Data Bus Slicing for Contention-Free Multicore Real-Time Memory Systems

Javier Jalle*,†, Eduardo Quiñones†, Jaume Abella†,
Luca Fossati§, Marco Zulianello§, Francisco J. Cazorla†,‡
†Barcelona Supercomputing Center *Universitat Politècnica de Catalunya
§European Space Agency ‡Spanish National Research Council (IIIA-CSIC)

***Abstract*—Memory access contention is one of the main contributors to tasks' execution time variability in real-time multicores. Existing techniques to control memory contention based on time-sharing memory access do not scale well with increasing complexity of multicores, leading to a rapid increase of WCET estimates. This is due to fact that requests from different tasks interleave in the access to memory, and for each of its requests a task has to make worst-case time allowances to account for the memory state left by the previous request, that may belong to a different task. In this paper, we propose a memory organization that controls contention by dividing the data bus into narrower independent data buses, thus removing conflicts among different tasks accessing memory. While narrower data buses require extra transfers, they allow exploiting memory locality, hence only slightly affecting average performance. Our evaluation on a solid space case-study shows that the proposed memory organization provides contention-free memory access facilitating timing analysis and tightening WCET estimates.**

I. Introduction

The migration to multicores in real-time systems offers the benefits of reducing space, weight and power since more functionality can be integrated onto less hardware. However, multicore transition challenges timing analysis. The use of single-core timing analysis tools [42] – developed and refined for decades incurring enormous costs – may fail to meet the requirements of multicores due to inter-task contention. Further, mixed-criticality applications [38] challenges their timing analysis. In this scenario, time composability – that stipulates that the worst-case execution time (WCET) estimate derived for a task does not depend on its corunner tasks – eases timing analysis and it is key to enable incremental verification, simplifying application integration. Therefore, time composability [12][33] must be preserved when designing the computing platform for real-time systems.

Future mixed-criticality applications will manage an increasing number of sensors, and implement complex value-added functionalities in the space [41], avionics [7] and automotive [36] domains. For instance, in the space domain, that is the main focus in this paper, the fact that missions are becoming more autonomous [15] accentuates the trend towards larger amounts of data processed per second in the processor, which ultimately increases memory bandwidth requirements. This also makes contention in the access to memory to be one of the main factors impacting both WCET estimates for tasks running in the multicore [4][29] and time composability.

In high-performance domains, memory systems are usually organized into pluggable memory modules which require little space on the motherboard, ease expansion and allow replacing memory modules in case of failure. The main timing characteristic of memory modules is that they are divided into several – up to 8 in current designs – ranks that can be exploited to control contention in memory [23]. However, pluggable memory modules are not common in embedded real-time systems. For instance, in the space domain, having pluggable elements would negatively affect reliability due to the harsh conditions in which these systems are often deployed – e.g. with high vibrations and acceleration. Further, in many real-time systems, including the space domain, it is infeasible to extend or replace failing memory modules once the system is deployed, which diminishes the need for pluggable modules. As a result, memory is usually organized into different memory chips that are soldered to the board. This memory organization typically implements a single rank.

Motivation: Our proposal builds on the observation that, when memory is shared (sliced) in time, the impact of memory contention on WCET is high. This occurs because in a mixed-criticality environment where tasks are likely developed by different partners and under different criticality levels, although a task enjoys exclusive access to the memory bus, when the bus is relinquished and assigned to a new task, the latter can make no assumption on the state left in memory by the previous task. For instance, the new task does not know whether the previous access was a write or a read, and since write-to-read and read-to-write timing constraints are greater than read-to-read and write-to-write, the new task has to assume that the previous task made an operation on the opposite direction (i.e., the previous task carried out a read and the new one a write or vice versa). As a result, the new task makes – for each request – worst-case time allowances to account for the memory state left by the last task using it. This maintains time composability though increases the allocations made in the new task's WCET estimate.

Contribution: Instead of allowing that each task accesses all memory devices simultaneously to enjoy a wide data bus and time-sharing the memory system, which leads to pessimistic WCET estimates, our proposal uses narrower data buses (private to each task) each providing access to different memory devices, while the address bus is shared. With our space-sharing approach we heavily reduce the memory contention among tasks by dividing the available memory data bus to provide independent memory data buses accessing different memory chips that do not interfere each other. Dividing the data bus requires no changes on the memory device but only few extra processor pins. This approach requires more memory transfers to complete a memory transaction which reduces available per-core bandwidth when compared with wider data buses. However, this penalty is reduced since the locality of the memory row-buffers is completely exploited because the extra transfers needed are always sequentially ordered. The net result is a small reduction on average performance that pays-

off for having a heavily reduced memory contention, leading to tighter WCET estimates. Tight WCET estimates are key in critical real-time systems, since they are directly related to hardware provisioning made to ensure that time critical tasks finish before their deadline.

Our memory organization: (i) significantly reduces the impact of memory-contention on WCET estimates, which are less affected by high core counts and processor frequency in comparison to time-sharing approaches; (ii) uses standard memory technology, i.e. commercial off-the-shelf (COTS) memory chips, deployed in other domains; (iii) maintains time composability allowing to use single-core timing analysis tools; and (iv) provides flexibility for mixed criticalities such that the memory can be easily configured to satisfy different real-time and performance needs.

We assess the benefits of our proposal and compare it with state of the art memory controllers both qualitatively and quantitatively in the context of a space case study with real space applications on a simulator validated against real boards. We focus on an on-board-soldered memory setup where chips are organized as a single rank and are accessed in parallel through a single channel. Our proposal reduces contention by 35% and 51% with respect to private and shared bank schemes respectively, having a 2% average performance penalty with respect to other memory controllers.

