# Understanding Shared Memory Bank Access Interference in Multi-Core Avionics

**Andreas Löfwenmark and Simin Nadjm-Tehrani**

**Department of Computer and Information Science**
**Linköping University, Sweden**
**{andreas.lofwenmark, simin.nadjm-tehrani}@liu.se**

## Abstract

Deployment of multi-core platforms in safety-critical applications requires reliable estimation of worst-case response time (WCRT) for critical processes. Determination of WCRT needs to accurately estimate and measure the interferences arising from multiple processes and multiple cores. Earlier works have proposed frameworks in which CPU, shared cache, and shared memory (DRAM) interferences can be estimated using some application and platform-dependent parameters. In this work we examine a recent work in which single core equivalent (SCE) worst case execution time is used as a basis for deriving WCRT. We describe the specific requirements in an avionics context including the sharing of memory banks by multiple processes on multiple cores, and adapt the SCE framework to account for them. We present the needed adaptations to a real-time operating system to enforce the requirements, and present a methodology for validating the theoretical WCRT through measurements on the resulting platform. The work reveals that the framework indeed creates a (pessimistic) bound on the WCRT. It also discloses that the maximum interference for memory accesses does not arise when all cores share the same memory bank.

**1998 ACM Subject Classification** D.4.7 Organization and Design – Real-time Systems and Embedded Systems

**Keywords and phrases** multi-core, avionics, shared memory systems, wcet

## 1 Introduction

Future safety-critical avionic systems will use multi-core platforms, partly because of the more complex systems requiring more computational capacity and partly because of decreasing availability of single-core processors; but there are still challenges remaining to demonstrate the predictability needed for certification.

The memory hierarchy, and more specifically, shared caches and dynamic random access memory (DRAM) is one of the major sources of timing variability in a multi-core system [9]. Parallel accesses by cores can lead to interference and either of the resources can become saturated.

Shared caches introduce a number of problems when estimating worst-case execution time (WCET): an intra- or inter-task interference may occur when tasks on the *same* core evict either their own cache lines or another task's cache line respectively. In addition, asynchronous operating system activities can result in cache pollution. Furthermore, inter-core interference is the result of a task evicting a cache line used by a task on *another* core.

The DRAM memory system is composed of a memory controller and memory devices that store the data. The controller is a shared resource in most multi-core systems, which if accessed simultaneously from multiple cores has to somehow arbitrate the accesses and this arbitration can lead to non-determinism in the time domain. DRAM memory devices are organized into ranks containing banks. Banks contain a number of rows and each row has a

number of columns. For each bank there is a row buffer that is used to store the contents of one row in the bank. To read data from memory, the row containing that data must be opened and the contents read to the row buffer and from there the column containing the data can be read. Subsequent requests to the same row can be serviced with low latency, as the row is already open. If a request requires another row to be opened, this will increase the latency as the currently open row must be closed and the data written back to the row before the new row can be opened. This will also affect the worst-case response time (WCRT) if different cores request data from different rows in the same bank.

To mitigate these effects when estimating the WCET, several methods have been proposed [9]. One approach targeting the problems outlined above is the Single Core Equivalence (SCE) framework proposed by Mancuso et al. [12]. This approach combines several of the previously proposed approaches and consists of three parts: Colored Lockdown [11] for managing the shared cache; MemGuard [24] for monitoring and limiting the number of DRAM requests; PALLOC [23] for DRAM bank partitioning. Starting from single-core WCET estimations, they are able to add interference bounds resulting from shared resource usage on a multi-core platform to minimize the effects from other cores.

For some systems it may be possible to locate the data in such a way that each core can access its own private bank(s), but in the general case there will be some sharing of data between applications and these applications may reside on different cores resulting in a use-case where shared banks is a necessity. It may also be the case that we have more cores than banks, which also will result in the necessity of sharing banks. Currently, the number of cores in a multi-core chip is growing faster than the number of banks in the DRAM [8].

