CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

How to deal with control-flow information in parallel real-time applications?

José Fonseca
Vincent Nélis
Gurulingesh Raravi
Luis Miguel Pinho

CISTER-TR-141201

2014/07/08

How to deal with control-flow information in parallel real-time applications?

José Fonseca, Vincent Nélis, Gurulingesh Raravi, Luis Miguel Pinho

CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: jcnfo@isep.ipp.pt, nelis@isep.ipp.pt, guhri@isep.ipp.pt, lmp@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

No abstract (2-pages paper).

How to deal with control-flow information in parallel real-time applications?

José Carlos Fonseca, Vincent Nelis, Gurulingesh Raravi and Luís Miguel Pinho
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
Email: {jcnfo,nelis,guhri,lmp}@isep.ipp.pt

I. Motivation for a new parallel application model

Traditional sequential programming models struggle to harness the immense processing capabilities offered by the new massively parallel architectures. It is also widely recognized that coarse-grained thread-based parallel programming models scale poorly due to load balancing difficulties. As a result, a shift in software development towards fine-grained task-based parallel programming paradigms is currently under way. For the real-time scheduling community, this paradigm shift entails the adoption of new application models to represent *precedence constraints* and *synchronization points* among all the computing units (referred to as *sub-tasks* hereafter).

Current models typically represent a parallel application as a single *graph*. Each *node* of the graph models a sequential sub-task and the *edges* that connect the nodes model all kinds of functional and/or non-functional dependencies between these sub-tasks. Every sub-task (node) is typically characterized by (at least) a WCET estimation and by some additional parameters such as a best-case/average-case execution time estimation, a memory footprint, and an input and output list of parameters. The edges may represent functional or non-functional precedence constraints, as in the fork/join task model [1], or explicit data dependencies as in the synchronous parallel task model [2] and the parallel DAG model [3]. In addition, every edge may also be annotated with additional information/constraints on the dependency between the two nodes that it connects. However, *none of these models can capture the control-flow information (such as conditional execution of code) of the applications that they model.* Consequently, in such models, *all* the nodes in the graph *must* execute each time the application is run. As a result, a simple application (see Fig. 1a) that executes in parallel either (as a result of an if-statement) four instances of a function B or two instances of a function C, cannot be properly modelled by using any of the parallel application models proposed so far in the real-time literature. All the existing models are limited by their implicit assumption that all the nodes must execute each time the application is run. Due to this limitation, these models:

Case 1: model function A as a node connected to four nodes B and two nodes C, which are in turn connected to node D (see Fig. 1b). The resulting graph is then composed of 8 nodes that are all assumed to be executed each time the application is run. This leads to an obvious over-approximation of the maximum workload[1] (denoted W) of the application, which makes the higher level analyses (like the schedulability analysis) more pessimistic.

Case 2: consider only the "worst-case execution flow" of the application and model that single flow as a graph of sub-tasks that must all execute. However, it is not trivial to determine the worst-case execution flow of a parallel application. Assuming an infinite number of cores, the WCET of the application is reached by executing the flow (i.e. the graph of sub-tasks) of maximum "critical path length" (denoted CP). In our example, such a WCET is reached by taking the "else" flow depicted in Fig. 1d which has the longest critical path length ($= 1 + 5 + 2 = 8$). However, this flow is not necessarily the one causing the maximum interference to the other applications of the system. Assuming that our example application has the highest priority in a system with 6 cores, although its WCET is reached by executing the "else" flow, this flow will occupy at most two cores at the same time and has a maximum workload of $W = 13$. On the other hand, the "if" flow depicted in Fig. 1c is shorter ($\mathrm{CP} = 7$) but it may cause more interference on the lower priority applications as it will demand more processing resources ($W = 19$) from four cores at the same time. Therefore, when the schedulability of all the applications in the system has to be assessed, it is a non-trivial task to identify these so-called application worst-case execution flow and to the best of our knowledge, this problem is still an open one.

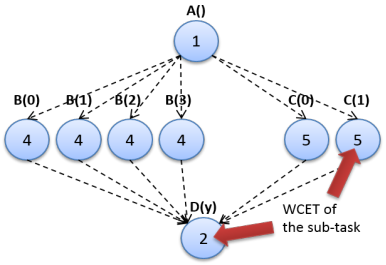
Our research identifies the need to support modelling of such applications for which each run may execute a different set of sub-tasks and all these sub-tasks may have different dependencies from one run to another. The main difference with the existing parallel application models is that we intend to represent an application as *a set of DAGs* where each DAG models a single execution flow of the application. We believe that such a model is more accurate and efficient than the state-of-the-art as it does *not* create a single DAG by cross-cutting the structures of multiple execution flows with different parameters (for instance, we will model the "if" and "else" flows of Fig. 1c and Fig. 1d as two separated DAGs instead of modelling the application by using the graph of Fig. 1b). We also believe that one of the first computation phases of the currently-available WCET analysis tools [4], during which the control flow graph is reconstructed by parsing the code of the application, can be adapted to our application model such that the tool will identify every feasible execution flow. However, instead of modelling an execution flow as a simple sequence of instructions or basic blocks (as the tools currently do), a flow will be modelled as a DAG of sub-tasks with data dependencies between them. Further, we also believe that the same tools may be adapted to derive various useful information on every execution flow such as its maximum workload, its "critical path length", its maximum concurrency level, etc.