II. Background

DRAM devices are connected to a *channel*, i.e. data and command bus. In order to provide a wide read/write operation (e.g., 64-bit), a channel consists of several memory devices (e.g. 4) that are accessed in parallel. Each of them provides narrower data width (e.g. 16 bit) and contains several banks, with only one bank at a time accessed in a given channel. Since memory operations take several cycles, different banks can be active simultaneously processing commands. Each bank contains a row buffer from which data are read and written. The memory controller is in charge of scheduling the different requests coming from the same or different processors and translating the requests into the appropriate commands. Also the memory controller defines the mapping of physical addresses from the processors to the actual memory blocks in the memory devices.

There are two ways of managing memory rows (pages) from the point of view of the row buffer: close-page and open-page [17]. The former precharges a row immediately after the column access and open-page leaves the row open in the row buffer to exploit the locality of future accesses, called *Row Buffer Locality*. Under close-page, all requests perform the same actions: activate, column access and precharge. In an open-page scheme, depending on if the access is a *row-hit* or a *row-miss*, a request behaves differently. If the access is a *row-hit* it accesses the same row as the previous access, and hence it can directly perform the column access. In the case of a *row-miss*, the request precharges the actual row and activates the new one before the column access.

All commands sent to the DRAM devices satisfy certain timing constraints [17]. The most important parameters are the column read latency t_{CAS}, write latency t_{CWD}, activate latency t_{RCD} and precharge latency t_{RP}. A detailed list of the timing constraints can be found in [17]. We do not consider refresh operations in this paper. Their impact on execution time is limited and can be bounded as shown in [5].

A. Related Work

Several approaches deal with the high impact of memory contention. Some works use a *shared-bank* scheme in which data are mapped across all memory banks. To exploit the bank-level parallelism, access to banks is interleaved so that every request accesses all banks simultaneously in a pipelined fashion, hence removing bus conflicts among memory requests. In particular, this scheme has been used in [4] with CCSP (Credit-Controlled Static-Priority) scheduling policy to accommodate different latency and bandwidth requirements. It is also found in [29] with Round-Robin policy or in [11] with reconfigurable TDMA. Most of them implement a close-page policy that reduces the memory jitter at the cost of preventing the exploitation of the row-buffer locality. In [10] authors exploit this locality with a conservative open-page policy. A similar technique to the bank interleaving is used in [9] to interleave memory channels in order to exploit channel-level parallelism besides bank-level parallelism. Accessing individual DRAM devices in a memory module instead of all devices at the same time, has been proposed to improve energy efficiency with a small impact on system performance [3], [40] or storage efficiency [43]. However, contention is neither discussed nor evaluated in those works.

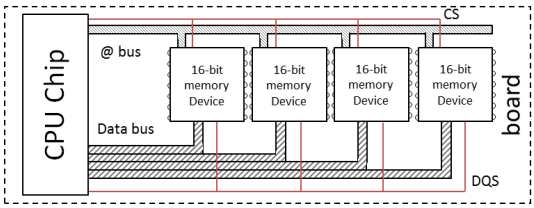
Private-bank scheme reduces contention by providing each core exclusive access to certain banks, effectively removing bank-conflicts. In [34], authors use a close-page TDMA memory controller that effectively removes memory contention. In [21], authors derive bounds on the interference on a COTS processor with an open-page FR-FCFS (First-Ready First-Come First-Served) memory controller [35] . In [31], authors use a FIFO policy instead of FR-FCFS, removing the reordering effect that FR-FCFS introduces, in order to be able to derive tighter bounds on the request latency. Authors in [23] further reduce write-to-read and read-to-write contention in [31] by switching between several ranks.

III. Motivation

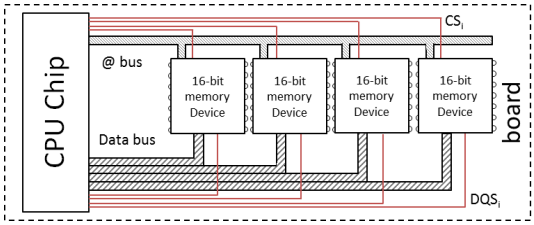
In many real-time systems, memory, which suffers stress conditions and it is not upgraded or fixed during operation, is not organized into modules but memory chips are directly soldered on the board. This setup is called *memory-down* [13] and is shown in Figure 1a. Furthermore, the use of multiple ranks is challenged in embedded memory-down systems since it requires extra space and weight on the board [16], critical parameters for embedded systems. For these reasons, single-rank memory-down configurations are used in the space domain which is the focus of this paper.

In the absence of multiple ranks, we observe *bank-conflicts* and *channel-conflicts*. *Bank-conflicts* are caused by the timing constraints between commands going to the same bank, like the act-to-act constraint (t_{RC}), precharge time (t_{RP}) and by row-misses when using an *open-page* policy. *Channel-conflicts* are caused by the timing constraints between commands going to different banks in the same devices, thus, using the same channel. For instance the write-to-read (t_{WTR}) and read-to-read (t_{BURST}) constraints affecting the timing of read/write requests even if they target different banks.

