Experience Report: Memory Accesses for Avionic Applications and Operating Systems on a Multi-Core Platform

Andreas Löfwenmark
Avionics Equipment
Saab Aeronautics
Linköping, Sweden
andreas.lofwenmark@saabgroup.com

Simin Nadjm-Tehrani
Department of Computer and Information Science
Linköping University
Linköping, Sweden
simin.nadjm-tehrani@liu.se

Abstract*—The deployment of multi-core platforms in safety-critical avionic applications is hampered by the lack of means to ensure predictability when processes running on different cores can create interference effects, affecting worst-case execution time, due to shared memory accesses. One way to restrict these interferences is to allocate a budget for different processes prior to run-time and to monitor the adherence to this budget during run-time. While earlier works in adopting this approach seem promising, they focus on application level (user mode) accesses to shared memory and not the operating system accesses. In this paper we construct experiments for studying a multi-core platform running an ARINC 653 compliant operating system, and measure the impact of both application processes and operating system (supervisor mode) activities. In particular, as opposed to earlier works that considered networking applications, we select four avionic processes that exhibit different memory access patterns, namely, a navigation process, a matrix multiplication process, a math library process and an image processing one. The benchmarking on a set of avionic-relevant application processes shows that (a) the potential interference by the operating system cannot be neglected when allocating budgets that are to be monitored at run-time, and (b) the bounds for the allowed number of memory accesses should not always be based on the maximum measured count during profiling, which would lead to overly pessimistic budgets.

***Index Terms*—real-time; predictability; multi-core; temporal partitioning; memory management; avionic systems; ARINC 653**

I. Introduction

Multi-core platforms are prevalent in many different application areas, but have not yet found the inroad to the safety-critical segment, due to lack of understanding on how to ascertain software predictability. Although the use of concurrent processes and shared resources is well-studied in real-time systems, the estimation of worst-case execution time (WCET) and maximum response time for each process is based on well-defined regimes allowing mutually exclusive access to each shared resource. With applications running on multi-core platforms this option no longer exists unless all caches are partitioned and the timing effects of accesses to the shared memory are predictably estimated. One of the recent approaches for scheduling critical applications on multi-core platforms is based on the concept of multi-resource servers (MRS) [1]. In this approach all memory accesses of all servers are estimated and the schedule allows a budget for memory accesses that is monitored at run-time. In this paper, we study the extended application of this technique to safety-critical systems, and in particular, whether a bound on memory accesses from servers running on multiple cores suffices for assuring predictability of the timing.

Traditionally, avionics have used functions implemented and packaged as self-contained units (in a federated system). Integrated Modular Avionics (IMA) [2] uses a high-integrity, partitioned environment, that hosts multiple functions of different criticalities on a shared computing platform. The partitioning concept ensures a fault containment level equivalent to its functionally equivalent federated system, which implies partitioning in both space and time [3], [4]. Space partitioning is not a new problem as this is also a concern in other types of systems, but with the introduction of multi-core the time partitioning property needs to be addressed as the avionics functions can now concurrently share resources on the same computing platform.

The main approaches for solving the problems with shared resources, as summarized in our previous work [5], are: *resource management* e.g. dividing the cache among tasks to minimize the interference; *resource arbitration* where access to the resource is scheduled and the task gets private access for a period of time, which can be seen as a special case of resource management using special hardware; and *resource monitoring* where the resource usage is monitored and access can be restricted at run-time if a provided limit is exceeded.

In this paper we look at resource monitoring systems and shared memory in particular. In addition to the memory accesses performed by the applications running on the different cores, we also consider the memory accesses performed by the real-time operating system (RTOS). Most previous works have used a standard desktop operating system (e.g.,

Linux) when performing experiments. Linux is recognized as not being suitable for safety-critical avionics systems, and therefore all effects of the OS are ignored. However, this neglects the memory accesses performed by the RTOS on each core. For instance, system calls, memory management for space partitioning and task switching could all introduce an overhead in the number of memory accesses performed from a particular core. These extra memory accesses could interfere with memory accesses performed from other cores, and thus affect the WCET of those applications.