In this paper we consider integrating the SCE concepts in an ARINC 653 [1] real-time operating system (RTOS) designed for avionic systems. Specifically, we study the general case of bounding the interference delay when using shared DRAM banks.

The contributions of this paper are:

- We adapt the SCE approach for WCRT estimation in avionics software by integrating assumptions valid for our context, namely cache partitioning and memory bank sharing.
- We adapt a custom RTOS to restrict memory accesses according to earlier works ([5,10, 14,24]).
- We present a methodology for validation of the WCRT estimates using the modified RTOS, COTS multi-core hardware, and repeatable measurements.
- We show that accessing the same bank from all cores does not necessarily represent the worst-case interference delay.

The remainder of this paper is structured as follows. Section 2 contains related work and Section 3 contains relevant background. We describe our SCE adaptation and the validation in Section 4 and Section 5 respectively. We conclude the paper in Section 6.

## 2 Related Work

The early work on utilizing multi-core processors for deterministic systems includes CPU scheduling. Anderson et al. [2] propose a hierarchical scheduling with different levels of execution time estimation requirements for the different criticality levels in RTCA/DO-178 [18]. Mollison et al. [13] and Herman et al. [4] continue building on that framework, turning attention to other shared resources, such as shared last-level cache and the DRAM. Both of these shared resources have to be addressed in order to make the execution times of tasks predictable.

Several methods for handling the shared cache have been proposed. These include page coloring [11,19] and explicit reservation [20]. In our work we will adopt the latter by using dedicated cache partitions for each application.

When cores need to access the main memory due to e.g., a cache miss, the system has to somehow arbitrate among the cores. This creates another bottleneck introducing interference. The interference[1] delay experienced by one core depends on how many memory requests the other cores issue, making it hard to estimate WCRT. One way of handling this is to specify a memory request budget for each core or a group of tasks on a core and then to monitor all memory requests and temporarily disallow access by the core if too many memory requests are performed. This will result in an upper bound of the interference delay. Inam et al. [5] use the concept of multi-resource servers, where they monitor both CPU usage and memory requests for each server. Nowotsch et al. [14] focus on extending existing estimation techniques by introducing an interference-delay analysis and a run-time monitoring mechanism that make it possible to analyze each task in isolation and then add the interference delay to account for the shared resources. A similar approach is presented by Fernandez et al. [3], who introduce resource usage signatures and templates to abstract the contention caused and experienced by tasks on different cores. These signatures and templates are used to determine an execution time bound instead of the actual tasks. Yun et al. [24] propose MemGuard, a memory bandwidth reservation system that provides bandwidth reservation for temporal isolation and a reclamation component. None of these monitoring systems consider the effects of the memory requests of the RTOS(es) running on the cores, this was addressed and shown significant in our earlier work [10]. In this paper we will consider both application and RTOS memory requests when profiling or estimating response times.

The memory has often been regarded as a single resource (black box) and a constant time used for the access times, but the DRAM can be in one of several states affecting the time required to perform the access. The DRAM consists of banks that can be accessed in parallel; Yun et al. [23] use this to reduce the interference delay when accessing the DRAM. They implement a memory allocator (PALLOC) that reserves memory in private banks for each core. This will also eliminate any row collisions, where two cores access different rows in the same bank. Row collisions are more expensive than row hits as the currently open row has to be closed. Wu et al. [21] and Kim et al. [8] both model the memory system more realistically than as a single resource. The memory controller has one request queue for each DRAM bank. Wu et al. only consider private DRAM banks for each core. Kim et al. who also include shared banks relax this limitation. Yun et al. [22] study interference arising in COTS platforms that can generate multiple outstanding memory requests and evaluate their approach on a simulation platform. In our work we use a physical COTS platform.

There are also efforts to develop predictable memory controller hardware [15,16]. This is of course a potential scenario in the future. In this paper we are interested in deploying our system to an available COTS hardware platform in the absence of these options.

