Using NPS-F for Mixed-Criticality Multicore Systems

Konstantinos Bletsas and Stefan M. Petters
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
ksbs,smp @isep.ipp.pt

1. MOTIVATION

Hard real-time multiprocessors scheduling has recently seen the flourishing of semi-partitioned scheduling algorithms – a category of scheduling schemes that combine elements of partitioned and migrative scheduling to allow more efficient processor usage, while providing improved schedulability guarantees at the same time. Yet, so far, semi-partitioning has not made any inroads into mixed-criticality scheduling. We aim to address this by proposing a way of combining mixed-criticality scheduling with semi-partitioning. Specifically, we adapt the NPS-F scheduling algorithm [1] for this purpose. The timeslot-based dispatching of NPS-F allows for fairness and responsiveness for lower-criticality workloads with no detriment to the schedulability guarantees of high-criticality tasks.

NPS-F can be outlined as follows. Initially, tasks are partitioned to servers offline, using bin-packing. These servers are then mapped to one or more processors each (using non-overlapping time windows, for each server that employs multiple processors), creating a sort of cyclic executive which repeats every S time units – the timeslot. At run-time dispatching inside each server is dynamic, using EDF. The share of the timeslot S reserved for each server is static (computed offline) and depends on its workload. The principle guiding the sizing of each server is that its share of the timeslot should be sufficiently large (according to schedulability analysis) such that no deadlines may be missed by the tasks it serves.

2. NPS-F FOR MIXED CRITICALITIES

Our approach for adapting NPS-F to mixed-criticality workloads consists in partitioning the high-criticality tasks ("H-tasks") over m non-migrating servers, each server mapped to one corresponding processor. The remaining capacity on all processors (the shares of the timeslot that remain unassigned) is "recycled" for mapping servers for the non-critical workload ("L-tasks").

Two system operation modes exist: "normal" mode and "high-criticality" mode. Each H-task τ_i has two different estimates of its WCET: C_i^L, which is considered sufficient, but lacks proof and C_i^H, which is provably always safe and possibly much greater than C_i^L. For L-tasks, only C_i^L is determined. Under "normal" mode, every task τ_i (irrespective of its criticality) executes and both critical and non-critical tasks are guaranteed to meet their deadlines as long as they run for no more than their low-criticality estimate C_i^L. However, as soon as any H-task overruns its C_i^L, the system switches to "high-criticality" mode: non-critical workload is idled and the system switches to partitioned EDF scheduling of H-tasks only, assuming a WCET of C_i^H for each H-task. In high-criticality mode, and during the mode transition, the deadlines of all H-tasks must be met, which poses challenges.

A naive approach would (i) partition the H-tasks such that they are schedulable under uniprocessor EDF, considering the overly conservative estimates C_i^H (to meet deadlines in high-criticality mode) and, for normal mode execution, would (ii) assign budgets to the respective servers so that they meet deadlines with WCETs of C_i^L. However, this may lead to missed deadlines during mode transition.

A conservative approach would, instead size budgets for H-task servers according to their C_i^H. However, this would be too inefficient for normal mode operation, because it would limit the processor capacity available for scheduling L-task servers. Ideally, we would like to analytically identify the "sweet spot" between these two extremes, so as to (i) minimise the processor capacity used for H-task servers (i.e. maximise the capacity available for L-task servers), while (ii) ensuring that no H-tasks may ever miss deadlines (even when a mode transition is triggered).

3. APPROACH OUTLINE

The approach taken centers around (i) identifying (after an overrun has been detected in a server) the amount of execution which may need to be executed before the respective task deadlines and (ii) sizing the servers such that the point of detection is sufficiently early to execute the remaining workload in time. Issues to solve are (i) identifying the critical instant, which leads to the latest detection of an overrun of any task and (ii) sizing the servers such that the regular load can be executed beforehand. The particular challenge here is that these two issues are not independent.

Pen and paper exercises with a number of case studies indicate that most servers can be kept at substantially lower utilization than what is required if straight C_i^H is considered. The benefit of the proposed approach is that it maintains the isolation to L-tasks, in most cases with a larger share utilization than what would otherwise be possible, leading to a more even and fair distribution of resources.

Acknowledgements: Work 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 SMARTS project, ref. FCOMP-01-0124-FEDER-020536.

4. REFERENCES

[1] K. Bletsas and B. Andersson, "Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound," *Journal of Real-Time Systems*, vol. 47, no. 4, pp. 319–355, 2011.