From a real-time perspective, this new model poses serious challenges, especially when trying to compute the response time of an application scheduled conjointly with other applications. With this model, it must be verified that *every* feasible execution flow (which is now a DAG) of the application under analysis completes before the application deadline, and this must hold

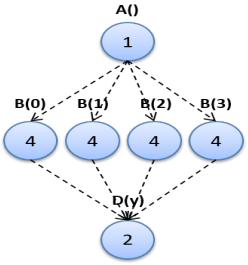
[1]The maximum workload is defined as the maximum amount of work that the application can impose on the system, i.e. it is the sum of the WCET of all its nodes.

```
x = A();
y = 0;
if (x > 1) {
  #pragma omp parallel for reduction(+:y)
  for (i = 0; i < 4; i++)
    y += B(i);
} else {
  #pragma omp parallel for reduction(+:y)
  for (i = 0; i < 2; i++)
    y += C(i);
}
z = D(y)
```

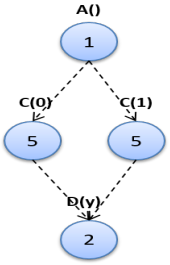
(a) Example of a parallel application using Open MP.



(b) Complete graph of the application. $W = 29$ and $\mathrm{CP} = 8$.



(c) "If" execution flow. $W = 19, \mathrm{CP} = 7$.



(d) "Else" execution flow. $W = 13, \mathrm{CP} = 8$.

true considering all possible interference patterns from the other applications. Knowing that this interference can greatly vary depending on which execution flow is taken by these other applications (some of these flows may execute for a very long time but using only a few cores whereas others may use many cores and execute for a very short time), the interference analysis could consider every combination of execution flows from every application in the system, but this would be prohibitively expensive in terms of the computation time.

II. The Envisioned Approach

The objective of this work is to come up with a framework to enable the computation of the response time of every application in the system in polynomial time. Our approach intends to compute a single DAG of independent periodic servers for each application in the system, and define a mapping rule to decide which ready sub-task must be assigned to which server at run-time. These servers and the mapping rule must be defined such that, for any execution flow taken by the application at run-time, enough cpu-budget is provided to the sub-tasks of the currently executed flow so that the application completes by its deadline and respects all the precedence constraints. Specifically, our envisioned solution will work as follows:

Step 1) We will propose an efficient mapping rule to arbitrate the assignment of the sub-tasks that are ready-for-execution to the running servers.

Step 2) *For each execution flow (DAG of sub-tasks) of the application under analysis*, we will compute a graph of servers where each node of the graph (i.e. each server) has a cpu-budget and each edge models a precedence constraint between two servers. This graph of servers together with the mapping rule defined in Step 1, will provide the following guarantee at run-time: no matter how the servers are scheduled on the cores, *as long as their precedence constraints are not violated*, the total budget provided by *the k'th job of every periodic server* ($\forall k > 0$) — and the way this budget is distributed between the sub-tasks by the mapping rule — provides enough cpu-budget for all the sub-tasks of the currently-taken execution flow to complete.

Step 3) All the graphs of servers computed in Step 2 for every execution flow of the application will be merged into a single graph of servers. That resulting graph will be defined such that it preserves the guarantee defined in Step 2, i.e. it provides enough cpu-budget to the sub-tasks of any execution flow.

Step 4) The resulting graph of servers computed in Step 3 will be converted into a set τ of asynchronous periodic constrained-deadline independent real-time tasks by using techniques such as [5], where each real-time task τ_i is defined by four parameters: an offset O_i, an execution time (cpu-budget) C_i, a period T_i, and a deadline $D_i \leq T_i$.

Step 5) Finally, by repeating Step 1 to Step 4 for each parallel application in the system, we will end up with several sets of real-time tasks (one set for each application). We can then group all these task sets in a super-set $\bar{\tau}$ of asynchronous periodic constrained-deadline independent real-time tasks. This task model has been widely studied in the past and many efficient scheduling algorithms have been proposed. Our solution will then investigate which of these scheduling algorithms can successfully schedule the resulting set $\bar{\tau}$ by using their respective schedulability tests.

Acknowledgments

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project(s) FCOMP-01-0124-FEDER-037281 (CISTER), FCOMP-01-0124-FEDER-020447 (REGAIN); by the European Union, under the Seventh Framework Programme (FP7/2007-2013), grant agreement nr. 611016 (P-SOCRATES).

References

[1] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *the 31st RTSS*, 2010, pp. 259–268.

[2] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *the 32nd RTSS*, 2011, pp. 217–226.

[3] S. K. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *the 33rd RTSS*, 2012, pp. 63–72.

[4] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[5] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Techniques optimizing the number of processors to schedule multi-threaded tasks," in *the 24th ECRTS*, July 2012, pp. 321–330.