Closed Loop Controller for Multicore Real-Time Systems

Johannes Freitag(✉) and Sascha Uhrig

Airbus, Munich, Germany
johannes.freitag@airbus.com

Abstract. In critical and hard real-time applications multicore processors are still not used very often. One of the reasons is the lack of timing predictability or the high Worst Case Execution Time (WCET) overestimation caused by the use of shared resources. Nevertheless, multicore processors can significantly increase system integration density also in critical and hard real-time applications.

We present a Closed Performance Control Loop that enables a standalone WCET estimation of a hard real-time application and execution on a multicore system concurrently to other applications. The advantage of our proposal is that it is transparent and non-intrusive to the critical application. Moreover, it is implemented as an external safety net and no additional software functionality on the multicore is required. The previously presented Fingerprinting approach to measure an application's performance is used as sensor element, extended by a Pulse Width Modulated core thwarting technique and two different control algorithms are combined to a Closed Control Loop.

Keywords: Embedded multicore systems · Critical systems
Safety net · Real-time systems

1 Introduction

Future avionic applications will require higher computation performance while at the same time a reduction in space, weight and power is needed. These needs are shown for example in the concept of the Airbus *Vahana*, *Pop-up*, or *CityAirbus* [1] aircrafts which will be ultra lightweight electrical helicopter-style vehicles providing novel autonomous urban transportation. In comparison to current aircrafts the avionic systems must be much smaller and lightweight while at the same time provide sufficient performance to compute not only the flight control data similar to current aircrafts but additionally compute the complex algorithms for autonomous flying, navigation, and collision avoidance. One solution for these demands is the consolidation of flight applications, currently running on multiple single core computers, on a small number of multicore processors. Furthermore, legacy applications shall be reused without major modifications.

Even though first ideas of the regulations on how to apply multicore systems to avionics are presented in the CAST-32 position paper and its follow-up CAST-32a [2], both authored from the Certification Authorities Software Team (CAST), concrete design details are still open. One of the major challenges in this context is the interference between applications since theoretically one application can compromise another one, at least in the timing domain. Accordingly, an essential requirement for certification is a clear and reliable isolation of safety-critical applications that needs to be demonstrated to the certification authorities.

The *Fingerprinting* technology presented in [3] allows non-intrusive tracking of an application's progress. Moreover, it allows continuous online quantification of an application's slowdown caused by interferences on shared resources compared to the stand-alone execution of the same application. Starting from this ability, we developed a closed loop controlling mechanism that keeps the current slowdown of an application inside given acceptable boundaries compared to stand-alone performance. This is done by thwarting the execution of other cores if necessary, in order to reduce interferences. Consequently, an estimated Worst Case Execution Time (WCET) of the stand-alone execution can hold for the multicore execution with the same acceptable bounds. The fingerprinting technology is targeting applications that are executed in a periodical way which is a typical feature of applications used in aircrafts.

The contributions of this paper are

– adjustable performance reduction techniques based on a Pulse Width Modulated (PWM) signal,
– a complete closed control loop system controlling an application's performance.

The remainder of this paper is organized as follows. Section 2 provides an overview of mature techniques and related work. The closed loop control techniques including a background on the Fingerprinting is described in Sect. 3 while the evaluation is presented in Sect. 4. The paper concludes with Sect. 5 including an outlook on future work.

2 Related Work

The use of multicore systems in avionic applications is still not wide spread. One reason is the difficulty to obtain suitable Worst Case Execution Time (WCET) estimates since application performance can theoretically drop significantly if multiple cores (i.e. applications) are sharing bus and memory [4]. Furthermore, it is not possible to identify all interference channels on COTS multicore processors [5]. Therefore, a WCET analysis on possible worst case scenarios leads to a high WCET overestimation (WCET to average execution time ratio) for current COTS MPSoCs. Hence, the performance gain of the multicore is neglected.

There exist several approaches to limit or even control the interferences between high and low critical tasks on multicore systems to relax the worst case scenario and, hence, improve WCET analysis results. Most of them focus on task or even thread granularity and are integrated into the scheduling of the system. The main idea of these approaches is counting e.g. bus accesses and limiting them by suspending the corresponding thread. Examples of such approaches are presented in [6–9]. An overview of these and other approaches is given in [10]. Even though these approaches are interesting for newly developed applications, they are not suitable for combing multiple legacy single core avionic applications on a multicore processor because the legacy applications or the underlying operating system would either have to be modified completely, which leads to a high effort in certification, or restrict the applications in a way that the performance gain of the multicore is neglected.

