Towards an efficient cost function equation for DDR SDRAM interference analysis on heterogeneous MPSoCs

Alfonso Mascareñas González
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
alfonso.mascarenas-gonzalez@isae-supaero.fr

Jean-Baptiste Chaudron
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
jean-baptiste.chaudron@isae-supaero.fr

Frédéric Boniol
ONERA, Université de Toulouse
Toulouse, France
frederic.boniol@onera.fr

Youcef Bouchebaba
ONERA, Université de Toulouse
Toulouse, France
youcef.bouchebaba@onera.fr

Jean-Loup Bussenot
ONERA, Université de Toulouse
Toulouse, France
jean-loup.bussenot@onera.fr

***Abstract*—Real-time applications must finish their execution within an imposed deadline to function correctly. DDR memory interference on multicore platforms can make tasks overpass their respective deadline, leading to critical errors. Bandwidth regulators and SDRAM bank partitioning are examples of techniques used to mitigate or avoid this interference type. Another possibility is to optimally place tasks and memory on the platform, i.e., task/memory mapping optimization. The algorithms used for finding optimal mapping solutions work using a cost function that indicates the fitness of the found solution. In this work, we propose a DDR SDRAM cost function that estimates the worst-case execution time for a giving map, and hence, implementable in an optimization algorithm. Our cost function considers the DDR memory device operation, the SoC manufacturer memory controller, the heterogeneity of the platform and the characteristics of the tasks to map. The cost function is evaluated by measuring directly the interference from the heterogeneous MPSoCs Keystone II and Sitara AM5728 by Texas Instruments.**

***Index Terms*—Heterogeneous multicore platforms, Cost function, Memory interference, DDR SDRAM**

I. Introduction

In the real-time domain, memory interference on multicore platforms has been a topic of interest as they can be responsible for tasks deadline misses. Depending on the memory interference source, different techniques have been proposed for mitigating their impact and improving execution predictability, e.g., cache partitioning and cache locking for shared caches, bank partitioning and bandwidth regulators for SDRAMs. Another way is through task and memory mapping, i.e., the optimal allocation of tasks on cores and the memory on SDRAM banks. The algorithm in charge of finding the best allocation combination needs a cost function for evaluating the fitness of the maps. Therefore, the aim of this work is to present a DDR SDRAM cost function equation which can be used by optimization algorithms for retrieving an estimation of the real *Worst-Case Execution Time* (WCET) of a task map. This cost function must be time efficient and correctly model the behavior of the SDRAM interference cost. For such a purpose, we consider the DDR SDRAM device operations and timings, the *Memory Controller (MC) Reordering* (RO) effects, the heterogeneity of the platform and the task properties. To achieve time efficiency, the previous is described by a set of equations instead of inequations, as the latter would imply the implementation of solvers that is time consuming. The cost function evaluation is measurement-based. The heterogeneous MPSoCs Keystone II and Sitara AM5728 by *Texas Instruments* (TI) are used. The implementation of both platforms, the DDR memory interference cost function equation evaluation, and the implementation of the cost function in a task/memory mapping tool, can be found in this repository[1]. The remaining sections of this work are structured in the following way. Section II describes state-of-the-art procedures to analyze and upper-bound SDRAMs interference. Section III introduces the theoretical basis for the comprehension of the equations development for the cost function. Section IV lists the important aspects to consider for the experimental setup. Section V explains the development of the cost function equation. Section VI compares the theoretical and the measured values on the MPSoCs. Finally, Section VII summarizes the entire work.

II. Related Work

DDR SDRAMs bounding can be done through many different approaches: (1) request-driven analysis, (2) job-driven analysis, (3) hybrid analysis and (4) holistic analysis. Request-driven focuses on the interference that an individual memory

[1]https://github.com/ISAE-PRISE/sinteo

request receives. Afterwards, all the task requests are multiplied by this value. Job-driven is based on the number of interfering memory requests from other cores of the platform. The hybrid analysis computes the memory interference bounding by mixing the request-driven and the job-driven analysis. The holistic analysis focuses on the system rather than its parts. Work [3] bases its analysis on request-driven. They are able to evaluate their work in 144 different platform instance (e.g., enabling/disabling write batching, out-of-order execution, priorities) thanks to the use of MacSim and DRAMSim2. In the same category we can find [12], which assumes that the task under analysis can only manage one request at a time, i.e., considers in-order execution processors. In their simulations, they take into account the row hit ratio, as well as the loads and stores ratio. This is also considered in our work as we believe it is important for reducing the pessimism and, at the same time, feasible to do through platform measurements. Works [9], [13] provide request-driven and job-driven analysis. In fact, both works propose to consider the minimum of the results of the two bounds results to obtain the tightest upper bound. [9] carries out their work evaluation on a homogeneous multicore platform of four cores (Intel Core i7-2600 @ 3.4 GHz). They work with 2 ranks of 8 banks each, which is uncommon among the real platforms related work. Conversely, [13] uses Gem5 to simulate a quad core ARM Cortex A15 processor. It considers the effect of write batching based on the watermark policy (mentioned but not implemented in [9]). As for this, our work also considers the effect of batching but with a different policy based on separated read and write thresholds. The hybrid approach can be found in [4]. This work shows through MacSim and DRAMSim2 that with this approach the DDR3 memory interference bound can be further tighten. Another key characteristic of this work is the differentiation of the request types, e.g., read and write, row hits and row conflicts, similar to the idea in [12]. As in their previous work [3], they support a lot of MC/platform configurations which makes it very flexible. Work [2] targets holistic analysis to bound the memory contention. They make use of a three-phase execution model, distinguishing between the copy-in phase, execution phase and the copy-out phase. We remark the use of inequations and solvers to obtain the bounding like done in works [2], [4]. Unfortunately, the use of inequations is not adequate for task/memory mapping optimization because of their complexity and the time it takes to solve these.