We evaluate our hypothesis on the T4240, a state-of-the-art multi-core System on Chip (SoC) from Freescale, using avionic representative applications and an ARINC 653 [3] compliant RTOS that we extend for monitoring the memory accesses of all applications, but also the accesses by the RTOS itself.

The contributions of this paper are:

- We perform experiments on memory access monitoring on a multi-core platform relevant for avionic systems. Specifically, we run four different avionic applications with different memory access patterns, to illustrate the diversity of memory access patterns.
- We show the magnitude of the RTOS memory accesses and that they should be considered when analyzing inter-core interference.
- We show that simply using the maximum measured number of memory accesses as the budget is not always the best option, since it could lead to an overly pessimistic budget in the same way the theoretical analysis would.

While the measurements in this paper are based on a given platform and a given set of applications, thereby not of general value, they clearly indicate the need for an additional term in the previously proposed general models for budget-based multi-core resource allocation. This term would have to bound the operating system induced memory accesses so far missing from the models.

The remainder of this paper is structured as follows. Related research and background are discussed in Sections II and III. Section IV introduces our methodology and in Section V we present our experimental results. We discuss some aspects of the results in Section VI and in Section VII we conclude the paper.

II. Related Work

Although a lot of research has been performed on multi-processor systems (e.g., [6]–[8]), focus of the great majority of these works is throughput and performance, not worst-case guarantees and predictability, thereby they are not further discussed here.

How to efficiently utilize all cores in a safety-critical multi-core system is a research topic receiving a substantial amount of interest. Pellizzoni et al. [9] attempt to overcome the problems with non-predictable COTS components by introducing a system execution model, PRedictable Execution Model (PREM), which introduces two new hardware devices, *real-time bridge* and *peripheral scheduler*, to make the scheduling of I/O peripherals more deterministic. They also divide tasks into intervals (compatible and predictable), similar to Schranzhofer [10], where the predictable interval is further divided into a memory phase and an execution phase. During the memory phase shared memory accesses are performed to fill the cache with the data needed for the execution phase, which is performed without memory accesses and cache misses. The I/O peripherals can access main memory during an execution phase without contention.

For hardware platforms allowing accesses to shared resource to be scheduled (e.g., using time division multiple access (TDMA)), the accesses can be arbitrated and the interference can be eliminated. Such a platform was used by Schranzhofer et al. [10] to analyze the worst-case response time (WCRT). Three different access models are examined: dedicated, general and hybrid access. An analytical worst-case analysis framework considering blocking/no-buffered accesses to a shared resource was proposed and evaluated for all three models. They showed that separating computations and accesses to the shared resource is important for accurately computing the WCRT. Whitham et al. [11] describe a hardware mechanism for explicit reservation of cache memory to reduce the cache-related preemption delay observed when tasks share a cache.

Resource arbitration using TDMA may lead to under-utilization and may not be the best choice when both predictability and performance are important [12], it also requires special hardware support. Giannopoulou et al. [13] use the dedicated access model from [10] and suggest a scheduling policy for a mixed-criticality multi-core system with resource sharing without the need for special hardware support. The policy prevents timing interference among the tasks of different criticality levels by allowing only a statically known set of tasks with the same criticality level to execute on the cores at any time. To achieve more efficient resource utilization, static and dynamic barriers are used for synchronization on the global level, and on the core level a flexible time-triggered scheduling strategy is used. This enforces timing isolation between criticality levels and enables composable and incremental certifiability. The cost is run-time overhead for the clock and barrier synchronization between the cores. An approach similar to Giannopoulou's was proposed by Mollison et al. [14] who presented a two-level hierarchical scheduling framework for mixed-criticality tasks. The top-level schedules container tasks, which in turn contain the "normal" tasks. Each container consists of tasks of the same criticality level and uses a given scheduling strategy. Biondi et al. [15] present a framework for component-based design in multi-processor real-time systems under partitioned scheduling. They map components to virtual processors implemented through reser-

vation servers. During integration, the virtual processors are assigned to the physical processors. The framework enables resource sharing among independently developed real-time applications. Since the resource access protocol makes use of FIFO non-preemptive spinlocks to ensure mutual exclusion among the processors, the approach is not really viable for a system where the whole memory is the shared resource.