In modern memory controllers the memory requests are not always sent to the DRAM in the order they are sent by the core. Instead, they are buffered in request buffers and issued to the DRAM in the order specified by a memory scheduler. The policy often used today, is the First-Ready First-Come First-Served (FR-FCFS) policy. This policy prioritizes requests to already open rows before closed rows in order to minimize row conflicts.

[1] Note that in a single core context the increase in response time due to a shared resource is referred to as blocking, but we use the term interference here to stay consistent with the recent multi-core literature.

# 3 Background

In this section, we review the basic concepts relevant to this work.

## 3.1 DRAM Controller

A DRAM controller sends a number of commands: precharge (PRE) to close an open row; activate (ACT) to open a row; read (RD) and write (WR) to read or write data to the row buffer. The commands take time to finish and the DRAM controller must satisfy timing constraints between the commands. The JEDEC standard [6] specifies a number of requirements for JEDEC-compliant SDRAM devices as shown in Table 1.

**Table 1** DRAM timing parameters

| Parameter | Value | Description |
|---|---|---|
| $BL$ | 8 columns | Burst Length |
| $CL$ | 13 cycles | Column Access Strobe Latency |
| $WL$ | 9 cycles | Write Latency |
| $t_{RCD}$ | 13 cycles | Activate to read/write latency |
| $t_{RRD}$ | 5 cycles | Activate to activate interval |
| $t_{RP}$ | 13 cycles | Precharge to activate interval |
| $t_{FAW}$ | 26 cycles | Window for four activates |
| $t_{WTR}$ | 7 cycles | Write to read interval |
| $t_{WR}$ | 14 cycles | Data to precharge min interval |
| $t_{CK}$ | 1 ns | DRAM clock cycle time |

## 3.2 Inter- and Intra-bank Delay

In the general case tasks share data and need to share banks. To more accurately model the worst-case memory interference delay, we build on the work by Kim et al. [8] that includes the interference delay resulting from shared banks. We briefly describe the request-driven notation used in this work to reuse in later sections.

The interference delay experienced by a core $p$ for a memory request is given by $RD_p = RD_p^{inter} + RD_p^{intra}$, where $RD_p^{intra}$ is the inter-bank interference delay for core $p$ and $RD_p^{inter}$ is the inter-bank interference delay. $RD_p^{inter}$ is the delay due to a memory request generated by a core $p$ is being delayed by requests from other cores due to timing effects of accessing the common command and data bus, it is given by:

$$RD_p^{inter} = \sum_{\substack{\forall q: q \neq p \ \wedge \\ shared(q,p)=\emptyset}} (L_{inter}^{PRE} + L_{inter}^{ACT} + L_{inter}^{RW}) \tag{1}$$

$shared(q,p)$ is the set of DRAM banks shared between core $q$ and core $p$, $L_{inter}^{PRE}$ reflects timings of the address/command bus scheduling. $L_{inter}^{ACT}$ is related to the minimum separation time between two activate commands sent to two different banks, and $L_{inter}^{RW}$ is related to the data bus contention and the bus turn-around delay as a result of the data flow direction change if a read is issued after a write or vice versa. $RD_p^{intra}$ is a result of multiple cores

accessing (different rows in) the *same* bank and is given by:

$$RD_p^{intra} = reorder(p) + \sum_{\substack{\forall q: q \neq p \ \wedge \\ shared(q,p) \neq \emptyset}} (L_{conf} + RD_q^{inter}) \tag{2}$$

where $reorder(p)$ calculates the delay from the reordering based on the number of queued row hits ($N_{reorder}$) that may be scheduled before the request under analysis and $L_{conf}$ is a constant that represents a row-conflict in the same bank, which requires both a PRE and an ACT command to close the current row and open a new row.

