Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems

Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Giorgio Buttazzo
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
How is parallel code structured?

```c
#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?

```c
```
Which branch leads to the WCRT?

1 processor

Upper branch

Lower branch

2 processors

Upper branch

Lower branch

if (condition) {...} else {...}
Which branch leads to the WCRT?

\[ \geq 3 \text{ processors} \]

Upper branch

Lower branch

3 processors + interfering task

Upper branch

Lower branch

if (condition) {...} else {...}

T₁

T₂

T₃

T₄

10

6

6

6

10

12
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
Some history

Fork-join

Synchronous-parallel

This work

Conditional parallel DAG (cp-DAG)

Directed Acyclic Graph (DAG)
The cp-DAG model

Each task is represented as a conditional parallel DAG (cp-DAG) $G_i = (V_i, E_i)$

Vertices can be of two types:

- **Regular**: all its successors may be executed in parallel
- **Conditional**: come in start/end pairs and require the execution of *exactly one* successor of the start node

$(v_2, v_6)$ is a pair of conditional nodes
Model restriction

There cannot be any connection between a vertex belonging to a branch of a conditional statement and vertices outside that branch.

It does not make sense for $v_5$ to wait for completion of $v_4$ if the branch corresponding to $v_3$ is executed.

Analogously, $v_4$ can’t be connected to $v_3$ since only one of them is executed.
System model & problem definition

- Set of conditional parallel tasks $T_1, ..., T_n$, expressed as cp-DAGs
- Each vertex $v_{i,j}$ of task $T_i$ has a WCET $C_{i,j}$
- Platform composed of $m$ identical processors
- Sporadic arrival pattern
- Constrained relative deadlines $D_i \leq T_i$

Problem

Schedulability analysis for cp-tasks, *globally* scheduled on $m$ identical processors
Work-conserving schedulers

Global schedulers are typically work-conserving (e.g., Global FP/EDF)

**Property:** a ready job cannot execute only if all $m$ processors are busy

We can safely assume that the interference is distributed across all $m$ processors
Types of interference

We need to deal with two types of interference:

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

Interfering (i.e., not critical)

Interfered (i.e., critical)
Inter-task interference

How to characterize the largest interfering workload from a higher-priority job, accounting for conditional branches?

In the *absence* of conditional branches, it is given by the *volume* of the DAG task

\[ \text{vol}(\tau_i) = \sum_{v_{k,i} \in V_i} C_{k,i} \]

In the *presence* of conditional branches, it is generalized by the **worst-case workload** of the cp-task

How to compute it?
Worst-case workload computation

Dynamic programming algorithm

i. Consider vertices in reverse topological order
   - If there is an arc from \( v_i \) to \( v_j \), then \( v_i \) appears before \( v_j \) in the topological order

ii. Compute the partial worst-case workload from the current vertex to the sink
Worst-case workload computation

- For a non-conditional vertex, sum the contribution of all successors to the worst-case workload.

- If the current vertex is the head of a conditional branch, select the branch with the largest worst-case workload.

Algorithm 1: Worst-Case Workload Computation

1: procedure WCW(G)
2:   σ ← TOPOLOGICALORDER(G)
3:   S(v_{sink}) ← \{v_{sink}\}
4:   for v_i ∈ σ from sink to source do
5:     if SUCC(v_i) ≠ ∅ then
6:       if ISBEGINCOND(v_i) then
7:         v* ← argmax_{v ∈ SUCC(v_i)} C(S(v))
8:         S(v_i) ← \{v_i\} ∪ S(v*)
9:       else
10:          S(v_i) ← \{v_i\} ∪ \bigcup_{v ∈ SUCC(v_i)} S(v)
11:     end if
12:   end if
13: end for
14: return C(S(v_{source}))
15: end procedure

Complexity

$O(|V||E|)$, i.e., quadratic in the graph size.
Advantages of worst-case workload

- With a **single parameter** we can characterize the interfering workload of a higher-priority task.
- It **abstracts** from the structure of the DAG and the conditional choices.
- It allows deriving an **accurate sufficient schedulability test**, based on traditional RTA approaches for globally scheduled systems.

![Diagram](image)

\[ W_i = 11 \]
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

Who is *interfering* and who is *interfered*?

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

It holds that critical chain length $\leq$ longest path length $L_k$

- The **intra-task interference** is given by vertices not belonging to the critical chain

$$I_{k,k} \leq W_k - L_k$$

Then, the contribution on the response-time due to the task itself is upper-bounded by:

$$Z_k \leq L_k + \frac{1}{m}(W_k - L_k)$$

interfered interfering
Intra-task interference

The previously introduced upper-bound is **pessimistic**!

\[
Z_k \leq L_k + \frac{1}{m} (W_k - L_k) = 5 + \frac{1}{2} (6 - 5) = 5.5
\]

- If the **upper-branch** is taken: \( Z_k = L_k = 5 < 5.5 \)
- If the **lower-branch** is taken: \( Z_k = 4 + \frac{1}{2} (6 - 4) = 5 < 5.5 \)

The problem is that the **longest path** corresponds to the **upper-branch**, the **worst-case workload** is given by the **lower-branch**, but the two branches are mutually exclusive.
Improving the bound

To solve the problem, we have derived a **refined upper-bound** on $Z_k$ that *jointly* computes the worst-case workload and the longest chain length.

**Dynamic programming algorithm**

i. Scan vertices in reverse topological ordering

ii. For each node store:
   - $S(v_k)$: set of vertices determining partial largest workload
   - $T(v_k)$: set of vertices determining partial longest chain
   - $f(v_k)$: partial $Z_k$ value

iii. Take different decisions depending on vertex type

**Complexity**

$O(|V||E|\Delta)$, i.e., still polynomial in the graph size

$\Delta$: maximum out-degree of a vertex

\[ Z_k = 5 \]
Wrapping up

Starting from a set of complex constraints, such as:

- global multiprocessor scheduling
- intra-task parallelism
- precedence constraints
- conditional executions

we derived:

- bounds on inter-task interference
- bounds on intra-task interference
- a simple sufficient test with pseudo-polynomial complexity

... but how does the test perform in practice?
Experimental setting

Synthetic cp-task generation [1]:

- First, series-parallel graphs are generated by recursively expanding non-terminal vertices to either terminal vertices, conditional subgraphs or parallel subgraphs;
- then, edges are randomly added (compatibly with the restrictions imposed by the model) to obtain a cp-DAG;
- we measured the number of schedulable task-sets as a function of:
  - task-set utilization $U$
  - number of processors $m$
  - number of tasks $n_i$ in each task-set

Conditional DAG tasks

Global FP, $m = 4$


Global FP, $m = 4, U = 2$

~5 times better
Our approach significantly tightens the schedulability of conditional DAG tasks, as well as of classical DAG tasks.
Conclusions

We have introduced the **conditional parallel DAG task model** as a generalization of the sporadic DAG model.

This new task model incorporates **conditional control flow structures** to derive tighter estimates of interfering contributions.

A **schedulability analysis** has been derived to compute safe upper-bounds on the response-time of each task, ensuring:

- the *same complexity* as the classic RTA for sequential tasks
- a *low amount of information* required to carry on the analysis
- the *best schedulability performance* over all existing analyses for conditional and/or parallel DAG tasks
Thank you!

Alessandra Melani
alessandra.melani@sssup.it
Other plots

Global FP, $U = 2$

Global EDF, $m = 8, U = 2$