Efficient Execution of Dependent Tasks on Many-Core Processors

Hamza Rihani, Claire Maiza, Matthieu Moy
Univ. Grenoble Alpes
CNRS, VERIMAG, F-38000 Grenoble, France
Email: {first}.{last}@imag.fr

I. Introduction

The increasing performance requirements of safety-critical real-time embedded systems made traditional single-core architectures obsolete. Moving to more complex many-core systems requires new techniques and tools for the certification of the embedded software. Timing and functional behaviours are subject to specific requirements of certification guidelines such as D-178B/C for avionics and ISO26262 for automotive systems. Determinism and predictability of such systems are a major challenge. Running tasks should be known in advance which makes the static and non-preemptive scheduling a suitable approach to reach an optimal execution with a guarantee of a certain degree of determinism. Recent work on mapping and scheduling problems [1], [2], [3] consider a known value (or a set of possible values) of the Worst-Case Response Time (WCRT) and computes a mapping that optimizes a predefined cost function. When the response time analysis is too pessimistic, the static scheduling may introduce an idle time which reduces the core utilization.

Scheduling techniques must rely on tight estimations of the WCRT which in turn depends on co-runner tasks. However, in order to obtain a tight upper-bound on the response time, a mapping and scheduling should be known in advance. Indeed, the response time is highly influenced by the co-runner tasks. Concurrent accesses to the same shared resource may introduce interferences that should be accounted for in the response time analysis. The search for an optimal scheduling with a tight WCRT analysis that includes the shared resource interferences is a challenging open problem.

II. The Open Problem

We illustrate the open problem on a Synchronous Data-Flow (SDF) application. SDF languages such as Lustre [4] offer an efficient programming paradigm that, using a certified compiler, can produce deterministic sequential code. An SDF application is represented with a task dependency graph, where the amount of exchanged data among the tasks is deterministic and known in advance. We consider the application to be running on a multi/many core architecture with a banked shared memory. Figure 1 represents an example of a node in an SDF application. The node increases the values of an input array by 1. Among the challenges while parallelizing such applications, we need to i) specify a task mapping that optimizes a certain cost function; ii) specify a scheduling per processing element that respects dependencies; iii) find a tight estimate of the response time that takes into account the interferences from co-runners. There exists several solutions for the mentioned points when taken individually. However, the connexion and the interaction among them remains as an open problem.

```
1 const n=1000;
2 node N (in_data: int^n) returns (out_data: int^n);
3 var N: int^n;
4 let
5     N = 1^n; -- array of n values of 1
6     out_data=in_data+N;
7 tel
```

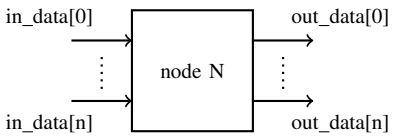

Fig. 1: Example of a Lustre node and its high level graphical representation

In this context, Figure 2 represents a potential solution for an efficient execution of such applications. The proposed technique relies on an iteration between a scheduling/mapping function and the WCRT analysis. Existing approaches may be used in each step but must interact altogether. In the following, we highlight open problems regarding such composability:

1) **Code Generation:** A code generator transforms the example given in Figure 1 into a low level language (such as C language) which in turn is compiled to run on a target architecture. We consider that `in_data` (produced by a previous node) is present in the local memory bank. After computation, `out_data` is copied to the next nodes' local memory banks. Several execution models can be applied: 1) A single phase execution model where the output is "sent" to the next node as soon as it is ready. 2) A structured execution model with an *execution phase* and a *replication phase*. The

