Schedulablity analysis of conditional parralel task graphs in multicore systems: Final Paper

*Real time systems

Patrick NONKI
Electronics Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
patrick.nonki@stud.hshl.de

***Abstract*—The parallel DAG model is extended in this study by incorporating conditional components to offer a more detailed examination of parallel task systems. Each job is represented by a DAG comprising both parallel and conditional nodes in the proposed conditional parallel task (cptask) paradigm. A formal description of cp-task is provided by outlining the possible connections between the various (conditional and non-conditional) components of the graph in order to reflect the structure of parallel applications. Using multiple global scheduling methods, effective techniques to determine an upper-bound on the response-time of each cp-task are derived for the cp-task model.**
A federated method to multiprocessor scheduling of systems of independent recurrent tasks is discussed, in which each task can either execute proactively on a single processor or can execute on several processors but has exclusive access to all of them. The conditional sporadic DAG tasks model is used to construct efficient polynomial-time algorithms for federated schedulability analysis and run-time scheduling of recurrent task systems. The speedup factor metric quantifies the combined cost of both confining oneself to the federated scheduling paradigm and needing the scheduling algorithms to run in polynomial time. [1]

***Index Terms*—DAG, cp-task, conditionnal tasks, scheduling, parralel tasks**

I. Introduction

Several task models, such as fork/join, synchronous parallel, DAG-based, and others, have been proposed in the literature to characterize the intrinsic parallelism of real-time activities. Although these task models have been given schedulability tests and resource augmentation bounds in the context of multicore systems, they are still too pessimistic to describe the execution flow of parallel tasks characterized by multiple (and nested) conditional statements, where it is difficult to choose which execution path to model the worst-case scenario. To address this issue, this study provides a task model that considers conditional parallel tasks (cp-tasks) represented by DAGs with both precedence and conditional edges to integrate control flow information. A set of useful parameters is found and computed by efficient algorithms for this job model. [2]
Extensive experiments are used to evaluate the efficiency of the suggested schedulability analysis. The research also demonstrates that the proposed response-time analysis may be extended to non-conditional task models (such as the DAG task model [11]) with ease. In comparison to existing techniques, experimental results reveal that these systems detect a significantly higher number of schedulable tasksets at a significantly lower time complexity.

II. Algorithm explanation

In this research, the main understanding of the topic is based on the schedulability and performance analysis of a system where we have conditional tasks running together in parralel and in different cores of the processor unit. The algorithm [2] below shows the process in general

```
void num_threads(N){
    void master{
        void task{ //t0
            if (condition){
                task {//t1}
            else{
               task{t2}
               task{t3}
               task{t4}
    }}}}
```

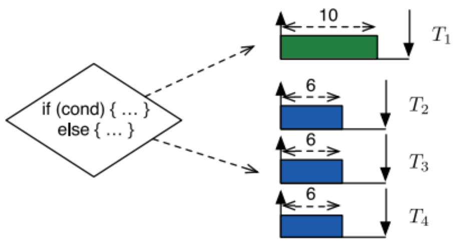

Fig. 1. Block diagram

Based on [2], let's assume we have a different set of n conditionnal parralel tasks (cp tasks) running in m identical processors. Let's also have a graph where we map nodes as sub-tasks, arcs to join nodes, and each node having the so called *worst case time execution (WCET).*

A node that has no incoming arc is called a source whereas a node with no ongoing arc is called a sink.
In our representation we can have 2 types of nodes:

- Conditional nodes : represented either by diamond (beginning of the condition) or by circle (end of the condition)
- Regular nodes : represented by rectangles, as normal subtask