We use the request-driven approach presented by Kim et al. [8] since it does not make any assumptions on the memory requests of applications running on other cores. The job-driven approach can in some situations reduce the pessimism of the delays, but it requires us to know the number of interfering memory requests from other cores and goes against the reconfiguration ideas of Integrated Modular Avionics (IMA) [17].

Based on this, Kim et al. extend the classical response time test [7] to include the memory interference delay:

$$R_i^{k+1} = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \cdot C_j + H_i \cdot RD_p + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \cdot H_j \cdot RD_p \tag{3}$$

where $H_i$ denotes the maximum number of memory requests generated by task $i$, $C_i$ denotes the WCET of task $i$ when run in isolation, $R_i$ denotes the response time of task $i$, and $T_i$ denotes the minimum inter-arrival time of task $i$. $R_i^0 = C_i$ and the test terminates when $R_i^{k+1} = R_i^k$.

## 4 SCE Adaptation

In this section we outline the steps needed to implement the SCE concepts on our evaluation platform and RTOS. We assume that the applications tasks are organized in ARINC 653 partitions scheduled in a static cyclic schedule unique for each core.

Instead of implementing the cache-coloring concept, used by Mancuso et al. [12], we utilize the ability to allocate cache ways for exclusive use by a specific core. In our system (described later) the L2 cache is set up to allocate four ways to each of the four cores. This effectively limits the available cache for each core to 512 KiB and will ensure that there is no inter-core cache interference.

The memory request monitoring within the RTOS has been modified to also suspend the partition if the counted number of memory requests exceeds a specified limit during its partition window, in effect regulating the number of memory requests that can be issued from a partition. This is accomplished by using the Performance Monitor Counters (PMCs) to generate an interrupt at overflow. The budget is replenished at the start of each period. The PMC is set up to count both the requests issued by the partition and the requests issued by the RTOS itself. The sum of the memory budgets for all partitions must not exceed the total number of requests possible during a regulation period as this would saturate the DRAM controller and introduce additional delays.

Earlier Linux based bank partitioning assumes that the page size of the memory management unit (MMU) is smaller than the row size of the DRAM (i.e. 4KiB). This makes it possible to always allocate contiguous virtual memory that will map to a set of physical memory pages belonging to the same bank. Our chosen RTOS uses a different approach, where all memory is allocated using variable page sizes during initialization. This will

minimize page misses in the MMU to be handled by the RTOS, but it will also make it very difficult to implement bank partitioning as the MMU page sizes used could possibly span multiple DRAM banks. Therefore, we aim to use the SCE concepts adapted with the shared bank interference delay estimations described in Section 2.

# 5 Validation of the SCE Adaptation

In this section we describe the methodology for validating the adapted SCE model using the implemented SCE mechanisms (as described in Section 4) on the platform, and show the validation results. We use the four avionics related applications from previous work [10]: Nav, Mult, Cubic and Image. Without loss of generality, in this setup all partitions consist of only one process (Nav has two, but we are only interested in the highest priority one), which simplifies the response time calculations. Equation 3 is therefore reduced to $R_i = C_i + H_i \cdot RD_p$.

## 5.1 Methodology

We use the following method for validating the WCRT estimations:

- We estimate the WCET and count the number of memory requests for each partition in isolation.
- The worst-case response time for each partition when executing in parallel is calculated based on the (adapted) SCE formulas.
- We measure the (worst case) response time for each partition and compare with the calculated estimates.
- For critical partitions where calculations indicate a small margin to the relative deadline, we perform additional interference studies with a memory-intensive synthetic application to ensure that maximum memory bank interference is properly accounted for.

Each application is run inside one partition and deployed to different cores and they all execute in 60 Hz. We run the system in an asymmetric multiprocessing (AMP) configuration (i.e. each core has its own instance of the RTOS).