A previous approach for characterizing an application's execution is presented in [11]. It is used in high performance systems to predict an application's future behaviour and needs for adjusting architectural parameters for performance optimizations. It is not related to embedded real-time systems but successfully uses a similar, but intrusive, technology for tracking an application's performance.

The use of feedback controllers in the utilization of real-time systems is not novel. For example, a closed loop controller is used in [12] for dynamic resource allocation and power optimization of multicore processors. An example for closed loop control in a real-time scheduler is presented in [13,14] while a controller for thermal control of a multicore processor is introduced in [15]. However, all of these methods require intrusive measurements and no non-intrusive approach for controlling the interferences between cores by an external device has been presented in the past.

3 Closed Performance Control Loop

In the following the basic idea of the *Fingerprinting* approach is briefly described. This *Fingerprinting* is used as the sensor element of the closed control loop. Subsection 3.2 describes the *actuator* to influence the performance of the other cores and, hence, the interferences. The complete closed control loop is presented in Subsect. 3.3. Figure 1 shows the setup of multicore processor and safety net system with the integrated closed control loop. Note that our proposed closed performance control loop does not require any additional software functionality running on the multicore.

3.1 Basic Fingerprinting

During the execution of an application, a flow of instructions is executed. This flow is not homogeneous in terms of type of instructions, source of the instructions, and execution time of instructions. Accordingly, measuring for example the number of executed floating point instructions per time unit will lead to a characteristic curve of an application or a part of the application. If the application is executed several times with the same input parameters the measured curves are very similar (if sample rates greater than 1 μs are applied). For tracking the progress of a known application, its measured curve can be compared to the recorded reference curve.

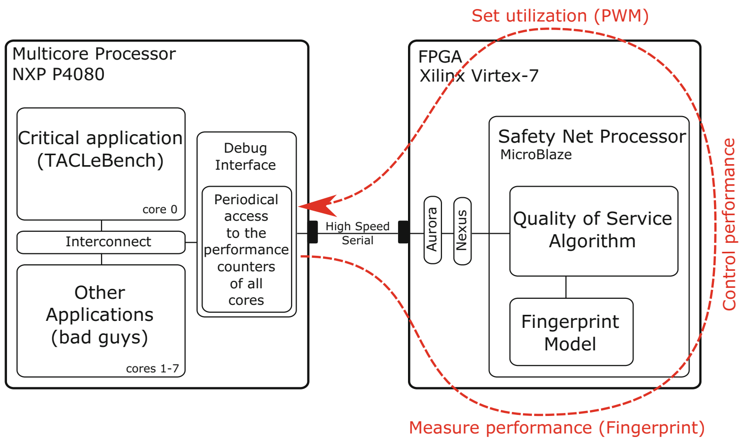

Fig. 1. Hardware setup with closed performance control loop implemented in the safety net system

In case an application executed on a multicore processor suffers from interferences with other applications on the shared memory hierarchy, its progress is slowed down. Slowing down the application will result in a stretched (in time) but shrunk (in the value range) curve. When comparing such a mutated measured curve with the original reference curve, the actual slowdown can not only be identified but also be quantified at any time during execution.

Many current MPSoC (e.g. based on ARM, PowerPC) include performance counters implemented in hardware which can be configured to increment every time a given event is raised. While the amount of events which can be configured is usually more than 100, the amount of counters that can be incremented simultaneously is small (around 4 to 6) [16]. An example of such curves is shown in Fig. 2.