With a shared-bank scheme [27], in which tasks are mapped into the same banks, bank-conflicts dominate over



(a) Original setup sharing *CS* and *DQS*



(b) Proposed setup with different *CS* and *DQS*

Fig. 1: Original and proposed 64-bit memory-down systems

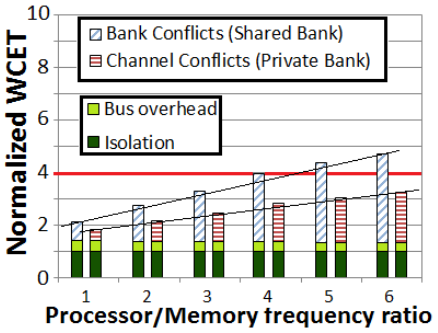
channel-conflicts. Meanwhile, with private banks, in which tasks use their own banks, as proposed in [31][34], bank-conflicts are removed. This is better illustrated in Figure 2 that shows the WCET estimates obtained by means of simulation for `cacheb` EEMBC benchmark when it runs in a multicore architecture as we increase the number of cores and the ratio between processor and memory frequency. For this experiment, in which the only shared resources are the processor on-chip bus and the main memory, we use shared-bank [27] and private-bank [31] solutions. More details on the experimental setup are shown in Section VI.

We breakdown the WCET estimates into the WCET in isolation, the bus overhead and the bank and channel conflicts. Results are normalized to WCET in isolation so that WCET estimate increments higher than the number of cores provide diminishing returns, so that it is better to run tasks in isolation to prevent them sharing the memory system.

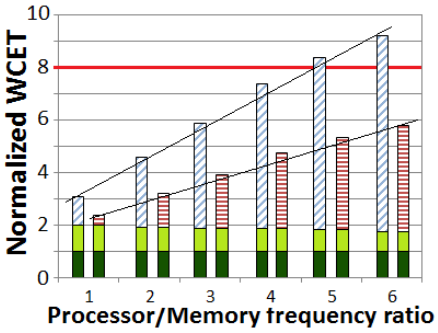
In Figure 2 we observe that WCET estimates increase rapidly with the number of cores and the processor-memory frequency ratio, since the larger the number of contenders, the higher the interference and the faster the processor, the bigger bottleneck the memory becomes.

- With the shared-bank scheme [27] (first bar in each pair) bank-conflicts have higher impact than channel-conflicts and mask them. In this setup we observe that WCET estimates rapidly increase making memory the main contributor to WCET.
- With a private scheme [34] [31] (second bar in each pair) bank-conflicts can be avoided since each task is assigned its own private bank. Yet, the WCET degradation is still high because of channel-conflicts.

Hence, memory impact of WCET is high either with private-bank or shared-bank. Interestingly, an effective way to remove conflicts would be by deploying a multi-channel memory system [9] in which each task uses a private channel with different memory interfaces and devices. In this case, the memory system is neither shared in time nor in space, since each task uses different interfaces and devices at the same time.



(a) 4 cores



(b) 8 cores

Fig. 2: Memory contention of the EEMBC benchmark `cacheb` with shared and private bank setups.

However, such a solution would require hundreds of processor signals to have separate memory interfaces.

For instance, the NGMP processor [2] – a candidate processor to be used in future space missions and our reference processor in this paper – uses a 625 pins package, out of which around 300 are usable. *A complete multichannel solution would require 544 pins which is completely out of reach*. Hence, for scalability reasons, a unique memory interface (channel) needs to be shared between cores due to the limited amount of processor signals. To reduce the impact of contention on shared memory interfaces, current solutions time-share it among the tasks running in the multicore.

A. Requirements

Memory-down systems face several requirements to enable their safe use in multicore systems as we describe next in this section. Although these are presented in a space-centric manner, other domains such as automotive or avionics have similar ones, and hence similar solutions apply.

Tight WCET estimates. As seen before, bank and channel conflicts have high impact on WCET estimates, specially bank-conflicts. These conflicts should be avoided in order to derive tight WCET estimates.

Factoring memory contention in WCET estimates. The memory system has to simplify factoring the impact of memory contention in the WCET estimates provided by timing analysis tools without requiring the latter to be changed.

Mixed-criticality. On-board space systems comprise two criticality levels [30], control and payload. Control tasks have real-time constraints and require WCET estimates. Payload

tasks have soft (or no) real-time constraints and are high-performance driven. The memory design has to provide *mixed-criticality flexibility*, so that it provides high average performance when no real-time execution is required and guaranteed performance for real-time tasks.

COTS technology. Using of COTS technology reduces non-recurring costs. In the space domain, while processor design can be changed to accommodate hardware support for real-time systems [8], non-customised (COTS) memory devices are preferred in order to reduce costs.

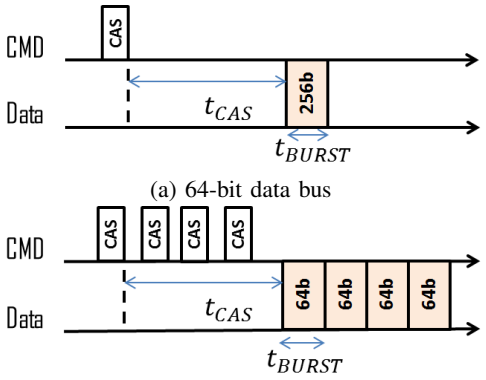
IV. Private-Data bus Shared-Command bus (PDSC)

We propose PDSC, which removes contention between tasks by dividing the available data bus into private independent data buses for each task, that target different memory devices, while sharing the command bus. PDSC removes contention between tasks since they are prevented from sharing the same data bus. The split of the data bus is done in a smart way such that COTS memory chips can be used. In particular, in our reference processor, comprising four cores and a 64-bit memory interface, PDSC divides the 64-bit data bus into four 16-bit buses, one for each core. The command bus is time-multiplexed for the different data buses, which means that the address and command bus is shared between cores, so that contention, or inter-task interferences, only happens on the command bus. Such contention is minimal, since only 1 bus cycle ($t_{CMD} = 1$) is needed to send commands. Hence each memory device can receive one command every four cycles.