The experiments are performed on an NXP (Freescale) T4240 using only one cluster with four cores sharing the 2048 KiB L2 cache, which is partitioned to allocate four ways for each core resulting in each core having 512 KiB of L2 cache each. Without loss of generality, we have in this work disabled the reordering of requests in the DRAM controller (i.e. $N_{reorder} = 0$).

## 5.2 WCET Estimation

Single-core WCET can be estimated either using static analysis, by measuring the execution time of the application when running on the target hardware or by some hybrid method. In this work we use a measurement approach where we (manually) insert instrumentation points (IPOINTs) that can be used to derive an estimate of the WCET. The four applications are run in isolation on core 0 with the rest of the cores disabled. Table 2 shows the measured WCET and the number of memory requests per period for each application and also for the RTOS.

To ensure that the IPOINTs do not introduce any unintentional probe effects, we measure the execution overhead of an IPOINT and also the number of memory requests with and without IPOINTS in the applications. The maximum execution time of one IPOINT is 23 ns, which gives a maximum overhead of 0.5 percent per partition window. No significant increase of memory requests is observed when using IPOINTs.

**Table 2** Characterization of partitions in isolation

| Partition | Period ($\mu s$) | WCET ($C$) ($\mu s$) | Memory Requests ($H$) (Partition) | (RTOS) |
|---|---|---|---|---|
| Nav | 16667 | 14 | 93 | 54 |
| Mult | 16667 | 16615 | 21740 | 160 |
| Cubic | 16667 | 9345 | 45 | 38 |
| Image | 16667 | 4391 | 560 | 40 |

## 5.3 Response Time Calculations and Measurements

To calculate the response time we need the interference delay, $RD_p$. Using the equations in Section 2 and the DRAM timing parameters in Table 1 we calculate $RD_p^{inter}$ and $RD_p^{intra}$ to get $RD_p$. Figure 1 shows the calculated intra- and inter-bank delays as well as the combined delay ($RD_p$). Intuitively, one would imagine that a higher number of cores gives higher interference. However, as we can see the maximum delay does not occur when all four cores share the same bank. This is a result of intra-bank delay depending on the inter-bank delay for other cores (Equation 2). When all cores access the same bank the inter-bank delay is zero, which will result in the seen delay time drop, given an $L_{conf}$ smaller than the $\sum RD_q^{inter}$ contributing in the case with two cores sharing bank. In the following estimates we use the maximum total interference corresponding to the highest point on the curve (209 ns).

The estimated WCRT listed in Table 3 show that the calculated response time of Nav, Cubic and Image is safely below their relative deadline, but for Mult the estimated WCRT exceeds the relative deadline. Mult performs several orders of magnitude more memory requests compared to Nav, Cubic and Image. For every partition window we use the earlier mentioned IPOINTs to record response times over a 30 second interval (1800 measurements) with partition placement on cores according to column 2. The maximum measured response time is then disclosed in column 4.

This shows that when the partitions execute in parallel no partition misses deadlines though the critical application Mult has a tight margin (see periods in Table 2). We can also see that the WCRT measurements do not differ in any significant way from the WCET measurements in that table. The memory controller can service all the memory requests without being saturated. The memory access patterns of the partitions are such that they do not interfere in many instances. The estimation model assumes that all cores issue memory requests simultaneously. To ensure the measurements are not overly optimistic we perform

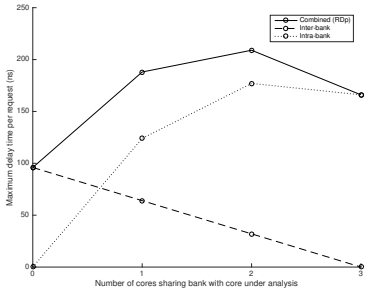


**Figure 1** Theoretical interference delay when using four cores

**Table 3** Maximum response time of partitions

| Partition | Core | Response time ($R$) ($\mu s$) | |
|---|---|---|---|
| | | Estimated | Measured |
| Nav | 0 | 45 | 14 |
| Mult | 1 | 21192 | 16620 |
| Cubic | 2 | 9362 | 9345 |
| Image | 3 | 4516 | 4391 |

additional measurements on Mult, whose estimated WCRT exceeds its relative deadline, in a scenario with maximal memory interference.