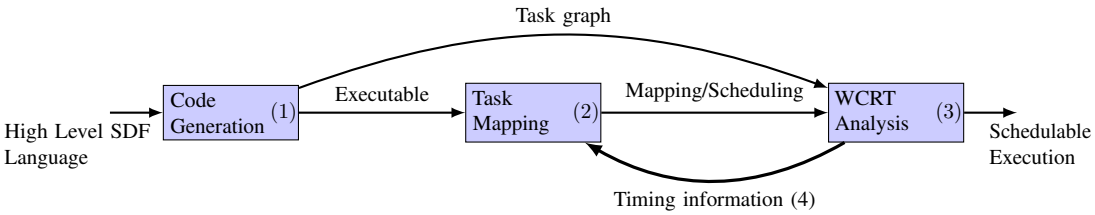

Fig. 2: High level work flow of a complete WCRT/Scheduling interaction

execution phase uses only the local memory bank to compute and store the output data. Then, a *replication phase* is added to copy the values of `out_data` into their memory destinations. The number of shared resource accesses may be significantly increased, but with a good scheduling of the execution and replication phases the interference may be reduced. The choice of the execution model affects the results in the next steps. It is not clear whether it is possible to recognize at this stage which execution model is more efficient by analyzing the structure of the application.

2) **Task Mapping:** We are interested in the optimization of the overall response time according to an estimated WCRT of each task. In memory intensive applications, the scheduler may consider minimizing the interference by avoiding concurrent accesses to the same shared resource. Each task is modeled by its release date, response time, deadline, and period. The single phase execution model may use traditional scheduling techniques to respect all the deadlines. The structured execution model offers more flexibility by co-scheduling communications and computations such that the interference is reduced [5]. However, we still need to investigate the efficiency of this method w.r.t a single-phase execution model where the number of accesses is not increased but the interference can be pessimistically overestimated.
3) **WCRT Analysis:** The WCRT analysis must be able to derive tight upper-bounds on the interference due to concurrent accesses to shared resources, and therefore estimate the response times and release dates for all tasks. This requires a task mapping and scheduling as well as the constraints on the task dependencies. The execution model given by the code generation step can highly influence the analysis. In the case of a structured execution model, the upper bounds on the shared resource interferences in the execution and replication phases may be too pessimistic due to pessimistic assumptions on the arbitration policy. This may as well result in a larger estimation on the overall interference per task depending on the application's topology. As a result, one may run the analysis on both execution models and retain the one with the best overall response time.
4) **WCRT/Mapping Iteration:** Depending on the chosen task mapping and scheduling per processing elements, the response times must be re-estimated by taking into account the interference in order to check that all constraints are respected and potentially reduce idle times. However, the optimality obtained from step (2) may be lost due to the updated timing information. A possible solution relies on constraint programming languages. A solver finds a satisfiable task mapping that optimizes the cost function. The WCRT Analysis and a schedulability test can then compute a tighter estimation by taking into account potential interferences. New timing constraints are added to the model and the search stops when no better solution can be found.

Since the mapping problem is NP-hard, it is hard to prove that the iteration done in $4)$ converges toward an absolute optimal solution (assuming its existence). In the absence of such guarantee, one can iterate for a certain amount of time and choose the best solution. This interaction remains as an open problem.

ACKNOWLEDGMENT

This work was funded by the grant CAPACITES (PIA-FSN2 n°P3425-146798) from the French *Ministère de l'économie, des finances et de l'industrie*.

REFERENCES

[1] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Scheduling of mixed-criticality applications on resource-sharing multicore systems," in *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, Sept 2013, pp. 1–15.
[2] W. Puffitsch, E. Noulard, and C. Pagetti, "Mapping a multi-rate synchronous language to a many-core processor," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pp. 293–302.
[3] J. Walter and W. Nebel, "Energy–aware mapping and scheduling of large–scale macro data–flow applications," in *1st International Workshop on Investigating Dataflow in Embedded Computing Architecture*, 2015.
[4] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language lustre," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, Sep 1991.
[5] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo, "Memory-processor co-scheduling in fixed priority systems," in *23rd ACM International Conference on Real-Time Networks and Systems (RTNS)*, Lille, France, November, 2015.