The Fingerprint model is obtained by the execution of the main application several (thousand) times without other applications running in parallel. The performance counter values of the selected events are recorded with the frequency defined by the safety net system (100 µs period in the prototype FPGA case). Afterwards, the recorded characteristics are clustered in order to reduce the amount of curves that are combined into a model. With a *bisecting k-means* algorithm slight variations of the curves are filtered out. As the *bisecting k-means* algorithm does not need a predefined number of clusters, the resulting amount of clusters is depending on the similarity of the curves which is defined by the distance function[1]

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} [|x_i - y_i| > limit] \tag{1}$$

[1] Please note the Iverson brackets: $[P] = \begin{cases} 1 \text{ if } P \text{ is true;} \\ 0 \quad \text{otherwise.} \end{cases}$

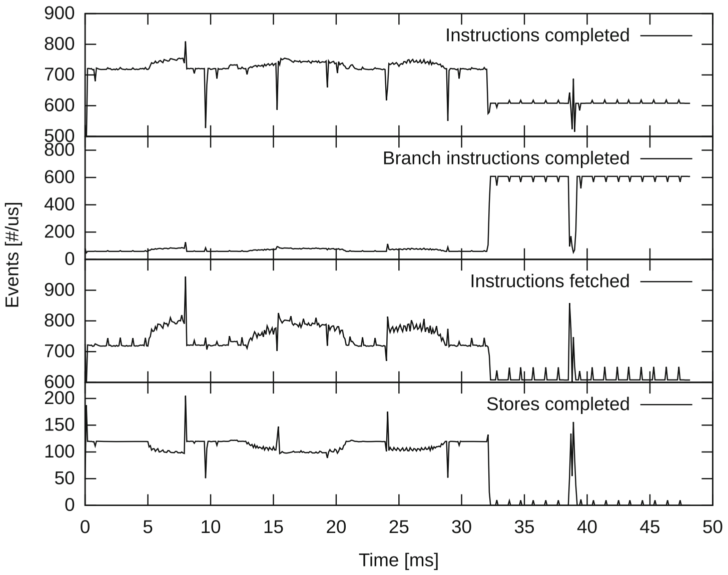

Fig. 2. Measured curves of four event counters when executing an avionic application

with **x** Runtime measurement vector, **y** Centroid vector and n length of the pattern. Finally, the medians of the resulting cluster centroids are combined into a tree model, the *Fingerprint*. The tree data structure is used because it can be accessed in a simple way to allow a fast access when the application is tracked. The root of the tree is the beginning of a new period of the application.

During the actual execution, the Fingerprint safety net system compares the performance counter values with the stored Fingerprint model and the actual execution path along the tree is tracked. In contrast to the generation of the Fingerprint model which can be created offline on a powerful compute node, timing is crucial for the tracking phase.

In case the slowdown of a critical application executed on one core exceeds a given limit (acceptable delay based on the single core WCET), other cores running less critical software are thwarted to reduce concurrency and, hence, increase performance of the critical application.

In summary, the fingerprint safety net approach [3] tracks the application's progress on the basis of characterized behavior of hardware event counters integrated inside the core of a multicore. Periodically reading and resetting such counters results in a curve that is characteristic for an executed application, more specifically, for the progress of that application. When comparing a recorded reference curve with the performance counter values measured online, the current progress with respect to the reference execution can be measured.

3.2 Pulse Width Modulated Interferences

The P4080 multicore system used in this paper provides means to halt and resume cores individually. Both actions can be triggered by messages on the back channel of the trace interface, i.e. by writing to control registers. This means that the *Safety Net* processor (see Fig. 1) is able to control the activity of the cores individually and externally. This way the cores that interfere with the memory accesses of the core under observation with the fingerprinting technique can be halted to not further increase the slowdown of the main application.

To provide a not only digital (on/off) way of setting the performance of the cores, we implemented a (software-based) Pulse Width Modulated (PWM) enabling/disabling of the individual cores, according to the signals from the closed loop controller. By halting and resuming a core, the application on this core can still make progress in contrast to suspending the application completely.

We have chosen a PWM period of 1 ms which is equal to 10 times the 100 µs period used for tracking the application's progress. Hence, we can reduce the performance of cores competing with our main core in steps of 10% from 0 to 100% utilization. For example, a utilization of 60% means that the corresponding core is halted for 0.4 ms and runs for 0.6 ms per millisecond.

3.3 Closed Loop Controller

Two algorithms are used as control element, a simple threshold-based algorithm and a proportional controller. Both techniques affect all concurrent cores synchronously. The threshold-based algorithm disables the concurrent cores when the slowdown of the main application exceeds a given threshold and enables the cores again when the slowdown falls below the same threshold again. The second technique uses a proportional controller and the PWM-based activity control as described in the previous section.