## 5.4 Studying Critical Processes Individually

When we run Mult on core 0 in parallel with a memory intensive task deployed on core 1–3 with disabled memory access regulation, Mult misses its deadline. If we turn on the memory regulation, mentioned in Section 4, for the memory intensive tasks on core 1–3 with a suitable budget we notice that Mult does not miss its deadline. This shows that *given* a suitable restriction of memory accesses by partitions running on other cores we are able to run Mult within its time constraints. So, for the critical task, the correct estimation of regulation budget of *other* tasks is essential. To measure Mult's response time in this scenario we disable the overrun detection function and perform repeated experiments where the memory budgets of the memory intensive tasks were reduced until a WCRT value below the relative deadline was found for Mult. The resulting response times (with and without regulation) can be found in Table 4.

**Table 4** Measured maximum response time of Mult with memory intensive tasks in parallel

| Partition | Core | Response time ($R$) ($\mu s$) | |
|---|---|---|---|
| | | No regulation | Regulation |
| Mult | 0 | 17075 | 16654 |

## 5.5 Applying the Method to an Earlier Benchmark

To further assess the validity of the approach we also use the *Latency* and *Bandwidth* benchmarks from Yun et al. [24], adapted to our environment and RTOS, to measure the worst-case memory interference. The benchmarks are modified to enable us to direct the requests from *Bandwidth* to a specified DRAM memory bank. We run *Latency* on core 0 and *Bandwidth* on core 1–3 several times with different number of *Bandwidth* instances targeting the same DRAM memory bank as *Latency*. These measurements compared to the estimations are shown in Figure 2. As we can see, the estimations are a conservative (and possibly somewhat pessimistic) approximation of the measurements.

# 6 Conclusion

In this paper we have presented an adaptation of the SCE framework for an avionics ARINC 653 RTOS targeting the T4240 multi-core SoC from NXP. We have relaxed the constraints of

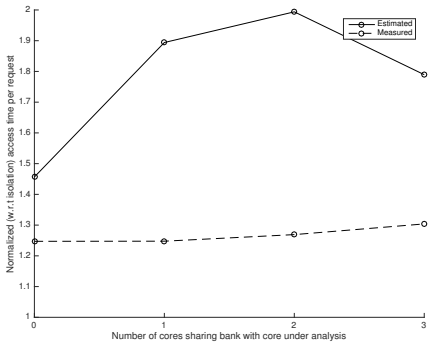


**Figure 2** Comparison of measured and estimated request time

requiring private memory banks for each core and our adaptation provides an analytic upper bound on the interference delay. The implementation on the avionics platform has been used to understand and validate the revised SCE framework using both synthetic and realistic applications for avionics systems. Our work has highlighted an interesting aspect of the calculated response times as a function of the number of cores, namely that the maximum core deployment need not give maximum memory bank interference. It also justifies the use of the SCE framework as an approach to assess schedulability of critical tasks on multi-core platforms. As future work, we will examine the DRAM request reordering we turned off in this work and an improved model of the DRAM controller for more precise estimates.

## Acknowledgement

This work was supported by the Swedish Armed Forces, the Swedish Defence Materiel Administration and the Swedish Governmental Agency for Innovation Systems under grant number NFFP6-2013-01203.

## References

1 Aeronautical Radio Inc (ARINC). ARINC 653: Avionics application software standard interface part 1 - required services, 2010.

2 James H. Anderson, Sanjoy K. Baruah, and Björn B. Brandenburg. Multicore operating-system support for mixed criticality. In *Proc. of Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009.