Splitting the data bus incurs low overhead in memory-down configurations. The Chip Select (CS) and Data Query Strobe (DQS) signals indicate to a DRAM device when a command and the data are ready respectively. In the original 64-bit bus design, one CS and DQS signal is used for all the 4 devices so that they work simultaneously, as seen in Figure 1(a). In our 16-bit design with four buses, each one accessing one memory device, we only require to wire three extra CS_i and DQS_i signals, one for each extra data bus added, as presented in Figure 1(b), which follow the same physical route as the original CS and DQS signal. These extra signals require few extra processor pins, that in our case incur a small cost since the processor we model, the NGMP [2], has about a dozen pins available. This is in contrast to a complete multichannel solution that in the NGMP would require 544 user-defined pins, hence making it infeasible.

PDSC requires more transfers to complete a memory transaction: A memory transaction brings 256 bits, which corresponds to a last-level cacheline, and requires only one transfer on the 64-bit bus since the memory has a burst of 4, i.e., $4 \times 64 = 256$ bits, as seen in Figure 3(a). With a 16-bit bus PDSC needs four memory transfers to get the 256 bits, i.e., $4 \times (4 \times 16) = 256$ bits. For instance, with a 64-bit bus we need one transfer to get the 256-bit memory block at `0x00` (bytes `0x00` to `0x1F`). With a 16-bit bus we need four transfers, at `0x00`, `0x08`, `0x10` and `0x18` that bring 64 bits each.

Hence, for every memory transaction, PDSC performs four sequentially ordered transfers, which allows exploiting the memory locality that offers the row-buffer using an open-page policy [31]. The first transfer can be a row-hit or row-miss, depending on the previous transaction, while the following



(a) 64-bit data bus

(b) 16-bit data bus

Fig. 3: Equivalent transfers.

three transfers are always row-hits. Since they are always row-hits, their latency $t_{CAS} = 6$ (memory cycles) overlaps with the first transfer, thus performing those transfers immediately one after the other. Note that we rely on the fact that any cacheline is mapped consecutively on a single memory row. In this case, every extra transfer only has to consider the command bus multiplexing and the transfer time on the bus ($t_{BURST} = 2$ cycles), since the read or write latency is overlapped between sequential transfers. An example of these accesses is shown in Figure 3 for a read.

The hardware cost of PDSC is similar to most tailored memory controllers [29][31][4], also requiring multiple queues and a command translator, which converts requests into the corresponding commands and keeps track of the open rows for each data bus and refresh counters. PDSC also deploys a command bus scheduler to share the command bus between each data bus.

Despite the use of multiple ranks is not common in memory-down setups, PDSC can also work with multiple ranks. While having ranks reduces channel conflicts [23] there are still conflicts that are caused because of the rank switching time. PDSC would allow to further remove contention, while having multiple ranks on each private data bus, which allows increasing memory bandwidth.

A. Configuration Options

PDSC can be configured to provide either increased average performance or increased guaranteed performance to respond to the asymmetric needs of mixed-criticality applications. We propose different configurations (modes) for PDSC, that use different addressing schemes which are easily configurable by software: *RT*, *HP* and *MIX*.

RT uses four independent 16-bit data buses to provide guaranteed performance, intended for real-time tasks. PDSC causes, in this mode, an average performance penalty for dividing the data bus as stated before.

In *HP* the average performance penalty of *RT* is reduced by interleaving the four 16-bit data buses [9]. When the four 16-bit data buses are interleaved, the four accesses needed per transaction are carried out by interleaving one access per data bus, thus using all of them at the same time. In this scenario, average performance is the same as with the original 64-bit

bus, except for the time switching of the command bus that introduces a small penalty (2% in our setup).

Under the *MIX* configuration, we can have half of the memory working with independent channels, and the other half working with interleaved channels, so that tasks targeting guaranteed performance can be scheduled with tasks targeting average performance. For that purpose, two of the data buses are accessed as a private data bus each one, and the other two data buses are interleaved. The latter two data buses are accessing the same data and suffer interference.

As explained before, the memory space is divided into four equal segments, one for each data bus in the system. The data bus partitioning is done by software, similar to the software bank partitioning [24] and requires a Memory Management Unit (MMU). The OS configures the MMU so that each core targets the appropriate memory region: real-time applications target their own data bus. For other applications the OS decides which data bus they can access by configuring the MMU.

The flexibility obtained by interleaving the data buses incurs a small hardware cost due to the complexity introduced in the data path that connects the data bus with the corresponding memory request. Further, PDSC modes can be easily configured at boot time: each processor instance is used as part of a different system function. Each of those functions may have different high performance and real time requirements that can be accommodated with PDSC through its modes. PDSC could also be configured dynamically though this would cause data movement among memory regions.

B. Scalability

The presented PDSC memory controller fits the needs of our space case study. Processor designs [20][26] with higher core counts usually deploy clustered architectures, which organize processor resources into "islands of execution", i.e., subsets of processors. In general, clustered architectures provide better scalability than flat architectures, higher degree of isolation across clusters, which is good for real-time, and reduced complexity. For this reason, when it comes to scaling to larger systems, we consider a clustered processor design.