4 Evaluation

We evaluated the effectiveness of the PWM-based activity setting on the main core's performance followed by the evaluation of the full closed control loop system. The selected performance counter values for the evaluation are *Instructions completed*, *Branch instructions completed*, *Stores completed* and *Bus interface unit accesses*. This selection results in very diverse curves which lead to a more robust model. Some paths may look similar in one curve but can be distinguished when taking different curves into account. Furthermore, the *Bus interface unit accesses* curve is an important measurement as it shows the operations that lead to cache misses. Therefore, this curve shows the possible interference hot spots.

4.1 PWM Effectiveness

The main application benchmarks in two different scenarios is used for the evaluation of the effectiveness of the PWM-based activity setting, i.e. the interference control. On the main core, the *TACLeBench benchmark* is executed in every case. The two benchmarks for the competing application cores are

– Read benchmark: This artificial benchmark generates high read traffic on the shared interconnect and the memory by performing read accesses to memory and does not profit from local data caches,
– TACLeBench [17]: A benchmark suite which is application oriented and generates realistic traffic on the shared interconnect and memory and profits from local data caches. Example algorithms used are *JPEG image transcoding routines*, *GSM provisional standard decoder*, *H.264 block decoding functions*, *Huffman encoding/decoding* and *Rijndael AES encoding/decoding*.

The two benchmarks are executed in the two scenarios with and without local caches enabled (L1 instruction and data caches). These scenarios show that the technique also works for very high interference configurations. Furthermore, disabling the caches is relevant for creating the single core WCET as mentioned in Subsect. 3.1. In both scenarios, no external memory is accessed and the internal L3 platform cache is configured as shared SRAM to reduce memory access delay and focus on interferences in the interconnect. The activity of the competing cores has been set by the PWM signal in parallel for all cores from 0% to 100% in steps of 10%. The execution time of the main application is measured. Figure 3 shows the results of the scenario without local caches. It can be observed that for the *Read* benchmark thwarting the competing cores by 10% still reduces execution time of the main application by nearly 30%. The decrease stays very intensive until the competing cores reach an activity rate of 60%. Below 60% the execution time of the main application decreases nearly linearly. The TACLe benchmark performs nearly 15% better in case competition is reduced from 100% to 90%. Below this value, the execution time decreases more or less linear until the competition is zero.

We ran the same set of benchmarks with active L1 data and L1 instruction caches for all cores. Here, the overall slowdown is not as dramatical as without caches. Even when running the *Read* benchmark as opponent the main core performs significantly better with a factor of 1.5 on execution time compared to nearly 4.5 as maximum slowdown without caches. This effect is not based on data accesses since the benchmark is constructed to generate the maximum cache miss rate on the data path but the L1 instruction cache is also enabled (disabled in previous scenario) now, which relaxes the pressure on the interconnect and memory significantly. The TACLeBench shows a maximum increase in execution time of only 10%, compared to a factor of 2.15 in the previous scenario. If the performance of the competing TACLeBench cores is reduced, the main core improves nearly linearly while execution time with *Read* opponents is reduced intensively for duty cycles over 80%. Below 80% the performance improvement is also linear.

Our evaluation of the PWM-based thwarting of competing cores show a suitable performance improvement of a memory intensive main application if the reduction is only 10–20%, depending on the use of instruction caches (data caches have no effect on this benchmark) (Fig. 4). In case of an application that is using shared resources to a realistic extend, PWM-based thwarting leads to nearly a linear improvement. The choice of the *Read* and *TACLeBench* shows that the

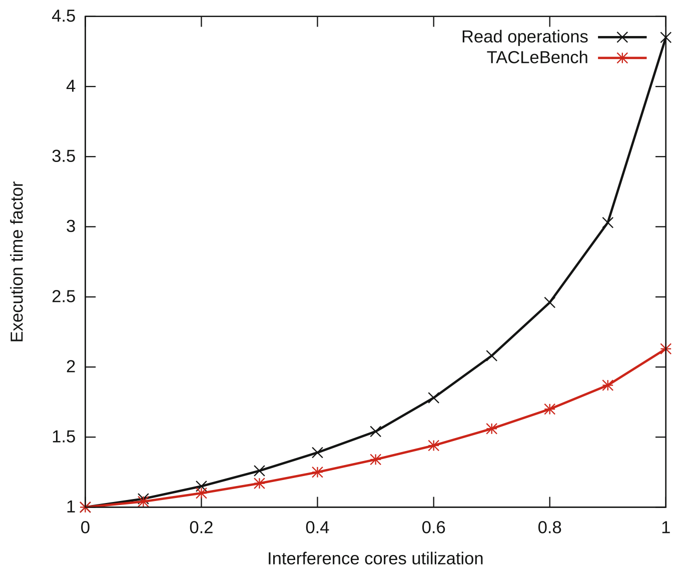

Fig. 3. Execution time of the TACLe benchmark without any local caches and different activity of competing cores