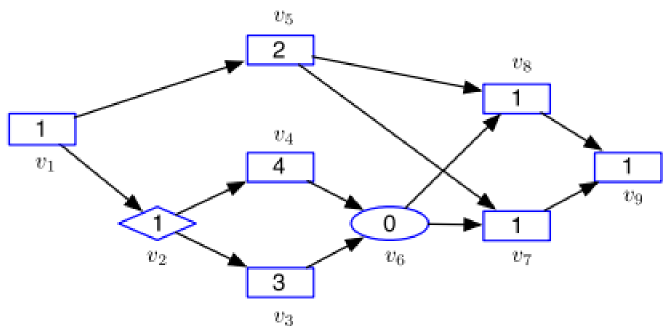

Fig. 2. Conditional parralel tasks graph

On the figure we can identify every element listed above.
In the graph we also define the length of the cp task as the length of any longest path in the graph. In the absence of conditional subtasks, it's calculated by having a sum of the WCETs of all the subtasks. In case we have conditional subtasks, the length is now called *Worst case workload* which is calculated by adding WCETs and maximun WCET amongst conditional branches. [2]
In our graph, the length is L=8 and since we have V4>V3 in our conditional branch, the worst case workload W=11. [2]

III. Interference of conditional parralel tasks

This section discusses the schedulability of cp-tasks that are globally scheduled by any work-conserving scheduler. The concept of interference is used to establish the analysis. The interference on a task τk is defined in the existing literature for globally scheduled sequential task systems as the sum of all intervals in which τk is ready but cannot execute because all cores are busy executing other tasks. We introduce the concept of critical interference to adapt this definition to the parallel nature of cp-tasks. [2]
Fix a work-saving scheduler and a collection of cp-tasks T. The crucial chain of a task is the first relevant concept.
Definition: A cp-critical task's chain λ*k is the chain of nodes that leads to the task's worst-case response time Rk (in case of ties, fix one such chain randomly). [2]
In principle, the critical chain of cp-task τk is defined by recursively pre-pending the last to complete among the predecessor nodes (whether conditional or not) until source vertex vk,1 is included in the chain. A critical node of task τk is a node in the critical chain of τk. The sink node is always a critical node since the response time of a cptask is determined by the response time of the task's sink vertex. It is then sufficient to characterize the highest interference incurred by a task's crucial chain in order to derive the task's worst-case response time. [2]
Definition 4.2 : Critical interference Ik on task τk is the total amount of time that some crucial nodes in the worst-case instance of k are ready but not executed because all cores are busy. [2]
Lemma: The worst-case response time Rk of each task τk satisfies given a set of cp-tasks T planned by any workconserving algorithm on m identical processors. [2]
Rk $\leq$ len(λ∗k)+ Ik
The term Ik may be difficult to compute, which makes this equation challenging to use for schedulability analysis. [2]
The entire interfering burden can be expressed as a function of individual contributions from interfering tasks, and such contributions can then be upper-bound using the worst-case workload of each interfering task τi. We'll show you how to calculate such interfering contributions and how they're related to establish the total interfering workload in the sections below.
Definition: Interference that is crucial The cumulative workload executed by sub-tasks of τI when a critical node of the worst-case instance of τk is ready to execute but not executing is defined as Ii,k imposed by task τI on task τk. [2]
Lemma: For all the work conserving algorithms we have :
Ik = 1/m $\sum\tau$i $\in T^S$Ii, k [2]

IV. Conditional Sporadic Dag Tasks [1]

The conditional sporadic DAG task model is now described as a generalization of traditional sporadic DAG tasks. Each conditional sporadic DAG job τI is given as a 3-tuple (Gi, Di, Ti), where Gi=(Vi, Ei) Figure 4 is a DAG and Di and Ti are positive numbers, just as traditional sporadic tasks. Gi must have a single source vertex and a single sink vertex in order to function. Conditional vertices are special vertices declared in pairs in Vi. In the DAG Gi=, let (c1, c2) be such a pair (Vi, Ei). Informally, vertex c1 can be regarded of as a place in the code where a conditional expression is evaluated and control is then flowed along one of several different possible paths in the code, depending on the outcome of the evaluation. All of these distinct pathways must intersect at a common location in the code, which is represented by the vertex c2. In a formal setting:

