Monitoring On-line Timing Information to Support Mixed-Critical Workloads

Angeliki Kritikakou, Olivier Baldellon, Claire Pagetti, Christine Rochange, Matthieu Roy, Fabian Vargas

▶ **To cite this version:**

Angeliki Kritikakou, Olivier Baldellon, Claire Pagetti, Christine Rochange, Matthieu Roy, et al.. Monitoring On-line Timing Information to Support Mixed-Critical Workloads. IEEE Real-Time Systems Symposium 2013, Dec 2013, Vancouver, Canada. pp.9-10. hal-01015455

HAL Id: hal-01015455
https://hal.science/hal-01015455

Submitted on 26 Jun 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Monitoring On-line Timing Information to Support Mixed-Critical Workloads

A. Kritikakou*, O. Baldellon†, C. Pagetti*, C. Rochange‡, M. Roy†, F. Vargas§
*ONERA, †LAAS/CNRS, ‡Université de Toulouse, France
§PUCRS, Brazil

I. Introduction and Motivation

Multi-/many-core architectures provide a drastic increase in computation power, enabling the simultaneous execution of several tasks on the system. Yet, in critical embedded systems, e.g. aeronautical systems, the uncertainty of the non-uniform and concurrent memory accesses prohibits the full utilization of the system potentials. To ensure safety, such systems impose static Worst Case Execution Time (WCET) estimations; memory access times are upper bounded considering a fully congested memory bus, leading to safe but almost unusable bounds and to overwhelmingly conservative schedules [1].

Existing techniques for multicore systems address the optimization of the scheduling objectives, e.g. processor utilization, when the task set is schedulable at least in the highest criticality level. In the mixed-criticality theoretical model of [2], multiple WCET values, related to different criticality levels, are considered for each task. The lower the level, the more unsafe is the WCET bound. In the highest level, the value is trustful. In [3], [4], [5], all tasks are started at the low criticality level. When a task has not finished on time, its level is increased and the less critical tasks are dropped. When the WCET of the highest level is larger than the deadline, the problem is considered as unschedulable.

Let us now consider a critical task with a WCET below the deadline, when it is the only task executed on the system, i.e. in isolation. We advocate that in this case, the safety of the critical task can be ensured, while some less critical tasks are run in parallel on other cores. Fig. 1 depicts the aforementioned problem, where one critical task T_C and one less critical task T_1 are considered on two cores. When both tasks are executed, the WCET of T_C is estimated above its deadline D, and thus the problem is unschedulable, as depicted in Fig. 1(a). However, if T_C is executed in isolation, it is schedulable, as shown in Fig. 1(b). In contrast with existing approaches, our methodology is capable of scheduling the T_C by considering two execution scenarios. Initially, both tasks are executed on the system. Monitoring points are used to on-line monitor the real execution time of T_C and decide switching to isolation execution of T_C, as shown in Fig. 1(c).

In this work, we propose an approach to improve core utilization by running several tasks in parallel and guarantee the critical task safety.

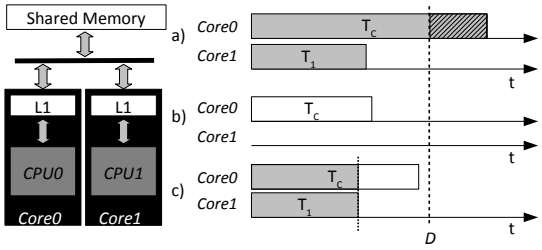

Fig. 1. Scheduling based on WCET when are considered for execution (a) both tasks, (b) only the critical task and (c) proposed hybrid approach.

II. Target System

Our application domain is described by a task set with a single critical task T_C, which is schedulable in isolation, but non-schedulable when executed with other less critical tasks. The WCET of the critical task is computable, i.e. the whole code can be unfolded into a bounded tree. We represent binary code of the application under study as a Control Flow Graph (CFG) [6]. Each node represents a basic block and edges connect nodes that can be executed in sequence. Our target platform is a time-predictable bus-based multicore system. More precisely, we consider several processing IP-cores on an FPGA platform with hardware monitoring capabilities [7]. We assume a number of cores equal to the number of tasks, which can then be statically co-scheduled.

III. Proposed methodology

Our methodology is two-fold: i) Off-line, we analyze the CFG and safely estimate the remaining WCET at several points of the T_C, under two scenarios: *isolation*, where the critical task is executed alone, and *maximum load*, where the maximum interference from co-running less critical tasks exists. ii) On-line, we use a hardware monitor to observe the real execution time of T_C and to check whether a risk exists that the critical task misses its deadline due to system overload. If so, the less critical tasks are stopped and the execution is continued in isolation scenario. The proposed approach is schematically depicted in Fig. 2.

