Online Time Interference Detection in Mixed-Criticality Applications on Multicore Architectures using Performance Counters

Stefano Esposito, Massimio Violante
DAUIN - Politecnico di Torino
Torino, Italy
{stefano.esposito,massimo.violante}@polito.it

Marco Sozzi, Marco Terrone, Massimo Traversone
Leonardo-Finmeccanica
Italy
{marco.sozzi,marco.terrone,massimo.traversone}@finmeccanica.com

***Abstract*— In this paper a novel technique is proposed for online detection of timing interference in multicore architectures. The technique is aimed at mixed-criticality workloads. This paper describes a method to use hardware performance counters to detect such misbehaviors. Experimental data is gathered, showing the viability of this method. The method can be used as safety-net in several scheduling approaches.**

Keywords*— *multicore processing; mixed-criticalities; fault detection; performance counters; safety critical applications; hard real-time

I. INTRODUCTION

A mixed-criticality system is one in which two or more tasks with different assurance levels, or criticalities, share the same hardware resources. In a multicore-based system, such resources include shared memory and bus. For certification purposes, all applications sharing resources should be designed at the same assurance level as the most critical one. This means that the multicore systems may suffer of under-utilization. To solve this problem, several scheduling algorithms have been proposed to ensure schedulability of a system with mixed-criticalities tasks running on multicore architectures. The main contribution of this paper is a method for online detection of temporal interference in multicore-based mixed-criticality applications. In section II we present a brief overview of the most relevant works. In section III we describe the proposed method. Section IV presents the experimental results. Conclusions are in Section V.

II. RELATED WORKS

Since [1], several works were presented on the topic of mixed criticalities applications. The basic idea in [1] was that the Worst Case Execution Time (WCET) of a task is computed with pessimistic assumptions for high assurance levels. A scheduling algorithm was proposed to exploit this observation. The scheduling algorithm was extended in the following years [2][3]. Starting from [4], cited approaches were extended to include the multicore case. Multicore architectures are of particular interest for avionics. Current Integrated Modular Avionic (IMA) approach is to use the same processing unit to implement different functions using time multiplexing strategies [5]. Using multicore would allow to integrate tasks that can run in parallel on the same processing unit. Use of multicore in safety-critical hard real-time systems can use different approaches [5]. Some efforts have been done from an architectural point of view [6] [7]. Several other works focused on the WCET analysis in multicore-based systems [8][9]. Authors of [10] proposed performance counters as safety-net, although they do not present experimental data.

III. FAULT DETECTION USING PERFORMANCE METRICS

The proposed method assumes that a feasible scheduling for the system has been implemented. The method has been developed to detect temporal interferences. The proposed method does not require online monitoring of the execution time as a watchdog timer: any performance metric correlated to the execution time can be used. If performance counters for the selected metric are included in the task context, several monitored tasks can share the same core.

A. Offline phase

We consider the selected metric as a random variable X. A detection threshold D is defined in (1). Any execution with a metric value above D is considered affected by a fault.

$$P(X \leq D) = \mathrm{C}_1 \tag{1}$$

C_1 is the desired level of confidence that a task with a metric lower than D is fault-free. The probability can be determined by fitting profiling data to a known distribution.

B. Online phase

The selected metric is monitored through performance counters, which trigger an Interrupt Request (IRQ) when the measured metric is above the threshold D, triggering a recovery action that can be either *graceful degradation* – the scheduling is modified so that the monitored task is scheduled alone, while tasks that could be scheduled in parallel are scheduled after its completion – or *hot-standby spare*.

C. Metric selection

Metric selection is dependent on the application and on the hardware architecture. In general, the metric should be selected to measure the penalty that the monitored task could suffer when other tasks use shared resources in an unexpected way. Since the goal is to detect temporal interference, the selected metric should be correlated to execution time. Moreover, the monitoring of the metric should not introduce temporal interference with any task.

IV. Experimental Setup and Results

The proposed method has been evaluated on a dual-core and a quad-core Cortex-A9. A commercial type-1 hypervisor was used to implement partitioning and scheduling. A synthetic benchmark composed of memory operations and bubble-sort was designed to stress the memory hierarchy. Monitored task is periodic and performs memory operations, while the other tasks execute bubble-sort with the same period. The performance metric to monitor was selected among those measurable using the Performance Monitor Unit (PMU), which is a component of each core that allows counting several events concerning performance. *Data cache dependent stall cycles* (DCSC) counts the cycles during which the core is stalled waiting for data; it is a good metric for the proposed method. Figure 1 presents data collected on a quad-core processor in 15,000 runs, showing that DCSC can detect interference; results are similar on a dual-core processor. The proposed method was applied with $C_1 = 0.9985$ to compute D. The benchmark was executed under different scenarios, 15,000 times in each. Interfering applications were changed to simulate a software bug causing bus saturation. Figure 2 shows effects on Xilinx Zynq (dual-core) and on NXP i.MX6Q (quad-core). Table 1 shows the ratio of executions for which the recovery action was triggered. It is worthy to mention that when the detection threshold is not crossed, the bug did not cause deadline miss.