- In Ei, there are several outgoing edges from c1. Assume that for some k>1, there are exactly k outgoing edges from c1 to the vertices s1,s2,...,sk. This conditional's branching factor is referred to as k. Then, from the vertices t1,t2,...,tk, there are exactly k incoming edges into c2 in Ei.
- Let Vl ⊂ Vi and El ⊂ Ei indicate all the vertices and edges on paths reachable from sl that do not include vertex c2 for each l ∈ 1,2,..., k. sl is the only source vertex in the DAG Gl=def(Vl, El) by definition. It must be true that tl is Gl's only sink vertex.
- It should be that V'l ∩ V'j = ∅ for all l,j, l≠j. Furthermore, for each l1,2,..., k, there should be no edges in Ei into vertices in Vl from vertices not in Vl, with the exception

of (c1, sl). In other words, Ei((ViVl)Vl)=(c1, sl) should hold for every l.

Edges (v1, v2) between pairs of vertices that aren't conditional vertices reflect precedence restrictions in the same way as they do in traditional sporadic DAG jobs, whereas edges containing conditional vertices represent conditional code execution. Let (c1, c2) denote a pair of conditional vertices that are defined (recall that conditional vertices are always defined in pairs). Because of the semantics of conditional DAG task execution,

- Following the completion of work c1, one of its successor jobs becomes eligible to perform; the successor job to execute is unknown ahead of time.
- Following the completion of one of its predecessor jobs, job c2 begins to run.

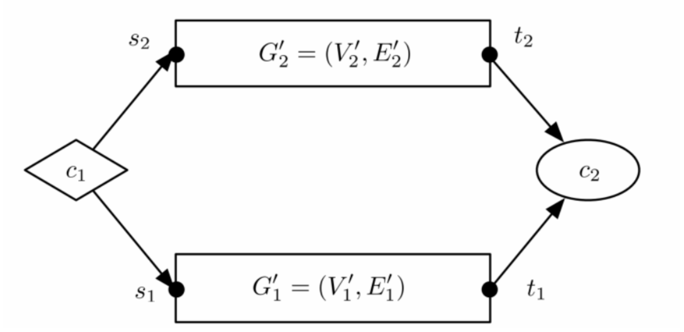

Fig. 3. A canonical conditional construct with branching factor 2 [1]

An "if-then-else" condition has a branching factor of 2. We will assume in this study that all conditionals have branching factor 2 without losing generality; conditionals with branching factor >2 can always be transformed to an equivalent sequence of conditionals with branching factor 2 with no more than a polynomial increase in DAG size.

Additional terminology: we refer to the subgraph of Gi beginning at c1 and finishing at c2 as a conditional construct in Gi for each pair (c1, c2) of conditional vertices in Gi. Each conditional construct in Gi's model represents a branching choice in the code. Figure 3 shows a graphical representation of a canonical conditional construct with branching factor 2. Conditional constructions can be nested in our model: a conditional construct can have other conditional constructs within it. The term "inner-most conditional construct" refers to a conditional construct that contains no other conditional constructs.

Figure 4 shows the DAG for an example conditional sporadic DAG task. Small solid circles represent "dummy" vertices, which correspond to tasks with wcet=0. The start and end conditional vertices are represented by diamond and oval vertices, respectively; this DAG has two pairs of conditional vertices, each with a branching factor of two. (On a color medium, the upper conditional construct is depicted in blue, while the lower conditional construct is represented in red.) This DAG