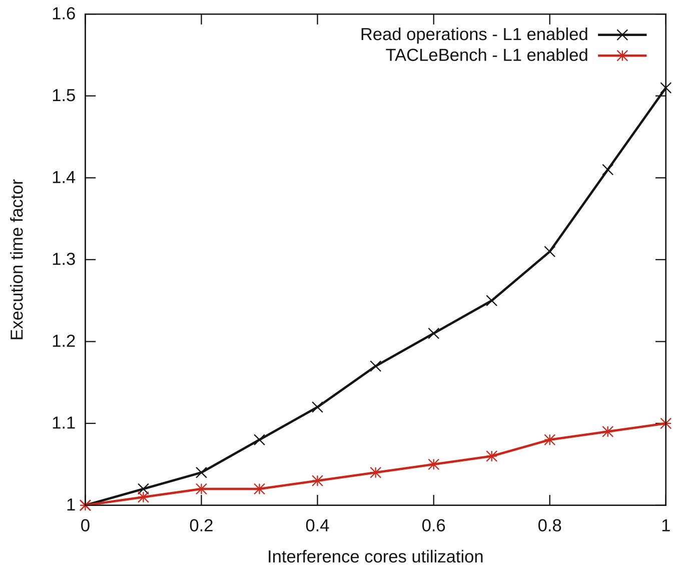

Fig. 4. Execution time of the TACLe benchmark with enabled local instruction and data L1 caches and different activity of competing cores

slowdown with realistic applications running in parallel on other cores are small and may also be within the acceptable delay (e.g. 10%) when the L1 caches are enabled. However, in case one of these applications turns into a *bad guy* (e.g. by a fault) similar to the *Read* benchmark, PWM-based thwarting can protect the correct timing of the main application.

4.2 Closed Loop Controller

We evaluated the closed control loop using TACLeBench as main application and *Read* as *bad guys* running on seven cores in parallel. We set a maximum slowdown of 4% as target performance of the main application compared to stand-alone execution.

Figure 5 shows the performance of the TACLeBench over time (upper part) and the development of the slowdown over time (lower part) without any interference control and with simple threshold-based control. The upper part presents the number of executed instruction per µs. It can be seen that the uncontrolled execution takes about 10% longer for execution at the end. The diagram in the lower part represents the slowdown of the main application as tracked by the *Fingerprinting*. Since tracking of progress is based on discrete steps, the performance reductions are manifested in sharp steps. The following phases of smooth performance increases are caused by relative distribution of a slowdown over a longer time, i.e. a one-time delay at the start of the application of 5% is reduced over the total execution time to a much lower slowdown. The dotted line represents the threshold (4%) i.e. the maximum target slowdown of the main application.

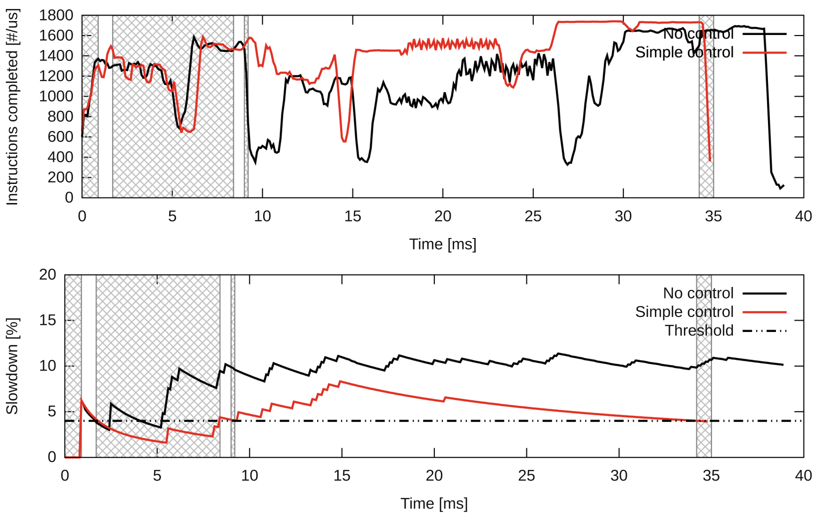

Fig. 5. TACLe performance over time without control and with applied simple threshold controller