III. DDR SDRAM Background

DDR3 SDRAM devices must comply with the JEDEC standard JESD79-3E [8] which defines the functionalities, electrical characteristics, packaging, etc. The main important components making up a DDR3 memory device are the ranks, the banks and the command, address and data busses. A rank is a collection of banks. A bank is a storage logic unit defined by columns and rows. The command bus is used for transmitting the commands to the banks. The address bus is used for selecting the memory location to access. The data bus is used for transferring the data.

A. DDR3 SDRAM Device Addressing

A convenient way to think about DDR SDRAM addressing is to see this memory as a set of banks. These banks, which are able to process commands in parallel, are made of a 2D array of memory locations defined by its column (X axis) and its row (Y axis). Attached to the banks there is a row buffer where the last accessed row is stored. Its purpose is to speed up data manipulation to that row. When accessing a different row than the one in the buffer, a row switch is produced. This entails a penalty as the row in the buffer has to be put back to its original position in memory and the new row to manipulate has to be brought to the buffer. Figure 1 shows an example of the bank-row-column organization. To intentionally use a given column, row or bank, we have to know how the physical address is decoded by the controller (see Table 2-5 in [5] or Chapter 15.3.4.12 in [7] for Keystone II and Sitara AM5728 respectively). For instance, the physical address 0x80283000 on Keystone II generates an output like: row 0x8028 (bits 31-16), bank 0x1 (bits 15-13), column 0x1000 (bits 12-0).

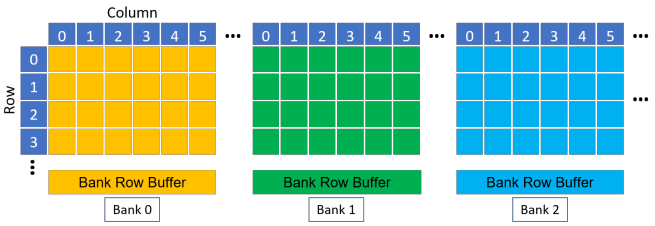


Fig. 1: SRAM organization: Columns, Rows and Banks.

B. Device Operations

Operation-wise, the memory is mainly based on a series of states, commands and timings. The current memory state and its possible transitions are determined by the commands. From the whole set of commands, special attention should be payed to *Read* (RD) and *Write* (WR). These two are burst oriented, meaning that the operation is applied to the given memory address and continue as a burst whose *Burst Length* (BL) is 8 columns. Note that DDRs are *Double Data Rate* and, therefore, the burst transmission time length is halved (i.e., $\frac{BL}{2}$). For a RD or a WR to be executed, the target row of a particular bank (see Section III-A) must be brought to the bank row buffer by executing an *Active* (ACT) command. If a different row of the same bank has to be accessed, then the actual information located in the row buffer must be first saved by executing a *Precharge* (PRE) command. Subsequently, an ACT of the new row is performed. There are specific transition timings associated to the commands execution (see Table I).

Data transfers are not done immediately. There is a delay between the command execution and the moment data is available on the data bus. This latency depends on the issued command. We can differentiate the *CAS Latency* (CL) and the *CAS Write Latency* (WL) for the RD and WR operations respectively. The elapsed time can be seen in Figures 2 and 3. Note how the MC issues the commands optimally by packing them, well-exploiting the DDR memory data bus.

TABLE I: Keystone II and Sitara AM5728 SDRAM values

SDRAM Feature	Keystone II MC@800MHz	Sitara AM5728 MC@533MHz
Number of Ranks	1 rank	1 rank
Number of Banks	8 banks	8 banks
Page Size	10 columns	10 columns
SDRAM Width	64 bits	64 bits
Burst Length (BL)	8 columns	8 columns
CAS to CAS delay (t_{CCD})	4 cycles	4 cycles
CAS Latency (CL)	11 cycles	7 cycles
CAS Write Latency (WL)	8 cycles	6 cycles
Write Recovery Time (t_{WR})	12 cycles	8 cycles
Write to Read (t_{WTR})	5 cycles	4 cycles
Precharge (t_{RP})	11 cycles	7 cycles
Read to Precharge (t_{RTP})	6 cycles	4 cycles
Active (t_{RCD})	11 cycles	7 cycles
Active to Precharge (t_{RAS})	28 cycles	19 cycles
Inter-bank Active to Active (t_{RRD})	6 cycles	7 cycles
Four Active Window (t_{tFAW})	24 cycles	28 cycles
Intra-bank Active to Active (t_{RC})	39 cycles	27 cycles

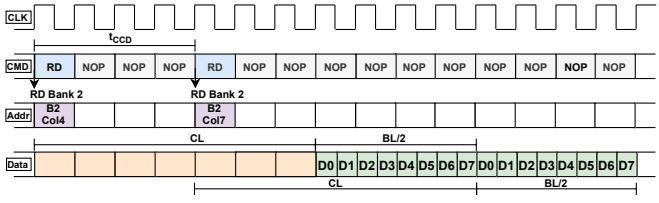


Fig. 2: Two consecutive read commands

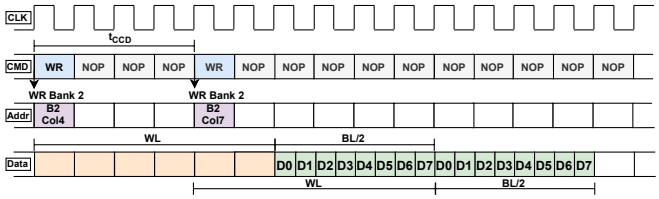


Fig. 3: Two consecutive write commands

Another important delay to consider is the data bus turnaround time caused by WR to RD or RD to WR state transitions. Figure 4 depicts the timing transitions for the WR to RD case when using two different banks. Figures 5 and 6 show the transitions from a RD to a WR command in the same and different banks.