We divide the available memory interface into several data buses and assign one bus per cluster of cores. Each cluster has a private data bus, so contention occurs between cores within a cluster but not across clusters. Contention between cores within a cluster can be handled by using predictable arbitration policies such as round-robin [29] and reduced by using private-bank schemes [31][34], since these techniques are orthogonal to our proposal.

Let assume a 64-bit memory interface that can be divided into 4x16-bit buses or 8x8-bit buses. The 4-bit bus case is not considered since there are no available COTS memory devices with such bit width. For the 8-core multicore we deploy 2-core clusters with 4x16-bit buses or 1-core clusters with 8x8-bit buses. Having more buses means that fewer cores are assigned per cluster, which reduces the interference within clusters. However, by dividing the memory interface into smaller buses, we also make the overhead of the extra memory accesses required and the command bus time switching bigger. This ultimately leads to a compromise between the number of data buses, the available memory interface width and the overhead of smaller data buses.

With wider memory interfaces, e.g., 128 bits, scaling to larger core counts becomes easier. For instance, a 128-bit memory interface can be divided into 8x16-bit buses with clusters of 2/4 cores to support 16/32 cores. Note that current designs in real-time systems have lower number of cores, such as AURIX [14] with 6 cores in automotive and P4080 [25] with 8 cores in avionics.

It is worth highlighting that PDSC also applies to other real-time domains, besides space, with needs for high memory mixed-criticality performance (e.g. automotive and avionics domains). PDSC, with its flexibility and scalability, would help handling the memory contention issue.

V. Timing Analysis

Cores accessing a private data bus do not suffer bank- and channel-conflicts. The contention on the command bus is removed with the time switching that allows to send one command every four cycles. Under this scenario, memory access times are composable so that access times, and not only worst-case access times, do not depend on the rest of the tasks accessing memory, thus having a predictable latency. Composable access times greatly simplify timing analysis since no special timing analysis techniques are needed and the same single-core analysis tools can be reused for multicore systems, analyzing tasks in isolation. Note that the contention in other on-chip shared resources such as shared caches is discussed in Section VI.

The clustering design approach in which several cores share a private data bus, used for larger core counts, requires deriving a latency bound that accounts for the contention. The latency, τ, of a request arriving at the memory controller can be divided[21] into intrinsic latency and request interference delay, $\tau = \tau_{req} + \Delta$. The former accounts for the time it takes the request to be processed once it is granted access, τ_{req}. The latter accounts for the impact of contention, Δ.

The request latency, τ_{req}, corresponds to a request for an open-page row buffer policy, $\tau_{open-req}$ and the three row-hit accesses, τ_{rh}, caused by dividing the 64-bit bus into four 16-bit buses, see Equation 6. On the event of an access to a non-open row (see Equation 2), we need to activate the row first, with t_{RCD} latency. Once the row is open, the request latency covers the column access, t_{CAS} or t_{CWD}, and transferring the data, t_{BURST}, which coincides with the latency of a row-hit. A row-hit, see Equation 1, happens when the requested data is on the open row. Finally, for a row-miss (Equation 3), which happens when a different row is open in the row buffer, the row is first precharged, with t_{RP} latency, before being activated. These latencies are:

$$\tau_{hit-row} = max(t_{CAS}, t_{CWD}) + t_{BURST} \quad (1)$$

$$\tau_{closed-row} = t_{RCD} + \tau_{hit-row} \quad (2)$$

$$\tau_{miss-row} = t_{RP} + \tau_{closed-row} \quad (3)$$

$$\tau_{open-req} = \begin{cases} \tau_{hit-row} & \text{if row-hit} \\ \tau_{closed-row} & \text{if closed-row} \\ \tau_{miss-row} & \text{if row-miss} \end{cases} \quad (4)$$

$$\tau_{rh} = 3 * \max(t_{BURST}, (M-1) * t_{CMD}) \quad (5)$$

$$\tau_{req} = \tau_{open-req} + \tau_{rh} \quad (6)$$

The interference (delay), Δ, that other requests can generate corresponds to the *channel-conflicts*. In our memory

controller there are only two possible sources of interference, $\Delta = \Delta_{intra} + \Delta_{inter}$: *intra-channel* interference, Δ_{intra}, and *inter-channel* interference, Δ_{inter}.

– *intra-channel* interference is originated in the channel FIFO queue and caused by previous requests. This happens because the data bus is shared across several tasks. In that case, a private-bank scheme is used. Assuming that the interference that such a request suffers can be split into the interference that each of those commands suffers independently when accessing the command and data buses [21][18]:

$$\Delta_{intra} = \begin{cases} \Delta_{R/W} & \text{if row-hit} \\ \Delta_{ACT} + \Delta_{R/W} & \text{if closed-row} \\ \Delta_{PRE} + \Delta_{ACT} + \Delta_{R/W} & \text{if row-miss} \end{cases} \quad (7)$$

A CAS/CWD (R/W) command is delayed in the worst-case by another column command sent in the opposite direction, which corresponds to the write-to-read, t_{WTR}, and read-to-write, t_{RTRS}, timing constraints [18]. The time between two ACTs to different banks is limited by t_{RRD}, and a maximum of four ACTs can be issued during the t_{FAW} time-frame, to restrict the peak current. The ACT command is then interfered in the worst-case by other ACT commands, due to ACT-to-ACT timing constraints. In the last case, the worst-case interference happens when the other command is an ACT command that is the fourth consecutive ACT so that t_{FAW} does not allow the actual ACT to be scheduled. A PRE command can only be interfered by other commands using the command bus, which is given by the time between commands, t_{CMD}:

$$\Delta_{PRE} = t_{CMD} \quad (8)$$

$$\Delta_{R/W} = max(t_{CWD} + t_{BURST} + t_{WTR}, t_{CAS} + t_{BURST} + t_{RTRS} - t_{CWD}) \quad (9)$$

$$\Delta_{ACT} = max(t_{RRD}, t_{FAW} - 3 * t_{RRD}) \quad (10)$$

– *inter-channel* contention is due to the command bus time-switching. Each command suffers interference from up to $M - 1$ commands due to the time-switching. Depending on access type (row-hit, row-miss, closed-row), different number of commands are sent, each of which is affected by contention:

$$\Delta_{inter} = \begin{cases} (M-1) * t_{CMD} & \text{if row-hit} \\ 2 * (M-1) * t_{CMD} & \text{if closed-row} \\ 3 * (M-1) * t_{CMD} & \text{if row-miss} \end{cases} \quad (11)$$