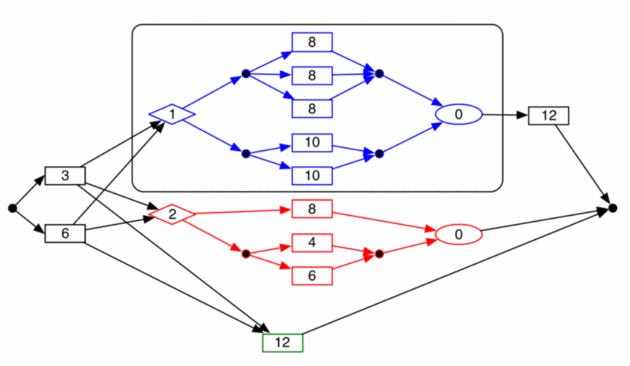

Fig. 4. The DAG of an example conditional sporadic DAG task [1]

task's semantics are as follows. The dummy source vertex has no execution requirement and hence completes execution immediately when a dag-job is released.Two vertices, with wcets 3 and 6, are now both eligible for execution. When both of these occupations are finished, three jobs become available at the same time.

- A conditional phrase with wcet of 1 is evaluated, and depending on the result, either three jobs with wcet equal to 8, or two jobs with wcet equal to 10, are executed. Following the completion of these jobs, a single job with wcet equal to 12 is run.
- A conditional statement with a wcet of 2 is also tested. Depending on the results of this evaluation, either a single project with a wcet of 8 is completed, or two jobs with wcets of 4 and 6 are completed.
- A single job is run with wcet equal to 12.

We want to use m unit-speed processors to arrange a set of n conditional sporadic DAG tasks $\tau 1$,$\tau 2$,...,τn. In this study, we focus on constrained deadline systems, such as those in which Di≤Ti applies to all tasks I in the system. This task system's federated scheduling is of particular relevance to us; that is, we want to:

- To begin, decide which tasks should be assigned to several processors for exclusive usage and which should be executed on a single processor (that will perhaps be shared with other tasks).
- Determine how many processors to assign to each task that will be executed solely on multiple processors.
- Determine such a division of the tasks for the remaining tasks - those that will be partitioned among the remaining processors.

A dag-job has the highest total wcet if the higher branch (1+(8 3)+0=25) is used for the upper conditional and the lower branch (2+(4+6)+0=12) is used for the lower conditional in our example DAG task $\tau 1$ in Figure 4. As a result, the total maximum wcet is 6 + 3+25+12+12+12=70, implying that vol1=70.

V. Conclusion

The cp-task model, which incorporates conditional branches, is a new task model that generalizes the standard sporadic DAG task model. The schedulability analysis uses this additional information to derive a more accurate estimate of the interfering contributions by differentiating their level of parallelism depending on the conditional path adopted. Two recursive composition methods have been used to characterize the topological structure of a cp-task graph.Then, using a schedulability analysis, a safe upper-bound on each task's reaction time in pseudo-polynomial time was computed. The suggested analysis has the advantage of requiring only two factors to characterize the complicated structure of the conditional graph of each task: the worst-case workload and the length of the longest path, in addition to its decreased complexity. Algorithms to derive these parameters from the DAG structure in polynomial time have also been presented. Scheduling tests using randomly generated cp-task workloads reveal that the suggested approach not only outperforms previously published solutions for conditional DAG tasks, but can also be utilized to greatly tighten the schedulability analysis of conventional (non-conditional) sporadic DAG task systems. [2]

Acknowledgement

This work has been made possible at the Hochschule Hamm-Lippstadt under the course Real time systems and supervised by Pr.Dr Stefan Henkler. Thanks also to Abdul-Azeez Olanlokun for the review that has contributed to make this paper better.

References

[1] Sanjoy Baruah. The federated scheduling of systems of conditional sporadic DAG tasks. In *2015 International Conference on Embedded Software (EMSOFT)*, pages 1–10, October 2015.

[2] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio Buttazzo. Schedulability Analysis of Conditional Parallel Task Graphs in Multicore Systems. *IEEE Transactions on Computers*, 66(2):339–353, February 2017. Conference Name: IEEE Transactions on Computers.