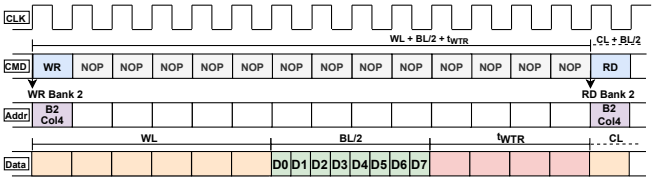


Fig. 4: Intra-bank write to read commands turnaround

Lastly, it must be mentioned that the transitions WR/RD to PRE and PRE to RD/WR due to a row switch produce a delay on the command bus. Figures 7 and 8 show the previous transitions respectively. The dashed lines in both figures represent a time skip.

Apart from the previous execution and transitions timings, there are time constraints to respect in order to achieve a well-

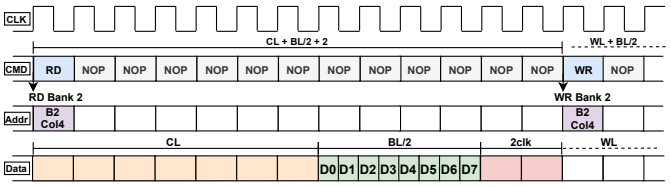


Fig. 5: Intra-bank read to write commands turnaround

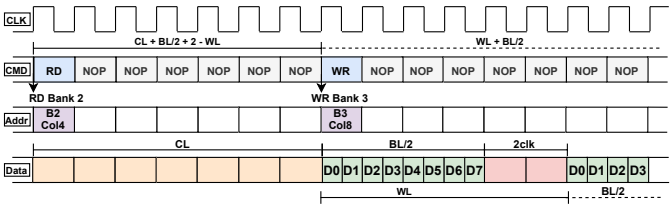


Fig. 6: Inter-bank read to write commands turnaround

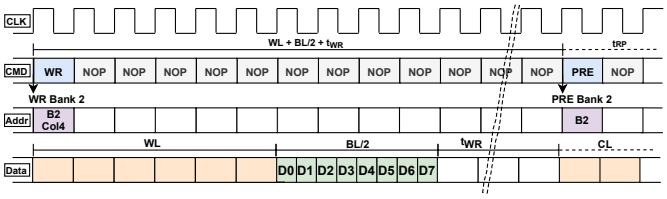


Fig. 7: Write to Precharge recovery time

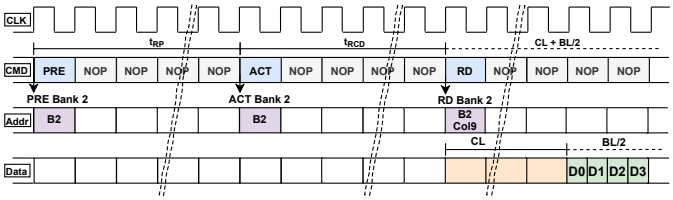


Fig. 8: Precharge to Read commands transition time

functioning of the memory. To avoid timing violations, it is paramount to consider the following constraints:

C1 **Consecutive active commands**: Two consecutive active commands must be executed within a time difference of t_{RRD}. Besides, only four ACT commands are allowed for a time window equal to t_{FAW}. Figure 9 describes this constraint.

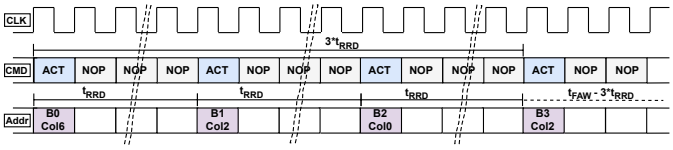


Fig. 9: Multiple active commands execution behavior

C2 **Active to Precharge**: For an intra-bank scenario, the elapsed time between the issue of an ACT and a PRE command must not exceed t_{RAS}.

C3 **Intra-bank active to active**: Similar to **C2**, the elapsed

time between the issue of two ACT commands must not exceed t_{RC} in an intra-bank situation.

C4 **RD/WR to Precharge**: The minimum RD to PRE spacing is given by t_{RTP}. Likewise, the WR to PRE minimum spacing time is t_{WR}. Figure 8 shows how this constraints is respected by waiting the write recovery time t_{WR} before performing a PRE.

C. Memory Controller Arbitration

In this work, the used DDR3 MC commands scheduling is determined by the First-Ready First-Come-First-Served (FR-FCFS) algorithm, which reorders the commands in the Command FIFO in order to maximize the total throughput. The following logic is followed [5]:

1) **Read prioritization**: For each master, the controller will advance a RD before an older WR if they are issued to a different block address[2] to reduce the platform cores stalling time. In the case both commands are to the same block, to maintain data coherency, the commands are dequeued in order of arrival (first in, first out logic).
2) **Opened row prioritization**: The commands pointing to an already opened row of a bank will be selected first to reduce the row switch cost. This is possible thanks to the implemented open-row policy. When there are no more commands with their corresponding banks opened, the deactivation of the current bank row (PRE) and the activation of the new row (ACT) is performed.
3) **RD/WR Batching**: The selected RD and WR commands are executed in batches or bursts of a defined size to reduce the number of turnarounds. The arbitration will switch to another batch type every time one of them has been executed. The oldest commands have priority.