An approach to avoid the shared resource interference while still using multiple cores was described by Fuchsen [16]. An ARINC 653 compliant partitioning model is used where a safety-critical partition would run on its own core and during its partition window no other partitions are allowed to run on any of the other cores. When no critical application is running, several partitions may use the cores or all cores can be assigned to one partition. This eliminates shared resource interference on the critical applications, but it is also quite expensive since many cores may be unused in several time windows.

Monitoring the shared resource usage is yet another approach, which can be done in a way similar to the CPU-monitoring concept of ARINC 653 where each partition is limited in the amount of allocated CPU-time. By using the performance monitor counters found in most CPUs, it is possible to track for example the consumed memory bandwidth and enforce limitations on the number of memory accesses to prevent contention. Yun et al. [17] use performance counters to get information on memory accesses to separate real-time and non-real-time tasks. They concentrate the critical tasks to one core (the critical core) and the other cores (interfering cores) contain the non-critical tasks. Run-time monitoring is used to throttle the memory accesses by the interfering cores if more memory requests than allocated are performed. The focus of the work is to schedule the critical tasks while the impact on the non-critical tasks is minimized. This approach may result in under-utilization if the critical tasks do not execute often.

In another work by Yun et al. [18] the concept of a memory-performance critical section is introduced, where a task uses an API to indicate it requires a high memory bandwidth and the OS regulates the other cores. This is designed to protect soft real-time applications and can result in heavy penalties on co-running applications. A more general approach is proposed by Inam et al. [1], where a multi-resource server (MRS) is presented. The server maintains two different budgets; one for the CPU usage and one for the number of memory requests, but the memory throttling is proposed per server instead of per core, and several servers can execute on a core. Nowotsch et al. [19] target an integrated approach of worst-case timing analysis for multi-core and a run-time monitoring mechanism. In addition to existing single-core WCET estimation techniques, the authors analyze the maximum number of shared resource accesses per shared resource. Based on these estimates they introduce a new phase, the interference-delay analysis, to account for the additional interference caused by shared resources. This way they can separate the analysis of tasks scheduled to execute in parallel and analyze each task in isolation, followed by the interference-delay analysis to estimate the multi-core WCET. However, compared to our work, none of the works based on monitoring have taken the memory accesses by the RTOS into account, which is the focus of this paper.

III. Background

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

A. ARINC 653

ARINC 653 [3] is a software specification in avionics systems. Central to ARINC 653 is the concept of partitioning, whereby the applications are partitioned with respect to space (memory partitioning) and time (temporal partitioning). A partition is a program unit of the application to satisfy these partitioning constraints. An ARINC 653 compliant RTOS supports robust partitioning, which allows partitions with different criticality levels to execute on the same computing platform, without affecting one another spatially or temporally, in the context of an IMA architecture. ARINC 653 defines an Application Programming Interface (API) called APlication/EXecutive (APEX), allowing applications to move between ARINC 653 compliant RTOSes without rewriting the whole application. The APEX API contains services to manage partitions and processes, to handle inter- and intra-partition communication, and also services for error handling.

Partitions are scheduled in a static cyclic schedule, which prescribes when and for how long a partition can execute. The major frame time determines the length of the repeating schedule, and each major frame is divided into partition windows with a specified start time within the major frame and a duration. A partition can be scheduled in arbitrarily many partition windows within a major frame.

A partition consists of one or more processes that combine to provide the functions associated with that partition. Processes within a partition share the same memory address space, and are scheduled according to a priority-based preemptive scheduling scheme. The naming convention using the terms partition and process is somewhat confusing as for general purpose operating systems the terms process and thread are often used, where the processes are separated in memory and the threads share the memory address space. In this paper we use the nomenclature of ARINC 653.

All resources needed by a partition are statically allocated in a configuration file. This includes for instance the number of processes, the memory requirements and inter-partition communication channels. During partition initialization all resources used by the partition are created. When the normal mode execution starts it is no longer possible to create additional processes.

The current revision of ARINC 653 is for single-core processors, but work is currently ongoing by the Airlines Electronic Engineering Committee (AEEC) APEX subcommittee[1] to adapt the standard for use on multi-core processor architectures.