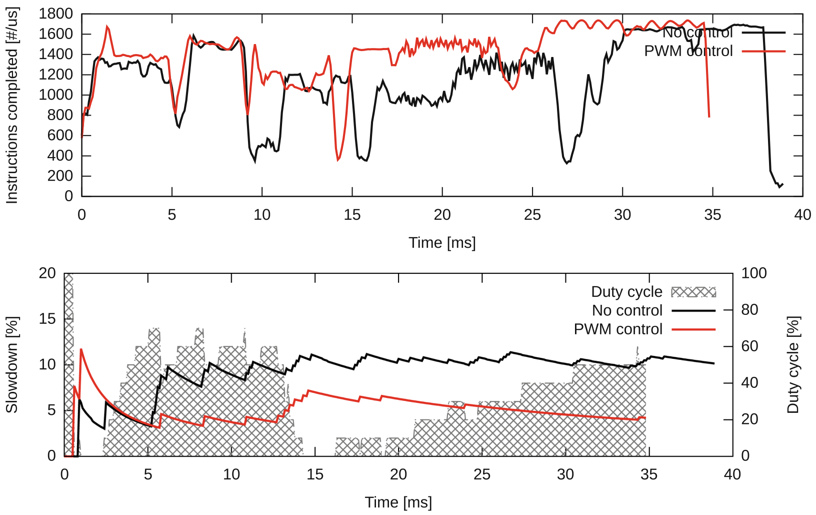

Fig. 6. TACLe performance over time without control and with applied PWM controller

As can be seen in the figure, TACLeBench experienced a slowdown of about 10% over the complete execution time if no control mechanism is applied. With our simple control, the target of 4% maximum slowdown is reached at the end. The grey shaded boxes identify the times when the other seven cores are active. No grey shading means that the other cores are disabled by the control mechanism. At first glance, the competing cores are most of the time disabled meaning that applications running on these cores will not get much execution time. But, note that the competing applications are seven *bad guy* applications flooding the shared resources with maximum traffic. However, even in this simple control case, the other cores each get 23.4% processing time.

In Fig. 6 we show the behaviour of the PWM controller. The duty cycles of the competing cores are set according to the actual slowdown. A slowdown of less than 2% allows full performance for all cores, a slowdown above 7% leads to completely disabled competing cores. Between 7% and 2%, the duty cycles are adjusted in 10% steps from 0% to 100% (one step per half percent of slowdown). The grey shaded areas represent the duty cycles of the PWM core activation signal.

As can be observed that the 4% target slowdown of the main application is also reached at completion. Moreover, the active phases of the competing cores are much longer in time but less intensive. Since we are using a PWM signal, this means that the cores are active for many but smaller periods. The period of a PWM signal is 1 ms (10 times the sampling period of the *Fingerprinting*). With this PWM control, the seven *bad guys* get 34% of the cores' performance while the main application still meets the performance requirements.

5 Conclusion

In this paper a closed control loop for interferences on a multicore processor is presented. It enables a stand-alone WCET estimation of a hard real-time application and execution on a multicore system concurrently to other applications by defining an acceptable slowdown. The presented approach is transparent and none-intrusive to the critical application as it is implemented as an external safety net using the debug/tracing interface for extraction of performance counter values. Furthermore, no additional software functionality on the multicore is required. The fingerprinting approach is used to measure an application's progress. In the closed control loop it is used as sensor element while a simple core on/off switch or, alternatively, a pulse width modulated core thwarting technique represents the actuator.

Our evaluations show that both control techniques perform well and can guarantee the given maximum slowdown factor. We used a *Read* benchmark application flooding the shared resources with maximum traffic and the TACLeBench benchmark suite as competing applications. With the simple control mechanism, a digital enable/disable of competing cores, the competing cores can still get 23.4% performance. When applying a PWM-based controller, the same competing applications will get 34.0% performance.