The quantified effects of the arbitration logic can be seen in [10]. In terms of commands priority, the controller follows a Column-First scheduling policy where RD and WR commands (column accesses) are prioritized over ACT and PRE commands (row accesses). The execution order in the Keystone II and Sitara AM5728 DDR3 controllers are: RD, WR, ACT and PRE. RD and WR commands have the same priority after being reordered according to the arbitration rules.

IV. Considerations

In this work, the following points always apply:

- **Platform**: The heterogeneous platform considered for carrying out measurement-based evaluations of the cost function are the **Keystone II** model TCI6636K2H [6] and **Sitara AM5725** [7], both by Texas Instruments. The former MPSoC is made up of 4 ARM Cortex A15 cores and 8 C66x DSPs. However, in this work, we evaluate the results with 2 ARM cores and 6 DSPs, i.e., a total of 8 cores in parallel. The latter MPSoC is made up of a large number of different processing units of which we consider the 2 ARM Cortex A15 cores and 2 C66x DSPs.
- **DDR SDRAM**: The memories used are a DDR3 SDRAM version DDR3-1600K and a DDR SDRAM version DDR3-1066F for the Keystone and Sitara respectively. Both have a single rank and 8 banks. During the evaluation, 4 banks are used. The main properties of these memories are found in Table I.
- **No data caches**: We disable the L1D and L2 cache to increase the access frequency of the tasks to the DDR3 memory as well as getting rid of the L2 shared cache interference.
- **Bare-metal**: No operating systems have been used in order to gain entire control of the system and ease the data treatment by avoiding OS derived effects, e.g., preemption, frequency throttling.
- **Priority**: We consider a single level of priority, i.e., same criticality level. Therefore, the priority of the slaves of the platform interconnection is the same.
- **Starvation**: The initial values of the interconnection slaves starvation counters are increased to the maximum possible value. If these counters reach zero due to starvation, the priority of the slave is temporarily increased. The DDR3 SDRAM command priority raise counter (anti-starvation mechanism) is left at default setting.

[2]On Keystone II and Sitara AM5728 a block address is defined as a 2048 bytes length region.

V. Formal Description

A. Tasks Model

The functions developed to estimate the worst-case interference rely on some tasks properties. Tasks are defined as:

$$\tau_i := (C_i, \mathcal{A}_i(t), SP_i, S_i, ACOR_i, PE_i, B_i, T_i)$$

where:

- C_i: The WCET in isolation. $C_i \epsilon \mathbb{N}^+$.
- $\mathcal{A}_i(t)$: The DDR3 memory accesses cumulative distribution function. $\mathcal{A}_i(t) \epsilon \mathbb{N}^+$ and $t \epsilon [0, C_i]$.
- SP_i: The *Store Proportion* (SP) indicates the share of stores in $\mathcal{A}_i(t)$. $SP_i = 1 - LP_i$ where SP_i and $LP_i \epsilon [0, 1]$. LP_i is the *Load Proportion* of $\mathcal{A}_i(t)$.
- S_i: The number of row switches (ACTs) in isolation. $S_i \epsilon \mathbb{N}^+$.
- $ACOR_i$: The *Average Commands per Opened Row* (ACOR) defines the number of DDR commands that τ_i can execute before a bank row switch is produced by another task. $ACOR_i \epsilon \mathbb{Q}^+$. This parameter is described in Section V-A1.
- B_i: The bank to which τ_i is mapped to. $B_i \epsilon [0, N_b - 1]$.
- PE_i: The *Processing Entity* (PE) to which τ_i is mapped to. $PE_i \epsilon [0, N_c - 1]$.
- T_i: The period of τ_i, which is also the deadline. $T_i \epsilon \mathbb{N}^+$.

The set of tasks is defined as $\mathcal{T} = \{\tau_0, ..., \tau_{n-1}\}$. During the equations development, τ_i will be often used for representing the task under analysis and τ_j for representing an interfering task from $\mathcal{T}$. To denote the total number of tasks sharing a specific bank b, we define $tsb(b) = |\{\tau_n \epsilon \mathcal{T} | B_n = b\}|$.

1) ACOR: The internal composition of a task is complex due to the diverse instruction set and the instructions dependency which can cause stalls. Besides, the number of instructions execution that can be carried out while waiting for data to arrive to the processor from a previous instruction is limited, for example, by the instructions queue size (reservation stations) or data paths number. These facts make it difficult to theoretically compute the value for the open-row policy effect $ACOR_i$ for each task. A solution not involving a deep task disassembly analysis is to obtain this value through measures. The task whose $ACOR_i$ is to be obtained, is run in parallel with a saturating benchmark, e.g., stream of stores. Both tasks must work within the same bank and different rows. In this way, interfering row switches other than S_i are produced ($S_{i_{||}}$). $S_{i_{||}}$ is obtained using the DDR3 memory controller counters. Thus, the switches imposed by the interfering task are obtained. The task under analysis SDRAM accesses are divided by this imposed row activations, resulting in accesses per row switch. By doing this, it can be known the processor average DDR3 command accumulation capacity for a given task execution. Equation 1 describes the previous:

$$ACOR_i = \frac{\mathcal{A}_i(C_i)}{S_{i_{||}}} \tag{1}$$

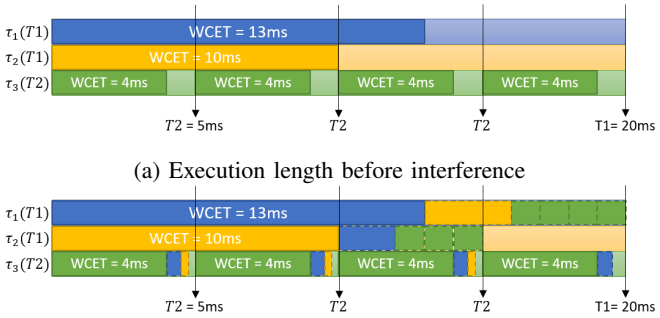
The idea behind Equation 1 is exemplified in Listing 1, which shows a fragment of benchmark *sb0* (see Table II). We can distinguish some DDR memory execution groups due to the instructions independence: (1) Lines 2 and 3, (2) Lines 5 and 7 and (3) Lines 9, 10 and 11. The average number of access per row active would be $(2+2+3)/3 = 2.33$ access/active. If we measure it using an interfering micro-benchmark as previously proposed, we obtain 2.46 access/active for the entire benchmark. This happens due to the open-row policy of the controller, that for efficiency purposes, executes together the commands in each group. The difference between both methods is very small. Hence, we consider the measurement-based method equivalent to the theoretical one.

Listing 1: sb0 benchmark disassembly fragment on an ARM Cortex A15

```
1  in0[i]= in0[i]+ in0[i]*in1[i];
2  80031a8c:    ldr         r1, [r8, r3, lsl #2]
3  80031a90:    ldr         r2, [r5, r3, lsl #2]
4  80031a94:    mla         r2, r1, r2, r2
5  80031a98:    str         r2, [r5, r3, lsl #2]
6  for (i=0; i<size; i++)
7  80031a9c:    ldr         r3, [r4]
8  80031aa0:    add         r3, r3, #1
9  80031aa4:    str         r3, [r4]
10 80031aa8:    ldr         r3, [r4]
11 80031aac:    ldr         r2, [r6]
12 80031ab0:    cmp         r3, r2
13 80031ab4:    blo         #0x80031a8c
```

B. DDR3 Memory Interference Cost Function

This cost function design relies on four aspects: (1) **DDR3 SDRAM device**, (2) **DDR3 Memory Controller**, (3) **Platform Architecture** and (4) **Application**. As a result of (1), we



(a) Execution length before interference

(b) Execution length after interference

Fig. 10: Biperiodical DDR3 interference example

permanently differentiate between the interference originated from inside the bank (intra-bank interference) and outside it (inter-bank interference).

1) Interference Time Interval: To begin with, we need to compute the number of interfering RDs and WRs executing before the task under analysis. To do so, we need to calculate the proportion of time in which a task is interfered by another. This proportion is dynamic in the sense that tasks will see their execution time increased due to the interference. Therefore, we need to recalculate this proportion until stabilization is achieved. In order to consider the tasks period differences, we compute the times the period of τ_j (T_j) entirely fits inside the execution time of τ_i (C_i). Finally, it must be noted that shorter period tasks with respect to τ_i may be able to execute again due to the experienced overhead introduced by τ_j. Figures 10a and 10b exemplifies this case by showing the overheads produced among the tasks τ_1 in blue, τ_2 in yellow and τ_3 in green. τ_3 is a task with a shorter execution period. It can be seen that due to the interference of τ_2 and τ_3, τ_1 total execution time exceeds 15ms. This causes τ_1 to be entirely interfered by the third and fourth τ_3 executions. The same applies for τ_2 that is interfered by three τ_3 executions instead of two.

To cope with the previous problems, we consider a recursive implementation described by Algorithm 1. The Relative Interference Exposition (RIE) shown in Equation 2 calculates the time length that τ_i is exposed to τ_j. The function calls the DDR3 interference cost function IC (Equation 12) for updating the C_i and C_j values.

$$RIE(\tau_i,\tau_j) = \left\lfloor \frac{C_i + \alpha * IC(\tau_i)}{T_j} \right\rfloor + min\left(\frac{C_i + \alpha * IC(\tau_i) - \left\lfloor \frac{C_i+\alpha*IC(\tau_i)}{T_j} \right\rfloor * T_j}{C_j + \alpha * IC(\tau_j)}, 1 \right) \tag{2}$$

Note that IC is expressed in MC cycles and C_i in core cycles. Then, the former has to be converted to the later by multiplying by a factor of $\alpha = freq_{core}/freq_{SDRAM}$ (i.e., the bus clock multiplier).

2) Number of interfering commands: The function named Periodic Generalized Number of Commands in Line (PGNCL) shown in Equation 3 estimates the total number of interfering

accesses. For this purpose, we make use of the cumulative DDR3 memory access distribution of the tasks. By using this equation, we make the assumption that all the interfering accesses affect the task under analysis.

$$PGNCL^X(\tau_i) = \sum_{\tau_j, j \neq i} \mathcal{A}_j(C_j * RIE(\tau_i, \tau_j)) \quad (3)$$

where X is intra or inter. Note that $PGNCL^{intra}$ and $PGNCL^{inter}$ consider those τ_j in the same ($B_j = B_i$) and different ($B_j \neq B_i$) bank than τ_i respectively.