B. Hardware Architecture

The T4240[2] belongs to the state-of-the-art Freescale QorIQ T series. The T4240 consists of 12 Power Architecture®e6500 cores. Each core supports two hardware threads, which software views as two virtual CPUs, and thus rendering a total of 24 (virtual) cores. The e6500 cores are clustered in banks of four cores sharing 2 MB level 2 (L2) cache, but with private 32 KB L1 data and instruction caches.

The e6500 core provides three privilege levels to provide different levels of protection for software to operate under. This allows for building systems that provide partitioning and virtualization. Hypervisor mode is the highest privilege level; it allows access to all instructions and resources, the supervisor mode is the next level; it has access to several privileged instructions, and user mode is the unprivileged and lowest level; this is where most normal applications run.

To implement the space and time partitioning of ARINC 653, the RTOS can execute in hypervisor or supervisor mode and the applications execute in user mode. In the rest of the paper we will use the term supervisor mode for the mode in which the RTOS is executing, regardless of whether it executes in supervisor or hypervisor mode.

IV. Methodology

This paper aims to evaluate the hypothesis that RTOS memory accesses on one core could interfere with applications executing on other cores when memory budgets have been determined only using knowledge about the behavior of the applications.

We use four applications, described in Section IV-D, with different memory access characteristics to evaluate the relation between the number of memory accesses (i.e. L2 cache misses) performed by the applications and the number of memory accesses performed by the RTOS. In addition, we measure the number of memory accesses performed by the RTOS during the allocated partition windows.

The applications are run for approximately 30 minutes, the length of a typical mission, after which the applications output the collected data.

[1] http://www.aviation-ia.com/aeec/projects/apex/

[2] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=T4240

A. Hardware Platform

All experiments are performed on a Freescale T4240QDS[3] development board using one cluster of four physical cores. Only one of the two hardware threads is used in each physical core, giving a total of four virtual cores in the cluster sharing one L2 cache. The L2 cache is however disabled since we are only interested in cache misses, and the L1 cache is invalidated before each partition window so the same preconditions apply to all partition windows.

The T4240QDS system is connected to a probe for loading software into the memory, running the experiments and debugging. All data is collected over a RS232 serial link.

B. Software Platform

We use a proprietary ARINC 653 compliant research RTOS, extended by us as described in Section IV-C. The system is run in an asymmetric multiprocessing (AMP) configuration, meaning separate RTOS instances on each core.

Each application is implemented in a partition separate from the other applications, and each partition is scheduled to execute on different cores. Thus, we have one partition running on each of the four cores.

A current single-core partition schedule would normally consist of more than one partition and partition window, typically 6-10 partitions, in a major frame. In our case, we would schedule all four partitions within a major frame. In this paper, the partition schedule for each core consists of one single partition window, since we only have one partition on each core. The partitions will execute in 60 Hz, which is a commonly used frequency in avionics. The partition window duration, and also the major frame time, will thus be 16.67 ms. Even though there is only one partition window in each schedule, the partition scheduling is still activated after the partition window duration when starting a new major frame.

C. RTOS Extensions

The RTOS is a proprietary software, originally designed for use on an e500 PowerPC core, and has been ported by us to the e6500 PowerPC core used in the T4240 SoC.

The monitoring is implemented using the Performance Monitor Counters (PMCs) that exist in the e6500 core. The cores provide the ability to count L2 instruction misses per core and L2 data misses per core. Six performance counters are set up, two for counting data and instruction misses in supervisor mode, two for counting data and instruction misses in user mode, and the last two for counting data and instruction misses in supervisor mode only when the Performance Monitor Mark (PMM) flag is set in the Machine Status Register. The RTOS is

[3] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=T4240QDS

also extended to set the PMM flag under the partition switch interval while scheduling. By setting the PMM flag during partition scheduling and setting the last two counters to only count when PMM is set, we can separate the memory accesses performed in supervisor mode during a partition window and the memory accesses performed during partition scheduling, which are also performed in supervisor mode.

On each core, the PMCs will count each instruction and data miss to the L2-cache in each partition window. During partition scheduling the RTOS will store the values to a buffer and the counters are set to zero. The buffers are available to the partitions in user mode.

