m} \sum_{\forall k \neq i} x_k^{ALG} \right\rfloor$$

- **Global FP**

  The fixed-point iteration updates the bounds in decreasing priority order, starting from the highest priority task, until either:

  - one of the response-time bounds exceeds the task relative deadline D_k (negative schedulability result);
  - OR no more update is possible (positive schedulability result), i.e., $\forall\, k$: $R_k^x = R_k^{x+1} \le D_k$

- **Global EDF**
  - Multiple rounds may be needed

Retis Real-Time Systems Laboratory

Reference

A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, G. Buttazzo, ***Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems***, Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)

Retis Real-Time Systems Laboratory

Thank you!

Alessandra Melani
alessandra.melani@sssup.it

Retis
Real-Time Systems Laboratory

45