In the future we will add another performance setting actuator that modifies the frequency of competing cores such that they can stay active all the time but with reduced clock frequency. Moreover, evaluations with other realistic traffic applications and different cache settings can be interesting. Moreover, we plan providing a formal analysis and, hence, a prove of our approach.

References

1. Airbus: future of urban mobility. http://www.airbus.com/newsroom/news/en/2016/12/My-Kind-Of-Flyover.html
2. Certification Authorities Software Team (CAST): position paper CAST-32A: multi-core processors, November 2016
3. Freitag, J., Uhrig, S.: Dynamic interference quantification for multicore processors. In: Proceedings of the 36th IEEE/AIAA Digital Avionics Systems Conference (DASC), pp. 1–6, September 2017
4. Nowotsch, J., Paulitsch, M.: Leveraging multi-core computing architectures in avionics. In: 2012 Ninth European Dependable Computing Conference, pp. 132–143, May 2012
5. Agirre, I., Abella, J., Azkarate-Askasua, M., Cazorla, F.J.: On the tailoring of CAST-32A certification guidance to real COTS multicore architectures. In: Proceedings of 12th IEEE International Symposium on Industrial Embedded Systems (2017)
6. Kritikakou, A., Rochange, C., Faugère, M., Pagetti, C., Roy, M., Girbal, S., Gracia Pérez, D.: Distributed run-time WCET controller for concurrent critical tasks in mixed-critical systems. In: Proceedings of the 22nd International Conference on Real-Time Networks and Systems, RTNS 2014, New York, NY, USA, pp. 139:139–139:148. ACM (2014)

7. Nowotsch, J., Paulitsch, M., Buhler, D., Theiling, H., Wegener, S., Schmidt, M.: Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In: ECRTS, pp. 109–118. IEEE Computer Society (2014)
8. Bak, S., Yao, G., Pellizzoni, R., Caccamo, M.: Memory-aware scheduling of multicore task sets for real-time systems. In: 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 300–309, August 2012
9. Agrawal, A., Fohler, G., Freitag, J., Nowotsch, J., Uhrig, S., Paulitsch, M.: Contention-aware dynamic memory bandwidth isolation with predictability in cots multicores: an avionics case study. In: 29th Euromicro Conference on Real-Time Systems (ECRTS), June 2017 (to appear)
10. Girbal, S., Jean, X., Rhun, J.L., Gracia Pérez, D., Gatti, M.: Deterministic platform software for hard real-time systems using multi-core COTS. In: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), pp. 8D4-1–8D4-15, September 2015
11. Duesterwald, E., Dwarkadas, S.: Characterizing and predicting program behavior and its variability. In: International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 220–231 (2003)
12. Maggio, M., Hoffmann, H., Santambrogio, M.D., Agarwal, A., Leva, A.: Power optimization in embedded systems via feedback control of resource allocation. IEEE Trans. Control Syst. Technol. **21**(1), 239–246 (2013)
13. Sahoo, D.R., Swaminathan, S., Al-Omari, R., Salapaka, M.V., Manimaran, G., Somani, A.K.: Feedback control for real-time scheduling. In: Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301), vol. 2, pp. 1254–1259, May 2002
14. Cucinotta, T., Checconi, F., Abeni, L., Palopoli, L.: Self-tuning schedulers for legacy real-time applications. In: Proceedings of the 5th European Conference on Computer Systems, EuroSys 2010, New York, NY, USA, pp. 55–68. ACM (2010)
15. Fu, Y., Kottenstette, N., Lu, C., Koutsoukos, X.D.: Feedback thermal control of real-time systems on multicore processors. In: Proceedings of the Tenth ACM International Conference on Embedded Software, EMSOFT 2012, New York, NY, USA, pp. 113–122. ACM (2012)
16. NXP Semiconductors: e500mc Core Reference Manual (2013). Rev. 3
17. Falk, H., Altmeyer, S., Hellinckx, P., Lisper, B., Puffitsch, W., Rochange, C., Schoeberl, M., Sørensen, R.B., Wägemann, P., Wegener, S.: TACLeBench: a benchmark collection to support worst-case execution time research. In: Schoeberl, M. (ed.) 16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016). OpenAccess Series in Informatics (OASIcs), Dagstuhl, Germany, vol. 55, pp. 2:1–2:10. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2016)