D. Applications

The applications have been selected as they exhibit characteristics representative for avionic applications. From Figure 1 we can also see that the applications exhibit quite different memory access characteristics. The applications are contained within one ARINC 653 partition each.

- Nav This partition consists of two periodic processes. One, running in 60 Hz, implements a navigation algorithm for an unmanned aerial vehicle (UAV). The other process, running in 1 Hz, simulates a Global Positioning System (GPS) device (by keeping a static array of coordinates). Every second a new GPS coordinate is made available from the GPS process to the navigation process, which in turn processes the position and takes the appropriate action in navigating to the destination.
  The GPS path has 76 coordinates, going from point A to point B and back again. There are four waypoints along the path, two going from A to B and two going from B to A. It can be seen as a surveillance mission for a UAV, going back and forth between the two end points.
- Mult This partition consists of one periodic process. It performs a multiplication of two 1000x1000 matrices. One complete matrix multiplication is performed each period.
- Cubic This partition consist of one periodic process. It is based on the basic math test from the Automotive and Industrial Control category of MiBench [20]. It performs simple mathematical calculations such as cubic function solving, integer square root and angle conversions from degrees to radians, which are all calculations needed for calculating for instance airspeed or other avionics related values. The input data for these calculations is a fixed set of constants. It takes several partition windows to complete the entire set of calculations.
- Image This partition consists of one periodic process. It is based on the Susan test from the Automotive and Industrial Control category of MiBench [20]. It is an image recognition package including edge and corner detection, smoothing and adjustments for threshold, brightness, and spatial control. The input data is a black and white image of a rectangle.
  For each period the process alternates between the available image recognition operations, and it takes several periods to complete all of them.

V. Measurement Results

In this section we evaluate the memory accesses resulting from the described applications and also from the RTOS.

Figure 1 shows that the average number of memory accesses from the four selected applications each running in its own partition differ by orders of magnitude. Also the standard deviation, shown as error bars, varies greatly between the partitions. Mult, for instance, has a very low standard deviation, meaning it is deterministic in its memory accesses from one partition window to another. Cubic on the other hand has a more diverse memory access pattern between partition windows. This indicates that the impact of the RTOS memory accesses will depend on the memory access characteristics of the partitions.

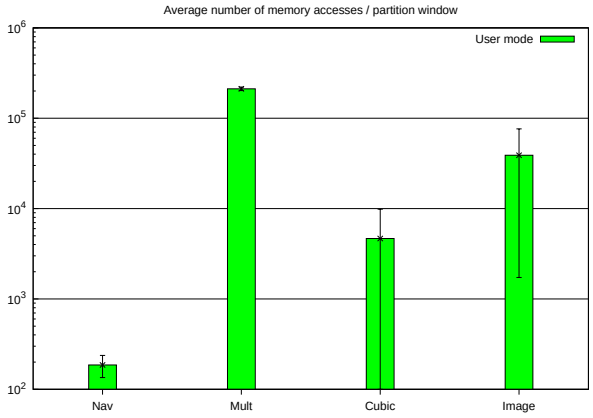

Fig. 1. Memory access characteristics for included applications

Figure 2 and Figure 3 show the memory accesses per partition window during the experiment for Nav and Cubic respectively. These two are illustrated here as they represent two distinct characteristics. For Nav, the supervisor mode accesses are of the same magnitude as the user mode accesses, while for Cubic the user mode accesses are orders of magnitude larger. The underlying reason for the large standard deviation for Cubic in Figure 1 shows up in Figure 3 through the order of magnitude difference in the number of accesses between partition windows. The compact band of access numbers in the range just below 10^4 accesses represents a kind of dominant behavior in some partition windows, and the thinner bands just below 10^5 accesses represent another type of behavior exhibited in fewer number of partition windows.