Δ as computed above can be easily factored in WCET estimates by adding it as extra delay required to compute each memory access.

VI. Evaluation

A. Experimental setup

We use a solid evaluation setup comprising real space applications and a simulator whose accuracy we assessed by comparing its average performance against a real NGMP implementation, the N2X [1] board. Our results for EEMBC Automotive benchmarks showed a deviation in terms of accuracy of less than 3% on average and for the space NIR HAWAII benchmark [19] the inaccuracy reduces to 1%.

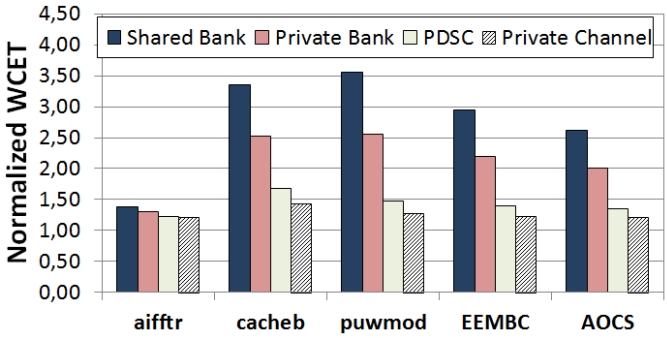


Fig. 4: Normalized WCET estimates

Platform. We use a modified version of the SoCLib [37] simulator framework that models a NGMP [2] running at 450MHz. The NGMP comprises 4 cores, an on-chip bus, connecting cores to the L2 cache, and an on-chip memory controller. Each core has its own private instruction and data caches, while the second level (L2) cache is split among cores. The first level caches are 16KB, 4-way with 32-byte lines. From the L2 each core receives one way of the 256KB 4-way L2. All caches use LRU replacement policy. The on-chip bus uses a round-robin arbitration scheme. With DRAMsim2 [39] we model a 2-GB one-rank DDR2-667 [22] (processor to memory frequency ratio is 3) and a 64-bit bus.

Timing analysis. We derive WCET estimates using measurements with hardware support to force requests to work on their longest latency for the on-chip bus [28]. The L2 cache has fixed access times, since it is split among cores. Single-core timing analysis tools can be used with *PDSC* since it offers composable memory access latencies. Thus, the memory has the same behavior as in a single-core system.

Real Applications. We use real space applications, control and payload [30]. The former require real-time execution and are designed to meet requirements in the worst case, while the latter are high-performance driven. As control application we use the Attitude and Orbit Control System (*AOCS*) from the EagleEye project [6]. AOCS contains the Guidance and Navigation Control system in charge of the correct position and orbit of the spacecraft. It is one of the most critical systems of a spacecraft, since a wrong position or orbit could mean the complete loss of the spacecraft. As payload we use the On-board Data Processing (*OBDP*) which contains the algorithms used to process raw frames coming from the state-of-the-art NIR HAWAII detector [19], already used on real projects, like the Hubble Space Telescope to detect cosmic rays.

For the evaluation in other application domains we use the EEMBC Autobench suite [32], which mimics some real-world automotive critical functionalities.

B. Experimental results

WCET. We compare PDSC with a *shared-bank* approach [29] and a *private-bank* approach [31]. We also consider a multichannel solution in which each core has its own 64-bit memory channel, i.e., *private-channel*, thus no memory contention (i.e., bank and channel) is observed, only the bus

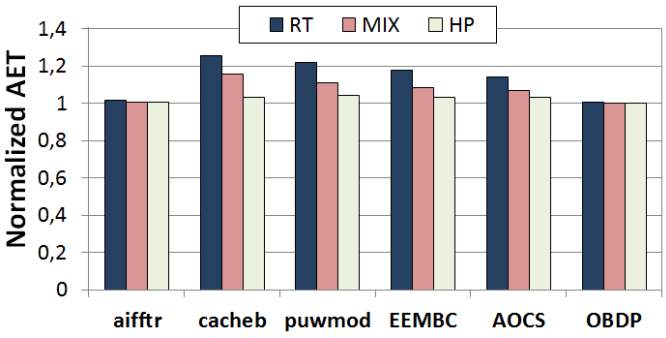


Fig. 5: Normalized Average Execution Time.

contention is present. Despite such a solution would require hundreds of extra processor pins[1] – posing an unaffordable cost for our reference processor – we use it as an optimal reference design since it removes contention because channels are completely independent.