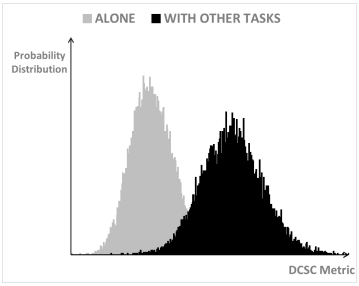

Figure 1. Probability distribution of the selected metric on a quad-core architecture, when running alone and when running at the same time as other tasks (one task per core).

V. Conclusions

A new online temporal interference detection technique has been presented, showing its effectiveness with preliminary experimental data. Results show that the technique is able to detect timing interference. Although cycle counters could be used as interference metrics, this measure is often more useful when used with watchdog timers, which can be complementary to the proposed method.

Acknowledgment

The research was partially supported by the ARTEMIS Joint Undertaking project in the Innovation Pilot Programme "Computing platforms for embedded systems" (AIPP5) under grant agreement n. 621429 (project EMC^2). Thanks to Serhiy Avramenko for the valuable insights.

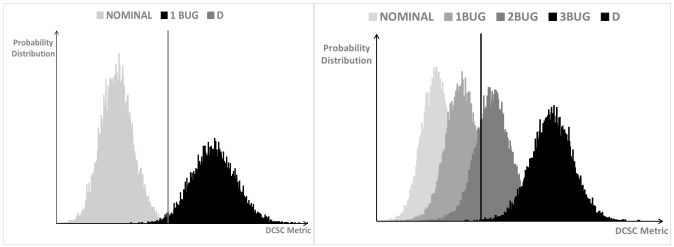

Figure 2. Fault effects. The vertical line is the detection threshold.

TABLE 1. RECOVERY ACTION TRIGGER RATIO

Case	Dual Core	Quad Core
Nominal	0.28%	0.19%
1 Affected Task	99.06%	11.12%
2 Affected Tasks	-	75.65%
3 Affected Tasks	-	99,93%

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," *Proc. - Real-Time Syst. Symp.*, pp. 239–243, 2007.

[2] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," *Proc. - Euromicro Conf. Real-Time Syst.*, pp. 147–155, 2008.

[3] S. K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-Criticality scheduling of sporadic task systems," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6942 LNCS, pp. 555–566.

[4] J. Anderson, S. Baruah, and B. Brandenburg, "Multicore operating-system support for mixed criticality," in *Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009.

[5] J. Nowotsch and M. Paulitsch, "Leveraging Multi-Core Computing Architectures in Avionics," *2012 Ninth Eur. Dependable Comput. Conf.*, pp. 132–143, 2012.

[6] J. Barre, C. Rochange, and P. Sainrat, "An architecture for the simultaneous execution of hard real-time threads," *Proc. - 2008 Int. Conf. Embed. Comput. Syst. Archit. Model. Simulation, IC-SAMOS 2008*, pp. 18–24, 2008.

[7] S. Avramenko, S. Esposito, M. Violante, M. Sozzi, M. Traversone, M. Binello, and M. Terrone, "An Hybrid Architecture for Consolidating Mixed Criticality Applications on Multicore Systems," in *2015 IEEE 21st International On-Line Testing Symposium*, 2015, pp. 26–29.

[8] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury, "Timing Analysis of Concurrent Programs Running on Shared Cache Multi-Cores," *2009 30th IEEE Real-Time Syst. Symp.*, pp. 57–67, 2009.

[9] S. Girbal, A. Grasset, E. Q. U. I. Nones, S. Yehia, and F. J. Cazorla, "On the Evaluation of the Impact of Shared Resources in Multithreaded COTS Processors in Time-Critical Environments," vol. 8, no. 4, pp. 1–25, 2012.

[10] J. Nowotsch, M. Paulitsch, A. Henrichsen, W. Pongratz, and A. Schacht, "Monitoring and WCET analysis in COTS multi-core-SoC-based mixed-criticality systems," *Des. Autom. Test Eur. Conf. Exhib. (DATE), 2014*, pp. 1–5, 2014.