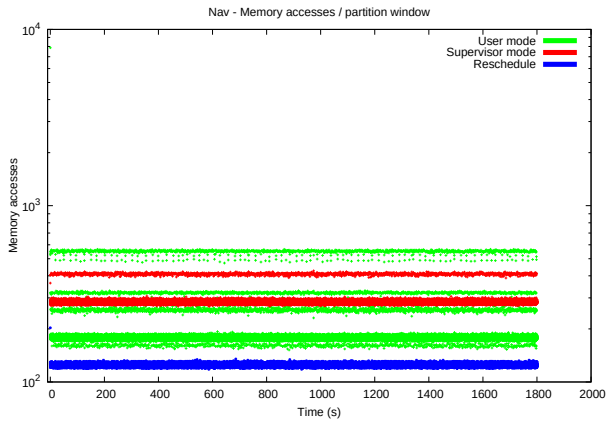

Fig. 2. Nav

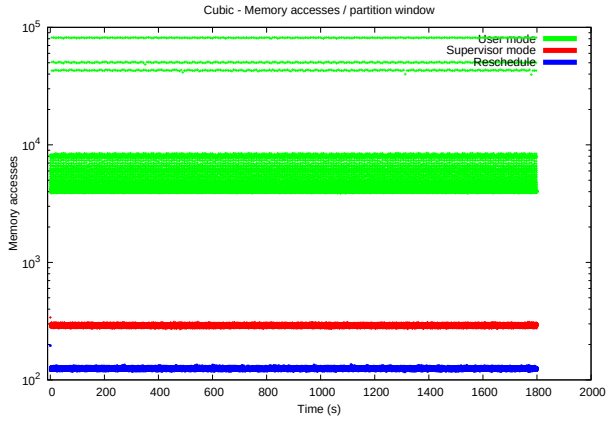

Fig. 3. Cubic

Mult and Image show an access pattern similar to Cubic, but the user mode memory accesses are magnitudes larger than the supervisor mode accesses, and are not shown here.

Both supervisor mode and partition scheduling (Reschedule in the figures) accesses are fairly similar for all partitions, as shown in Figure 4. This figure displays the total number of memory accesses monitored in all partition windows during the experiment period. The magnitude of the supervisor mode memory accesses depends on the behavior of the application running in that partition (e.g., the usage of system calls and is thereby different for different applications). The partition scheduling accesses, on the other hand, are independent of application behavior, and these memory accesses are a constant overhead regardless of whether the partitions on the core execute or not. However, the fact that stable numbers are exhibited displays a degree of determinism in this function of the RTOS.

Partition	Supervisor Mode	User Mode	Reschedule
Nav	48.3%	30.92%	20.78%
Mult	0.07%	99.87%	0.06%
Cubic	5.60%	91.96%	2.44%
Image	0.77%	98.60%	0.32%

TABLE I
DISTRIBUTION OF TOTAL NUMBER OF MEMORY ACCESSES

For Nav, the supervisor mode actually contributes more memory accesses than the application itself. There are more than twice as many memory accesses compared to only using application behavior. This clearly illustrates that had a budget been set on the memory accesses solely dependent on the application behavior, the budget would have been far to tight, and the application operation disrupted at run-time far too often.

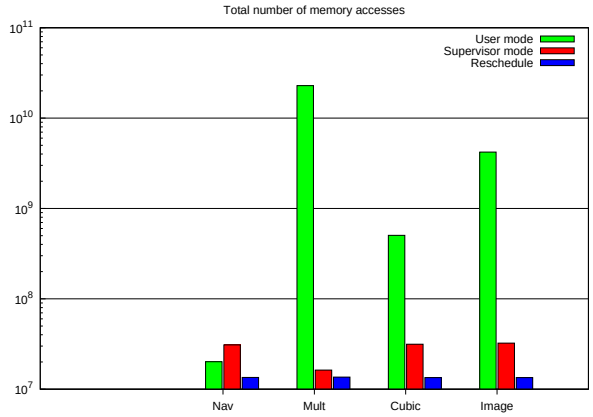

Fig. 4. Total number of memory accesses

Table I summarizes the observations above. For the memory intensive Mult and Image, the supervisor mode accesses account for less than 1 percent. For Nav, on the other hand, being more computationally intensive and consisting of two processes (resulting in more supervisor mode accesses due to process scheduling overhead), the supervisor mode accesses account for almost 50 percent of the total number of memory accesses. Note that even though the percentage is low for three out of the four applications, the extra memory accesses in supervisor and reschedule modes could still have an actual impact on the worst-case response time of the applications on other cores. Looking at individual partition windows, Table II shows that the memory accesses from the RTOS can be quite significant. Even for Image where the supervisor mode accesses account for less than 1 percent of the total number of accesses, we can see that for an individual partition window supervisor mode accesses can account for 66 percent.