3) Write and Read transmission cost: The two most common command types are WRs and RDs. These commands take $\frac{BL}{2}$ cycles for its data burst to be transmitted through the data bus. t_{CCD} cycles must pass for a command of the same type to be executed in the command bus. In normal conditions $t_{CCD} = \frac{BL}{2}$ (see Figures 2 and 3). In addition, we have to consider the intra-bank and inter-bank turnaround cases (see Figures 4, 6 and 5). The set of Equations 4 shows the Write Transmission (WT) and Read Transmission (RT) calculations with turnaround penalties which uses the values found in Table I. We don't consider the effect of ranks as only one is present in the platforms of this work. Please refer to [9], [12] for more information on the effect of ranks on turnaround transitions and the t_{RTRS} rank switch delay.

$$\begin{aligned} WT^{intra} &= WT^{inter} = WL + \frac{BL}{2} + t_{WTR} \\ RT^{intra} &= CL + \frac{BL}{2} + 2 \\ RT^{inter} &= CL + \frac{BL}{2} + 2 - WL \end{aligned} \quad (4)$$

4) Data Transmission Cost: The MC executes a specified number of RDs and WRs in batches to avoid the turnaround penalty (RD/WR batching technique). Therefore, to fairly calculate the overall transmission penalty, we distinguish how many RD/WR commands are executed together. This is done by multiplying PGNCL by a proportion obtained using the Command Batch Size (CBS). CBS is the amount of RDs or WRs that can be executed before switching from one batch to another. The worst case is estimated with Equation 5.

$$CBS(\tau_i) = min\Big(\sum_{\forall b \in B} min\Big(\frac{\sum_{B_k=b} \mathcal{A}_k(C_k)}{\sum_{B_k=B_i} \mathcal{A}_k(C_k)}, 1\Big), MS\Big) \quad (5)$$

where MS is the Maximum Size set in the RD/WR threshold register. CBS considers the amount of commands in each used bank, which are relativized to the task under analysis bank. CBS is limited by MS as we can not exceed the configured batch threshold[3]. The total Data Transmission Cost (DTC) is expressed in Equation 6, where X is *intra* or *inter*.

$$\begin{aligned} DTC^X(\tau_i) = &\frac{1}{CBS} * PGNCL^X(\tau_i) * \\ &\Big(\bar{SP}^X * WT^X + \bar{LP}^X * RT^X\Big) + \\ &(1 - \frac{1}{CBS}) * PGNCL^X(\tau_i) * \frac{BL}{2} \end{aligned} \quad (6)$$

The equation distinguishes between (1) the turnaround and (2) the consecutive data transmission. To calculate (1), it is necessary to know how many turnarounds are produced through the tasks execution ($\frac{1}{CBS} * PGNCL^X(\tau_i)$). Then, multiply the resultant value by the sum of the writes and reads transmission cost ($\bar{SP}^X * WT^X + \bar{LP}^X * RT^X$). $\bar{SP}^X$ and $\bar{LP}^X$ are the average intra/inter store and load proportions of $\mathcal{T}$ respectively, i.e., $\frac{1}{N}\sum_{\tau_k \in \mathcal{T}|X} SP_i$ and $1 - \bar{SP}^X$. To calculate (2), we multiply the number of consecutive reads and writes by its burst length ($(1 - \frac{1}{CBS}) * PGNCL^X(\tau_i) * \frac{BL}{2}$).

5) Row Switch Cost: Now we include the cost coming from the row buffer management of the banks. The Row Switch Cost (RSC) is the time delay suffered by τ_i when the row in a row buffer is replaced by another. Equation 7 is used when a row switch is produced in the same bank as τ_i. Otherwise, Equation 8 is applied.

RSC^{intra} is the sum of four addends: (1) the maximum between the RD to PRE and WR to PRE transition cost ($max(t_{RTP}, t_{WR})$), (2) PRE execution cost (t_{RP}), (3) ACT execution cost (t_{RCD}) and (4) the maximum between CL and WL ($max(CL, WL)$). Figures 7 and 8 show the previous timings and transitions. In (1) we get the maximum because we can't be sure of the last command type (RD or WR) before the execution of PRE. This term complies with **Constraint C4**. Other two important timing violation constraints to verify are **Constraint C2** and **Constraint C3**. These are satisfied as $max(t_{RTP}, t_{WR}) + t_{RCD} + max(CL, WL) + \frac{BL}{2} > t_{RAS}$ and the previous plus t_{RP} is $> t_{RC}$.

RSC^{inter} introduces: (1) a single DDR3 cycle delay for the PRE command and (2) either t_{RRD} or $t_{FAW} - 3 * t_{RRD}$ cycles depending on the situation. In order to satisfy **Constraint C1** (see Figure 9), when 4 or less banks are used, t_{RRD} is applied. Otherwise, $t_{FAW} - 3 * t_{RRD}$ is used.

$$\begin{aligned} RSC^{intra} = &max(t_{RTP}, t_{WR}) + t_{RP} + \\ &t_{RCD} + max(CL, WL) \end{aligned} \quad (7)$$

$$RSC^{inter} = \begin{cases} 1 + t_{RRD} & \text{if } NB \leq 4 \\ 1 + t_{FAW} - 3 * t_{RRD} & \text{otherwise} \end{cases} \quad (8)$$

where NB is the number of banks used.

6) Number of Row Switches: For the intra-bank interference, we start by assuming that for every interfering access command that took place during the execution of τ_i (i.e., Equation 2), the latter suffers a row switch. This value can't exceed $\mathcal{A}_i(C_i)$ as the number of actives can't be higher than the number of accesses. To consider the open-row policy, $\mathcal{A}_i(C_i)$ and $\mathcal{A}_j(C_j * RIE(\tau_i, \tau_j))$ are divided by $ACOR_i$ and $ACOR_j$ respectively.

[3]TI refers to the write and read batches as SDRAM read/write bursts. Register RWTHRESH is used for setting the burst size, where its field WR_THRSH is for the writes and RD_THRSH for the reads. Both fields accept values from 0 to 31 (see [5], [7]).