Figure 4 shows WCET estimates normalized with respect to the WCET obtained with the original memory system, i.e., 64-bit data bus, when running in isolation without interference. These WCET estimates are obtained in isolation for each task and take into consideration the worst-case that any workload this task runs in could generate. Results are shown for AOCS kernel, three representative EEMBC with varying cache behavior (`aifftr` with low L2 miss rate, `puwmod` with high L2 miss rate and `cacheb` with a L2 miss rate close to the average) and the average for all EEMBC benchmark suite. We observe that PDSC obtains 33% tighter WCET estimates for AOCS with respect to a private-bank scheme and a 49% with respect to a shared-bank scheme. In comparison with the private-channel, PDSC is only 10% worse than this ideal solution. Interestingly private-channel WCET estimates are 20% bigger than when the application runs in isolation which is caused by the contention on the on-chip bus. Similar results are obtained for EEMBC on average. The private-bank suffers a slowdown of 110% because, despite banks are private to each task (each one pinned to a different core), tasks suffer contention on the data bus – removed with PDSC.

On average, including all EEMBC benchmarks and AOCS, PDSC leads to 35% tighter WCET estimates than the private-bank scheme and 51% than the shared-bank. Those WCET estimates are 12% higher than those with the private-channel solution, which is caused by the command bus multiplexing and the extra transfers overhead, i.e. the impact of sharing the command bus and reduced data bus width. These overheads have much less impact than bank and channel conflicts in private- and shared-bank schemes. Overall, PDSC provides tighter WCET estimates than the other approaches and quite close to the multi-channel solution, which provides the tightest WCET estimates that can be achieved in this case.

Average Performance. PDSC has two operation modes involving payload applications: MIX and HP. We compare these modes with a 64-bit data bus, i.e., the setup leading to the highest average performance. For the *MIX* and *HP* we respectively use 2 and 4 interleaved 16-bit data buses.

Figure 5 shows that *MIX* and *HP* have an average performance overhead of 5% and 2% respectively on average (for all EEMBCs, AOCS and OBDP) in comparison to the 64-bit bus solution. The *HP* scheme is equivalent to the 64-bit original memory system except for the time switching of the command bus that leads to a performance loss as low as 2%. For completeness we also included *RT* in Figure 5, although it is the mode used with real-time applications in which WCET is the main optimization factor. On average its performance degradation is 11%.

Scalability. We compare different setups in which the memory bus is split into a different number of available buses. Buses are assigned to the different cores according to two possible setups: (1) same number of buses and cores and (2) fewer buses than cores. We do not consider the case of having more buses than cores, since this is unlikely to occur in reality. In setup (1), each core owns a private data bus, while in setup (2), cores are organized into clusters that use a private data bus , as explained in Section IV-B. Cores within a cluster use a private-bank scheme that reduces the interferences among them. In this case, the worst-case latency, derived as presented in Section V, is used for each memory access. This latency can be applied either directly with static timing analysis techniques or measurement-based timing analysis techniques by means of the worst-case mode [28] in case of measurement based techniques.

For this experiment we use a crossbar interconnect, since the bus becomes a bottleneck for 16 cores masking memory impact. Hence, in this experiment, in contrast to previous ones, the increment on WCET estimates is exclusively caused by the memory contention. Figure 6 shows the WCET estimates, on average for all benchmarks, for 4, 8 and 16 cores normalized to the single-core WCET obtained with the original memory system, i.e., 64-bit data bus. The X-axis shows two bus setups, 64b and 128b width. The former is split into 2, 4 and 8 buses and the latter into 2, 4, 8 and 16 buses. In general terms, we see that for all core counts (4, 8 and 16) WCET estimates are largely below 4x, 8x and 16x respectively with respect to the single-core WCET. If this were not the case, the WCET results of multicores would have diminishing returns due to the memory contention. Instead we see that with PDSC, WCET shows good scalability for large core counts and wide buses. Looking at per core-count results we find the following:

– For the 4-core setup, the 4-bus configurations, i.e., 4x16b and 4x32b, are the best performing ones, since there is no interference on the data bus. Note that the 4x16b configuration is the one analyzed in detail in the paper.

– For the 8-core setup, 8x8b buses do not improve the 4x16b configuration. This is so because the benefit of not having interference on the data bus (since there is a private data bus per core in the 8x8b configuration) is outweighed by the overhead of the extra transfers required. This does not happen for the 8x16b configuration because fewer extra transfers are required due to the wider data bus.

– For the 16-core case, 8x8b and 8x16b are the best performing ones for 64-bit and 128-bit interfaces respectively. In both cases the contention reduction outweighs the overhead of the extra transfers required. This is so because of the large impact of contention in the 16-core setup. On the other hand, 16x8b configuration presents diminishing returns, due to the overhead of the extra transfers required.

[1]Even a 16-bit multichannel solution would require more than hundred pins due to the per-core 32 bit address and control signals.

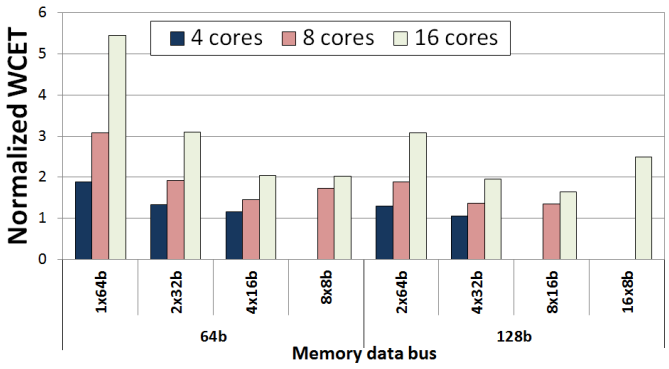


Fig. 6: PDSC scalability results.