Using the box plot in Figure 5 to visualize the diversity of the number of memory accesses over different partition windows,

Partition	Supervisor mode accesses	
	Min	Max
Nav	3.2%	72.0%
Mult	0.1%	7.2%
Cubic	0.3%	8.3%
Image	0.3%	66.0%

TABLE II
PROPORTION OF SUPERVISOR MODE ACCESSES / PARTITION WINDOW

we see the difference in the nature of the four applications in another light.

A relatively narrow box in this context, as illustrated by Nav and Mult access patterns, demonstrates a more deterministic behavior. Here Mult is the most deterministic with fewer outliers. Nav is typically deterministic, but there are partition windows in which outliers exhibit an order of magnitude higher number of accesses, thus clearly questionable if the narrow band would be a better estimate for a budget, or the pessimistic count based on outliers. A further accentuated version of this behavior is shown by Cubic, where a larger number of outliers are shown. This is an example of a process where using the maximum number of accesses in some partition windows, if used for setting the budget, would be too pessimistic. However, barring those outliers, the process shows a relatively deterministic memory access pattern (a narrow band). Finally, image processing is depicted in a wide box with an average that is far less than the other potential number of accesses exhibited. This would be an example of a process for which making an educated guess to a relevant budget for memory accesses would be quite difficult. Reconsidering the structure of the code and potentially dividing it into more deterministic subfunctions would be a recommended approach.

VI. DISCUSSION

If we look closely at Figure 2, we can see that, for Nav, there is one sample in the very beginning (just below 10^4) where the user mode accesses are a magnitude larger than anywhere else. This has to do with the initialization of the partition before entering normal mode. This can also be seen from the outliers in Figure 5, where we can see a similar behavior for Cubic (although we cannot see if the outliers belong to the initialization phase). If we were to use this maximum for doing inter-core interference analysis, the result would be very pessimistic. The outliers should be further analyzed for relevance, and if only occurring during initialization or fatal error handling the outliers can safely be ignored.

The large differences in number of accesses between partition windows exhibited by Cubic and Image may result from the fact that the calculations take several partition windows to complete and the memory access balancing could perhaps be improved to yield a more narrow band of memory accesses. Without paying attention to memory access characteristics during application design, it may be difficult to avoid overly pessimistic budgets and this can affect the possibility to efficiently integrate legacy applications on a multi-core platform.

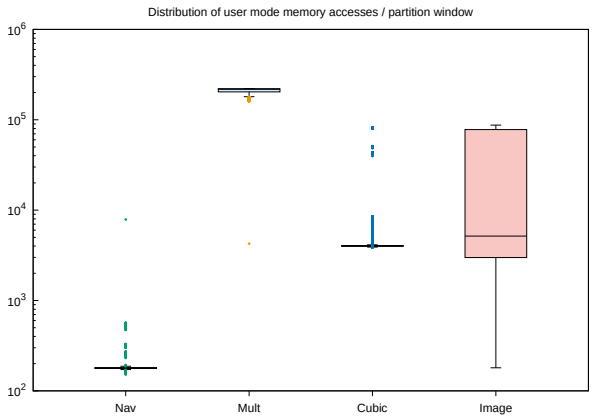

Fig. 5. Distribution of user mode memory accesses / partition window (outliers have been grouped together into one if there are many with the same value)

We have a simplified system for our experiments and because of this there are additional sources of supervisor mode memory accesses that have not been accounted for. The fact that there is only one partition on each core probably results in less space partitioning overhead as the memory management unit of the core is less utilized and this will lead to fewer misses. As stated previously, the partition scheduling of an ARINC 653 system will incur a number of memory accesses due to the static cyclic schedule. Since partition scheduling occurs after each partition window, the overhead will increase as new partition windows are added to the schedule. In our case with only one partition there is also only one partition window, which will result in less partition-scheduling overhead than would occur in a real avionic system. None of these simplifications make the implications less valid. On the contrary, there would be additional memory accesses to support our hypothesis.