A. Off-line critical task analysis

As we consider critical tasks, our approach uses safe static WCET analysis [8]. From the annotated CFG, an Integer Linear Programming (ILP) formulation is written to express

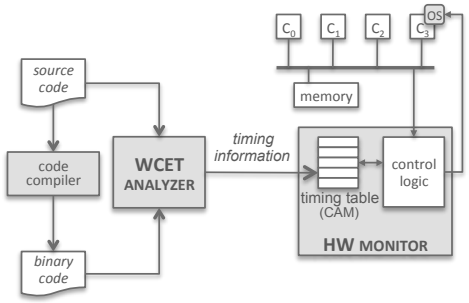

Fig. 2. Overview of the proposed approach

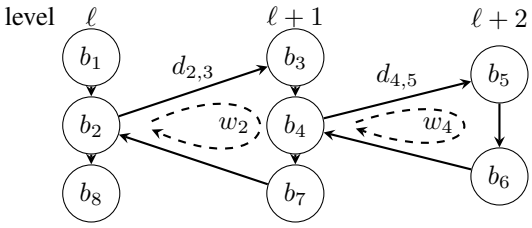

Fig. 3. CFG with nesting levels and partial WCET

the program execution time as the sum of the individual times of basic blocks (nodes) weighted by their execution counts. This expression is maximized to find the required WCETs, with a number of constraints that reflect flow facts, e.g. loop bounds and unfeasible paths. This approach is adapted to our requirements: by modifying the ILP formulation appropriately, we are able to compute partial WCETs, i.e. WCETs between two monitoring points in the code, and the remaining WCET from one monitoring point until the end of the program. The monitoring points are set at the first instruction of each basic block of CFG, at this project stage. Ideally, for each monitoring point b, the remaining WCET until the end of the program should be estimated. However, as a point may be visited several times, e.g. when it belongs to a loop, the number of point instances to be stored is equal to the number of loop iterations. This highly increases storage requirements. Hence, we need a hybrid approach: the initial remaining WCET until the end of the program ($\mathsf{RWCET_{iso}}$) of the first point instance is computed off-line; then at runtime, the $\mathsf{RWCET_{iso}}$ is updated each time the point is encountered, as described in Section III-B2. For this purpose, the information required for point b is: (i) the nested level in the CFG, $level_b$, used to identify the reference remaining time; (ii) if b is a loop header, the WCET of the loop body (w_b); (iii) the WCET from the inner loop header to point b (d_b). Figure 3 depicts such parameters. The timing values (ii) and (iii) are always estimated for the *isolation* scenario to guarantee the T_C safety.

B. On-line execution

1) Hardware monitoring scheme: Our approach relies on hardware monitoring, as proposed in [9]. The scheme observes the instruction addresses from the memory. We assume that cores do not have instruction caches at this project stage. An address that has been monitored indexes a local associative memory (CAM), where $level_b$, $\mathsf{RWCET_{iso}}$, w_b and d_b are stored. The CAM returns the timing information of the monitoring address to be used by the on-line control mechanism. Whenever the on-line control mechanism determines that the non-critical tasks should be suspended, the hardware monitor interrupts the operating system.

2) On-line control mechanism: Our on-line low-overhead control mechanism implemented in hardware guarantees the safety of the T_C execution. Initially, both the critical and less critical tasks are executed on the multicore system. At each monitoring point b, the on-line control mechanism decides whether switching from the *maximum load* scenario to the *isolation* scenario is required. This decision is taken based on the information retrieved from the CAM memory in each monitoring point b. The $\mathsf{RWCET_{iso}}$ is on-line computed and updated in CAM. When the CFG is traversed in the forward direction, the $\mathsf{RWCET_{iso}}(b)$ is computed as $\mathsf{RWCET_{iso}}(level_b) - d_b$. When the CFG is traversed backward to a loop header b, the $\mathsf{RWCET_{iso}}(b)$ is computed as $\mathsf{RWCET_{iso}}(b) - w_b$ and the new value is updated to the CAM memory. Scenario switching is triggered at monitoring point b when the condition of Equation 1 holds: $\mathsf{ET}(b)$ denotes the monitored real execution time of T_C until point b, $\mathsf{WCET_{mon}}$ is the maximum WCET between two successive monitoring points and t_{Over} is the cost of our recovery mechanism. The latter includes the time to monitor $\mathsf{ET}(b)$, the time for the hardware logic to access the CAM memory and to decide, and the time for the operating system to suspend the non critical tasks. Intuitively, Equation 1 means that the critical task might miss its deadline in the current (*maximum load*) scenario, but by switching to the *isolation* scenario in point b, the deadline can still be met.

$$\mathsf{ET}(b) + \mathsf{RWCET_{iso}}(b) + \mathsf{WCET_{mon}} + t_{\mathrm{Over}} > D \quad (1)$$

Our future work is the implementation of the proposed offline analysis tool and the online hardware control in the FPGA platform. We will also extend our approach to systems with several critical tasks and larger (scheduled) task sets.

REFERENCES

[1] R. Wilhelm *et al.*, "Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems," *TCAD*, vol. 28, no. 7, 2009.

[2] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *RTSS*, pp. 239–243, 2007.

[3] J. H. Anderson, S. K. Baruah, and B. B. Brandenburg, "Multicore operating-system support for mixed criticality," in *WMC*, April 2009.

[4] M. S. Mollison *et al.*, "Mixed-criticality real-time scheduling for multicore systems.," in *CIT*, pp. 1864–1871, 2010.

[5] H. Li and S. Baruah, "Global mixed-criticality scheduling on multiprocessors," in *ECRTS*, pp. 166–175, 2012.

[6] K. D. Cooper *et al.*, "Building a control-flow graph from scheduled assembly code," Tech. Rep. TR02-399, Rice University, 2002.

[7] M. Paolieri, E. Quinones, F. Cazorla, G. Bernat, and M. Valero, "Hardware support for wcet analysis of hard real-time multicore systems," in *Int'l Symposium on Computer Architecture (ISCA)*, 2009.

[8] R. Wilhelm *et al.*, "The worst-case execution-time problem: overview of methods and survey of tools," *TECS*, vol. 7, no. 3, 2008.

[9] D. Silva *et al.*, "An intellectual property core to detect task schedulling-related faults in rtos-based embedded systems," in *IOLTS*, 2011.