Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems

Alessandra Melani

What does it mean?
- « Response-time analysis »
- « conditional »
- « DAG tasks »
- « multiprocessor systems »

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, OpenCL, Cilk, Cilk Plus, Intel TBB)
- Early real-time scheduling models: each recurrent task is completely sequential
- Recently, more expressive execution models allow exploiting task parallelism
Each task is an alternating sequence of sequential and parallel segments.

Every parallel segment has a degree of parallelism ≤ m (number of processors).

Fork-join

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.

Synchronous-parallel

Generalization of the previous two models.

Every node is a sequential sub-task.

Arcs represent precedence constraints between sub-tasks.

Directed acyclic graph (DAG)

Generalization of the fork-join model.

Every node is a sequential sub-task.

Arcs represent precedence constraints between sub-tasks.

DAG

Conditional - parallel DAG (cp-DAG)

Two types of nodes:
- Regular: all successors must be executed in parallel
- Conditional: to model start/finish of a conditional construct (e.g., if-then-else statement)

Each node has a WCET \( C_{ij} \).

In this lecture, we will focus on this task model.

cp-DAG

Conditional pairs

\((v_2, v_3)\) form a conditional pair:
- \(v_2\) is a starting conditional node
- \(v_3\) 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., \(v_3\)) and nodes outside that branch (e.g., \(v_2\)), including other branches of the same statement.

Why this restriction?

It does not make sense for \(v_2\) to wait for \(v_4\) if \(v_3\) is executed.

Analogously, \(v_4\) cannot be connected to \(v_2\) since only one is executed.

Violation of the correctness of conditional constructs and the semantics of the precedence relation.
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 \(s_1, s_2, \ldots, s_q\), for some \(q > 1\). Then there are exactly \(q\) incoming arcs into \(v_2\) in \(E_i\), from some nodes \(t_1, t_2, \ldots, t_q\).

Formal definition (1)

For each \(i \in \{1, 2, \ldots, q\}\), let \(V_i' \subseteq V_i\) and \(E_i' \subseteq E_i\) denote all the nodes and arcs on paths reachable from \(s_i\) that do not include \(v_2\).

By definition, \(s_i\) is the sole source node of the DAG \(G_i' = (V_i', E_i')\). It must hold that \(t_i\) is the sole sink node of \(G_i'\).

Formal definition (2)

It must hold that \(V_i' \cap V_j' = \emptyset\) for all \(i, j, i \neq j\). Additionally, with the exception of \((v_1, s_1)\), there should be no arcs in \(E_i\) into nodes in \(V_i'\) from nodes not in \(V_i'\), for each \(i \in \{1, 2, \ldots, q\}\). That is, \(E_i \cap (V_i' \times V_i') = \{(v_1, s_1)\}\) should hold for all \(i\).

Formal definition (3)

How is parallel code structured?

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

Which branch leads to the WCRT?
Lesson learnt

Depending on the number of processors and on the interfering tasks, it is not obvious to identify the branch leading to the WCRT.

It makes sense to account for the different execution flows by enriching the task model.

Why don’t we do it also with sequential tasks?

- Only the longest path matters
- Conditional branches are already incorporated in the notion of WCET

System model

- \( n \) conditional-parallel tasks (cp-tasks) \( \tau_i \), 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 \( \tau_i \))
- Constrained relative deadline \( D_i \leq T_i \)

Quantities of interest

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

1. Chain (or path)

A chain (or path) of a cp-task \( \tau_i \) is a sequence of nodes \( \lambda = (v_{i,0}, \ldots, v_{i,j}, \ldots, v_{i,k}) \) such that \( (v_{i,j}, v_{i,j+1}) \in E_i, \forall j \in [a, b] \).

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

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

2. Longest path

The longest path \( L_i \) of a cp-task \( \tau_i \) 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: \( L_i \leq D_i \)
How to compute the longest path?

1. Find a topological order of the given cp-DAG

   - A topological order is such that if there is an arc from \( u \) to \( v \) in the cp-DAG, then \( u \) appears before \( v \) in the topological order.

   - Example: for this cp-DAG possible topological orders are
     - \( (v_1, v_2, v_5, v_6, v_7, v_8, v_9) \)
     - \( (v_1, v_2, v_5, v_6, v_7, v_8, v_9) \)
     - \( (v_1, v_2, v_5, v_6, v_7, v_8, v_9) \)

2. Longest path

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

   - 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 \( \max(5, 6) = 6 \)

3. Finally, the longest path in the cp-DAG may be obtained by starting at the vertex \( v_{ij} \) 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 backwards we find the sequence \( v_9, v_8, v_7, v_6, v_5 \)
     - The longest path is then \( v_5, v_6, v_7, v_8, v_9 \)

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

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 \( v_5 \), since the branch corresponding to \( v_5 \) yields a larger workload.
4. Worst-case workload

- What is the complexity of this algorithm?

```
Algorithm 1 Worst-Case Workload Computation
1. procedure WCC(W)
2. s = TaskScheduler(W)
3. for i ∈ [1..n] do
4.   S(i) = ∅
5. if InitialCond(s) then
6.   S(i) = S(i) ∪ (S(s(i)) ∪ R(s(i))
7. end if
8. end for
9. return (Con(s))
10. end procedure
```

- Complexity:
  - \( O(|E|) \) set operations
  - Any of them may require to compute \( C(T,v_i) \), which has cost \( O(|E|) \)

The time complexity is then \( O(|E|) \).