The results are also applicable to a symmetric multiprocessing (SMP) ARINC 653 system, but the partition scheduling would not cause interference in the same way as in AMP systems. The partition scheduling would always occur at the same time on all cores and the interference would be internally in the RTOS, but the delay until the next partition starts executing would probably be longer than if running on a single-core system.

VII. CONCLUSION

In this paper, we investigate the problem of RTOS induced memory accesses interfering with the execution of other cores in a multi-core AMP system, and present our experimental results. We use four applications and an ARINC 653 RTOS, all representative for avionic systems.

The results show that the impact of RTOS memory accesses are dependent on application behavior, ranging from quite significant for the computationally intensive Nav where the RTOS contributed more memory accesses than the application itself (48 percent vs. 30 percent of the total number of memory accesses), to marginal for the very memory intensive Mult where the RTOS contributed less than 0.1 percent. Even though the percentage is low for three out of the four applications, there may be a substantial amount of memory accesses from the RTOS adding extra interference on other cores and impacting the worst-case response time of the applications. The partition scheduling in an ARINC 653 RTOS also introduces memory accesses not accounted for due to the static cyclic nature of the schedule.

Our measurements on the four diverging types of applications reveal some challenges related to determining memory access budgets.

- Nav poses no challenges if the initialization outlier can be ignored.
- Mult is deterministic in its memory access behavior and poses no challenges.
- Cubic has too many outliers that nevertheless represent a smaller fraction of number of accesses, resulting in large pessimism.
- Image is an example where it is difficult to avoid pessimism.

While this is a limited study based on only one RTOS, our approach and methodology is equally applicable to other RTOSes. Our results clearly show that RTOS memory accesses need to be incorporated into the previously proposed worst-case timing analyses to correctly perform the inter-core interference analysis.

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] R. Inam, N. Mahmud, M. Behnam, T. Nolte, and M. Sjödin, "The multi-resource server for predictable execution on multi-core platforms," in *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2014.

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

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

[4] RTCA, Inc, "RTCA/DO-178C, software considerations in airborne systems and equipment certification," 2012.

[5] A. Löfwenmark and S. Nadjm-Tehrani, "Challenges in future avionic systems on multi-core platforms," in *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, Nov 2014.

[6] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration," *Computers Digital Techniques, IET*, vol. 5, no. 2, March 2011.

[7] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013.

[8] U. M. Mirza, F. Gruian, and K. Kuchcinski, "Mapping streaming applications on multiprocessors with time-division-multiplexed network-on-chip," *Computers & Electrical Engineering*, vol. 40, no. 8, 2014.

[9] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for cots-based embedded systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, April 2011.

[10] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing analysis for TDMA arbitration in resource sharing systems," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2010.

[11] J. Whitham, N. C. Audsley, and R. I. Davis, "Explicit reservation of cache memory in a predictable, preemptive multitasking real-time system," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4s, Apr. 2014.

[12] T. Kelter, T. Harde, P. Marwedel, and H. Falk, "Evaluation of resource arbitration methods for multi-core real-time systems," in *13th International Workshop on Worst-Case Execution Time Analysis*, 2013.

[13] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Scheduling of mixed-criticality applications on resource-sharing multicore systems," in *Proceedings of the International Conference on Embedded Software (EMSOFT)*, Sept 2013.

[14] M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *Proceedings of the 10th International Conference on Computer and Information Technology (CIT)*, June 2010.

[15] A. Biondi, G. Buttazzo, and M. Bertogna, "Supporting component-based development in partitioned multiprocessor real-time systems," in *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, July 2015.

[16] R. Fuchsen, "How to address certification for multi-core based IMA platforms: Current status and potential solutions," in *Proceedings of the 29th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct 2010.

[17] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2012.

[18] H. Yun, S. Gondi, and S. Biswas, "Protecting memory-performance critical sections in soft real-time applications," *CoRR*, vol. abs/1502.02287, 2015. [Online]. Available: http://arxiv.org/abs/1502.02287

[19] 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 *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2014. [Online]. Available: http://dx.doi.org/10.1109/ECRTS.2014.20

[20] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, Dec 2001.