VII. Conclusions

We have proposed a new memory organization, PDSC, for embedded real-time systems. PDSC builds on the observations that 1) in the embedded domain memory is unlikely to be pluggable because of reliability-related issues and due to the impossibility to upgrade/replace memory once the system is deployed; and 2) time-sharing memory has poor scalability with the core count and the processor-to-memory frequency ratio. PDSC uses space-sharing heavily reducing WCET, while providing enough flexibility to handle heterogeneous, i.e. high average and guaranteed performance, requirements of mixed-criticality multicore systems. Our evaluation on a solid space case study proves the benefits of PDSC to produce tight WCET estimates, 35% tighter than with private-bank schemes and 51% tighter than with shared-bank schemes with a minimum impact on average performance in a 4 core setup. PDSC also presents good scalability with core counts.

VIII. Acknowledgements

The research leading to these results has received funding from the European Space Agency under contract NPI 4000102880 and the Ministry of Science and Technology of Spain under contract TIN-2015-65316-P. Jaume Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

References

[1] Aeroflex Gaisler. *LEON4-N2X Data Sheet*, 2013.

[2] Aeroflex Gaisler. *Quad Core LEON4 SPARC V8 Processor - Data Sheet and Users Manual*, 2013.

[3] J. H. Ahn et al. Multicore dimm: An energy efficient memory module with independently controlled drams. *IEEE Comput. Archit. Lett.*, 2009.

[4] B. Akesson et al. Predator: A Predictable SDRAM Memory Controller. In *CODES+ISSS*, 2007.

[5] B. Bhat et al. Making DRAM refresh predictable. *Real-Time Systems*, 2011.

[6] V. Bos et al. Time and Space Partitioning the EagleEye Reference Mission. In *DASIA*, 2013.

[7] G. Edelin. Embedded systems at thales: the artemis challenges for an industrial group. In *ARTIST*, 2009.

[8] ESA Microelectronics. *http://microelectronics.esa.int/ngmp/*, 2015.

[9] M. Gomony el al. Architecture and optimal configuration of a real-time multi-channel memory controller. In *DATE*, 2013.

[10] S. Goossens et al. Conservative open-page policy for mixed time-criticality memory controllers. In *DATE*, 2013.

[11] S. Goossens et al. A reconfigurable real-time SDRAM controller for mixed time-criticality systems. In *CODES+ISSS*, 2013.

[12] S. Hahn et al. Towards compositionality in execution time analysis-definition and challenges. In *CRTS*, 2013.

[13] D. Hibler. Considerations for designing an embedded intel architecture system with system memory down (white paper), 2009.

[14] Infineon. AURIX Safety joins Performance.

[15] L. Innocenti. User cases: Active debris removal. In *ADCSS, ESA*, 2013.

[16] Intel. Intel quark soc x1000 platform: DDR3 dual rank memory down board layout guide (white paper), 2014.

[17] B. Jacob et al. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., 2007.

[18] J. Jalle et al. A dual-criticality memory controller (DCmc): Proposal and evaluation of a space case study. In *RTSS*, 2014.

[19] A. Jung et al. The H2RG infrared detector: introduction and results of data processing on different platforms. Technical report, ESA, 2012.

[20] Kalray. *MPPA 256 Many-Core Processor, http://www.kalray.eu/products/mppa-manycore*, 2013.

[21] H. Kim et al. Bounding memory interference delay in cots-based multi-core systems. In *RTAS*, 2014.

[22] Kingston. *KVR667D2S5/2G Datasheet*, 2011.

[23] Y. Krishnapillai et al. Roc: A rank-switching, open-row DRAM controller for time-predictable systems. In *ECRTS*, 2014.

[24] L. Liu et al. A software memory partition approach for eliminating bank-level interference in multicore systems. In *PACT*, 2012.

[25] J. Nowotsch et al. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *ECRTS*, 2014.

[26] M. Panic et al. Parallel many-core avionics systems. In *EMSOFT*, 2014.

[27] M. Paolieri et al. An analyzable memory controller for hard real-time cmps. In *Embedded System Letters (ESL)*, 2009.

[28] M. Paolieri et al. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, 2009.

[29] M. Paolieri et al. Timing effects of DDR memory systems in hard real-time multicore architectures: Issues and solutions. *ACM TECS*, 2013.

[30] M. Patte et al. System impact of distributed multi core systems. Technical Report ESTEC Contract 4200023100, European Space Agency, 2011.

[31] Z. Pei Wu et al. Worst case analysis of DRAM latency in multi-requestor systems. In *RTSS*, 2013.

[32] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

[33] P. Puschner et al. Towards composable timing for real-time software. In *STFSSD*, 2009.

[34] J. Reineke et al. PRET DRAM controller: Bank privatization for predictability and temporal isolation. In *CODES+ISSS*, 2011.

[35] S. Rixner et al. Memory access scheduling. In *ISCA*, 2000.

[36] Silabs. *http://www.silabs.com/Marcom%20Documents/Resources/automotive-applications-guide.pdf*, 2013.

[37] SoCLib, 2003-2012. http://www.soclib.fr/trac/dev.

[38] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, 2007.

[39] D. Wang et al. DRAMsim: a memory system simulator. *SIGARCH Comput. Archit. News*, 2005.

[40] F. A. Ware et al. Improving power and data efficiency with threaded memory modules. In *ICCD*, 2006.

[41] A. West. NASA Study on Flight Software Complexity. Final Report. Technical report, NASA, 2009.

[42] R. Wilhelm et al. The worst-case execution-time problem overview of methods and survey of tools. *ACMTECS*, 2008.

[43] D. H. Yoon et al. Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. In *ISCA*, 2011.