3 Gabriel Fernandez, Javier Jalle, Jaume Abella, Eduardo Quiñones, Tullio Vardanega, and Francisco J. Cazorla. Resource usage templates and signatures for cots multicore processors. In *Proc. of 52nd Annual Design Automation Conference*, DAC '15, New York, NY, USA, 2015. ACM.

4 J.L. Herman, C.J. Kenna, M.S. Mollison, J.H. Anderson, and D.M. Johnson. RTOS support for multicore mixed-criticality systems. In *Proc. of 18th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2012.

5 R. Inam, N. Mahmud, M. Behnam, T. Nolte, and M. Sjödin. The multi-resource server for predictable execution on multi-core platforms. In *Proc. of 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2014.

6 Joint Electron Device Engineering Council (JEDEC). DDR3 SDRAM Standard, 2012.

7 M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

8 H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar. Bounding memory interference delay in cots-based multi-core systems. In *Proc. of 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2014.

9 A. Löfwenmark and S. Nadjm-Tehrani. Challenges in future avionic systems on multi-core platforms. In *Proc. of 25th IEEE International Symposium on Software Reliability Engineering Workshops*, 2014.

10 A. Löfwenmark and S. Nadjm-Tehrani. Experience report: Memory accesses for avionic applications and operating systems on a multi-core platform. In *Proc. of 26th IEEE International Symposium on Software Reliability Engineering*, 2015.

11 R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-time cache management framework for multi-core architectures. In *Proc. of 19th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2013.

12 R. Mancuso, R. Pellizzoni, M. Caccamo, Lui Sha, and Heechul Yun. WCET(m) estimation in multi-core systems using single core equivalence. In *Proc. of 27th Euromicro Conference on Real-Time Systems*, 2015.

13 M.S. Mollison, J.P. Erickson, J.H. Anderson, S.K. Baruah, and J.A. Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *Proc. of 10th International Conference on Computer and Information Technology*, 2010.

14 J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *Proc. of 26th Euromicro Conference on Real-Time Systems*, 2014.

15 M. Paolieri, E. Quiñones, F.J. Cazorla, and M. Valero. An analyzable memory controller for hard real-time cmps. *IEEE Embedded Systems Letters*, 1(4):86–90, 2009.

16 Jan Reineke, Isaac Liu, Hiren D. Patel, Sungjun Kim, and Edward A. Lee. Pret dram controller: Bank privatization for predictability and temporal isolation. In *Proc. of the 7th International Conference on Hardware/Software Codesign and System Synthesis*, 2011.

17 RTCA, Inc. RTCA/DO-297, integrated modular avionics (IMA) development, guidance and certification considerations, 2005.

18 RTCA, Inc. RTCA/DO-178C, software considerations in airborne systems and equipment certification, 2012.

19 B.C. Ward, J.L. Herman, C.J. Kenna, and J.H. Anderson. Making shared caches more predictable on multicore platforms. In *Proc. of 25th Euromicro Conference on Real-Time Systems*, 2013.

20 Jack Whitham, Neil C. Audsley, and Robert I. Davis. Explicit reservation of cache memory in a predictable, preemptive multitasking real-time system. *ACM Trans. Embed. Comput. Syst.*, 13(4s), April 2014.

21 Zheng Pei Wu, Y. Krish, and R. Pellizzoni. Worst case analysis of dram latency in multi-requestor systems. In *Proc. of 34th Real-Time Systems Symposium*, 2013.

22 H. Yun, R. Pellizzon, and P. K. Valsan. Parallelism-aware memory interference delay analysis for cots multicore systems. In *Proc. of 27th Euromicro Conference on Real-Time Systems*, 2015.

23 Heechul Yun, R. Mancuso, Zheng-Pei Wu, and R. Pellizzoni. Palloc: Dram bank-aware memory allocator for performance isolation on multicore platforms. In *Proc. of 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2014.

24 Heechul Yun, Gang Yao, R. Pellizzoni, M. Caccamo, and Lui Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *Proc. of 19th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2013.