5. Critical chain

- Given a set of cp-tasks and a (work-conserving) scheduling algorithm, the critical chain \( \lambda^*_i \) of a cp-task \( \tau_i \) is the chain of vertices of \( \tau_i \) that leads to its worst-case response-time \( R_i \).

- How can it be identified?
  - We should know the worst-case instance of \( \tau_i \) (i.e., the job of \( \tau_i \) that has the largest response-time in the worst-case scenario)
  - Then we should take its sink vertex \( v_{i,m} \) 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 \( i_{jk}^{\tau_i} \) imposed by task \( \tau_j \) on task \( \tau_i \) is the cumulative workload executed by vertices of \( \tau_j \) while a node belonging to the critical chain of \( \tau_i \) is ready to execute but is not executing

- Types of interference:
  - **Intra-task interference**: from vertices of the same task on itself; peculiar to parallel tasks only
  - **Inter-task interference**: from other tasks in the system; analogous to the classic notion

For any work-conserving algorithm:

\[
R_i = \text{len}(\lambda^*_i) + I_i = \text{len}(\lambda^*_i) + \frac{\sum_{k} i_{jk}^{\tau_i}}{m}
\]
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 \([r_i, n + R_i]\)
- In the sequential case (global multiprocessor scheduling):

  ![Diagram of task scheduling](image)

What is the scenario that maximizes the interfering workload?

Inter-task interference

- In the sequential case (global multiprocessor scheduling):
  - Carry-in job
  - Body jobs
  - Carry-out job

Inter-task interference

- Pessimistic assumption
  - Each interfering job of task \(\tau_\text{c}\) executes for its worst-case workload \(W_e\)
  - 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 \(\tau_\text{c}\) in a scheduling window of length \(L\) is given by
  \[
  W_e(L) = \frac{L + R_e - W_e/m}{m} + \min\left\{W_e, m \cdot \left(\frac{L + R_e - W_e}{m}\right) \mod T_e\right\}
  \]

Inter-task interference

- Proof:
  - The maximum number of carry-in and body instances within the window is
  \[
  \frac{L + R_e - W_e/m}{T_e}
  \]
  - This scenario may not give a safe upper-bound on the interfering workload. Why?

Inter-task interference

- Lemma: An upper-bound on the workload of an interfering task \(\tau_\text{c}\) in a scheduling window of length \(L\) is given by
  \[
  W_e(L) = \frac{L + R_e - W_e/m}{m} + \min\left\{W_e, m \cdot \left(\frac{L + R_e - W_e}{m}\right) \mod T_e\right\}
  \]

Inter-task interference

- Proof (continued):
  - Each of the \(\frac{L + R_e - W_e/m}{m}\) instances contributes for \(W_e\)
  - The portion of the carry-out job included in the window is \(\left(\frac{L + R_e - W_e/m}{T_e}\right) \mod T_e\)
  - At most \(m\) processors may be occupied by the carry-out job
  - The carry-out job cannot execute for more than \(W_e\) units

Inter-task interference

- Sequential case
  - The first job of \(\tau_\text{c}\) 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

Inter-task interference

- 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

- Longest path is 10 time-units
- Critical chain can be either 10 or 6

Intra-task interference

- It is the interference from vertices of the same task on itself
- The interfered contribution is the critical chain
- Critical chain: chain that leads to the WCRT of the cp-task

Intra-task interference

- Critical chain ≠ longest path
- Longest path is 10 time-units
- Critical chain can be either 10 or 6

Who is interfering and who is interfered?
Intra-task interference

- Simple upper-bound
  \[ R_i = \text{len}(Q_i) + t_i + \frac{1}{m} (\text{len}(Q_i) + t_i) \]

\[ I_{lk}(L) \leq W_k(L) \]

putting things together

- 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 \( \bar{D}_i \) (negative schedulability result);
  - OR no more update is possible (positive schedulability result), i.e., \( \forall i: R_i^k = R_i^{k+1} \leq \bar{D}_i \)

- Global EDF
  Multiple rounds may be needed

Schedulability example

- Global FP
  \[ R^{(1)} = 28 \]
  \[ R^{(2)} = 32.5 \leq 35 \]

- Task 1 is schedulable

Solution sketch

- Task 1 is schedulable
  \[ R^{(1)} = 28 \]
  \[ R^{(2)} = 32.5 \leq 35 \]

- Task 2 is schedulable
  \[ R^{(1)} = 37 \]
  \[ R^{(2)} = 69.5 \]

Reference

Thank you!

Alessandra Melani
alessandra.melani@sssup.it