In the case of inter-bank interference, the number of row switches affecting τ_i is calculated per external bank. For each bank interference calculation we can differentiate two parts. The first applies only when there are no intra-bank interference for the given B_j, i.e., a single τ_j accesses the bank ($tsb(B_j) = 1$). Therefore, just those switches carried out by τ_j in isolation (S_j) are considered. The second part considers the forced switches caused by τ_j when intra-bank interference takes place ($tsb(B_j) > 1$). It is calculated dividing $\mathcal{A}_j(C_j * RIE(\tau_i, \tau_j))$ by its $ACOR_j$. We assume that the maximum interfering row switches per bank is limited by the number of row switches of τ_i.

These quantities are calculated with the Number of Row Switches (NRS) function (Equations 9 and 10).

$$NRS^{intra}(\tau_i) = \sum_{\substack{\tau_j, j \neq i, \\ B_j = B_i}} min\left(\frac{A_i(C_i)}{ACOR_i}, \frac{\mathcal{A}_j(C_j * RIE(\tau_i, \tau_j))}{ACOR_j}\right) \tag{9}$$

$$NRS^{inter}(\tau_i) = \sum_{b=0, b \neq B_i}^{NB-1} min\left(\frac{A_i(C_i)}{ACOR_i}, \sum_{\substack{\tau_j, j \neq i \\ tsb(B_j)=1}} S_j + \sum_{\substack{\tau_j, j \neq i, B_j = b \\ tsb(B_j)>1}} \frac{\mathcal{A}_j(C_j * RIE(\tau_i, \tau_j))}{ACOR_j}\right) \tag{10}$$

7) Total Row Switch Cost: The Total Row Switch Cost (TRSC) (Equation 11) calculates the total impact the interfering row switches have on τ_i. This is done by multiplying the number of switches calculated by Equation 9 and 10 by their respective switching cost returned by Equations 7 and 8.

$$TRSC^X(\tau_i) = NRS^X(\tau_i) * RSC^X \tag{11}$$

8) Interference Cost Function: Finally, the global Interference Cost (IC) expression is shown in Equation 12. It is the sum of the data transmission and the row switch cost for both cases, the intra-bank and inter-bank interference.

$$\begin{aligned} IC(\tau_i) = & IC(\tau_i)^{intra} + IC(\tau_i)^{inter} = \\ & TRSC^{intra}(\tau_i) + DTC^{intra}(\tau_i) + \\ & TRSC^{inter}(\tau_i) + DTC^{inter}(\tau_i) \end{aligned} \tag{12}$$

Algorithm 1 repeatedly calls IC(τ_i) until the interference value converges. Experimentation has shown that, after 4 or 5 iterations, the interference variation was not significant. This is necessary as explained in Section V-B1. An array with the IC(τ_i) values for all tasks is returned.

Algorithm 1: recursive_IC_calculation

Input: $\mathcal{T}$ (input task set)
Output:
– $IC[\mathcal{T}]$ (interf. cost for each task in $\mathcal{T}$)
– B (true iff the algorithm succeeds)
Local Variables:
– n (the recursion index)
– $IC^n[\mathcal{T}]$ (interference cost at step n)

```
/* Initialization                                   */
n=0
∀τ ∈ T, IC^n[τ] = 0
/* Recursive loop                                   */
while true do
    for each τ_i, τ_j ∈ T, τ_j ≠ τ_i, compute RIE^{n+1}(τ_i, τ_j) by Eq. 2 using IC^n
    for each τ_i ∈ T, compute IC^{n+1}(τ_i) by Eq. 3 to 12 using RIE^{n+1}
    if ∀τ_i ∈ T, IC^{n+1}(τ_i) == IC^n(τ_i) then
        return (IC^n, true)
    if ∃τ_i ∈ T such that C_i + IC^{n+1}(τ_i) > T_i then
        return (IC^n, false)
    n = n + 1
```

VI. Experimentation

A. Measurement framework

The measurement framework plays an important role in this work. We need it for: (1) retrieving the properties C_i, $\mathcal{A}_i$, SP_i, LP_i, S_i and $ACOR_i$ of the task (see Section V-A), (2) analyzing the behavior of the DDR device and controller, and (3) retrieve the measured WCET for the test scenarios. The procedure followed for obtaining the tasks metrics vary according to core type:

- **ARMv7 Performance Counters**: The ARM Cortex A15 cores use performance counters to monitor different events from the core perspective. This ARM core has a dedicated cycle execution counter and six general purpose counters for which we can choose an event (see events in [1]). We access these by using a Start-Stop pattern.
- **C66x DSP Time Stamp Counter**: The C66x DSPs can record the execution time of a task by reading a 64bit time stamp register. For each read to the 32 LSBs of the register, a copy of the 32 MSBs is automatically performed by hardware to avoid time inconsistencies. The time stamp register is accessed using a Start-Read pattern.
- **DDR3 Memory Controller Performance Counters**: The DDR3 MC is able to monitor SDRAM events. This is done through one dedicated cycle execution performance counter and two general purpose counters for which we can choose the target event (e.g., accesses, reads) and filter by master (e.g., Cortex A15, C66x, system). The counters are accessed using a Start-Read pattern.

B. Experimentation Results

Equation 12 is tested by doing a direct comparison between theoretical and measured outputs. To do so, the following expression is used: $Output(\tau_i) = 1 + \frac{\alpha * IC_i}{C_i}$. The interference impact IC_i for τ_i is retrieved from Algorithm 1 and is multiplied by the core-controller frequency conversion factor α (1.5 and 1.87 for our Keystone and Sitara configuration) and normalized with respect to τ_i. The result is added to the normalized τ_i itself, i.e., plus one.

In terms of execution time, our Python implementation of Algorithm 1 (running on an Intel Core i7-8750H CPU @2.20GHz) takes less than 3ms to complete for an 8 interfering cores test case.

The test scenarios used for evaluating the cost function equation makes use of a set of benchmark tasks which play the role of real tasks. We distinguish two types of benchmarks tasks: synthetic and real application tasks. The real application tasks are adaptations from the ROSACE case study tasks [11]. To carry out the cost function equation evaluation each active core runs either a synthetic or a real task. Different cores can execute different instances of the same type of task. Table II lists the tasks and their main characteristics. Several scenarios made up of different task-core, core-DDR3 bank and task period combinations are used, and have been divided in two parts. The first one is for monoperiodic conditions, i.e., all tasks execute according to a unique period, and the second for biperiodic conditions, i.e., two periods are used. The graphs depict the normalized execution time (Y axis) as function of interfering cores (X axis), which accumulate. The graphs show the measured execution time in yellow, the cost function output in blue and the cost function output without considering the MC RO in green.

TABLE II: Tasks main properties on Keystone II

Task	Description	C	$\mathcal{A}$(C)	SP	S	ACOR
sb0 (ARM)	Synthetic benchmark. Vectors operations.	36315	437	0.31	0	2.46
sb0 (DSP)		33202	443	0.30	0	1.70
rb0 (ARM)	Real benchmark. ROSACE engine management.	3107	51	0.63	16	3.40
rb0 (DSP)		12698	197	0.52	16	2.20
rb1 (ARM)	Real benchmark. ROSACE altitude filtering.	5110	84	0.63	14	3.36
rb1 (DSP)		18927	304	0.51	23	2.10
rb2 (ARM)	Real benchmark. ROSACE elevator management.	6229	96	0.67	25	3.31
rb2 (DSP)		21708	356	0.52	32	2.21

1) Monoperiodic: The tests carried out consider a unique period T1 of 900μs. Figures 11, 12 and 13 show intra-bank interference, i.e., a single bank, and the same type of task (sb0). The results show that the measured WCET are correctly upper-bounded by the theoretical computation and, most importantly, the increasing tendency as function of the interference cores addition is well described. It is interesting to see the final value difference depending on the core type executing τ_i (sb0). For the ARM case (Figure 11) we end up having 3.37 and 5.22 units while for the DSP case (Figure 13) we find 4.17 and 6.79 units of measured and theoretical cost respectively. The core difference impact is well captured by the DDR3 interference equation. However, note that the theoretical value is a bit far from the worst-case measured value, which is normal taking into account the pessimistic assumptions made in Equation 12, e.g., no batching for intra-bank interference where some may be found, always assuming the more time consuming WR to PRE transition (RD to PRE can also occur). The theoretical value with no MC RO is even more pessimistic as the open-row policy optimization is not considered, obtaining 8.9 and 9.68 units for the ARM 0 and DSP 0 respectively. Figure 12 shows the total number

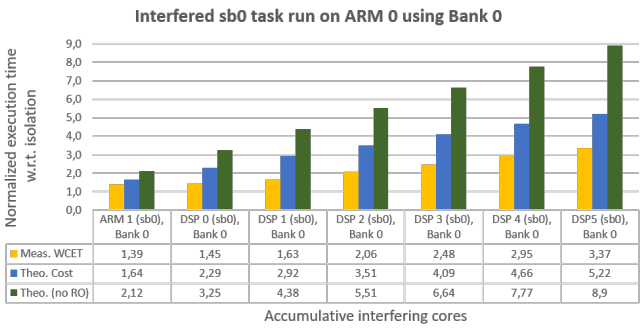


Fig. 11: Execution time of sb0 on ARM 0 as function of other cores. Intra-bank interference. Keystone II.

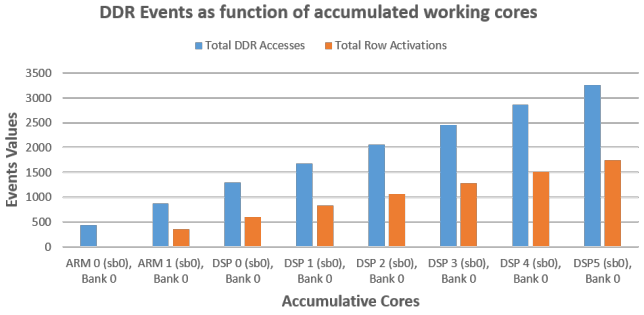


Fig. 12: Total SDRAM accesses and row activations evolution as function of cores. Intra-bank interference. Keystone II.

of accesses and row switches evolution for the scenario in Figure 11 from a system point of view. As only a single bank is used, the number of ACTs increases every time a core is activated due to the compulsory row switches. Note that when only ARM 0 is running, no row switches are produced because it works within the same DDR memory row the whole time (see that S value is zero in Table II). The figure reports an average of 2.08 access/actives. This figure is useful for understanding the open-row policy advantage and the importance of considering it by Equations 9 and 10 by using Equation 1.

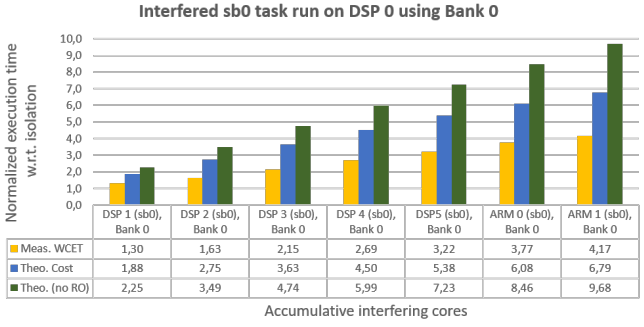


Fig. 13: Execution time of sb0 on DSP 0 as function of other cores. Intra-bank interference. Keystone II.

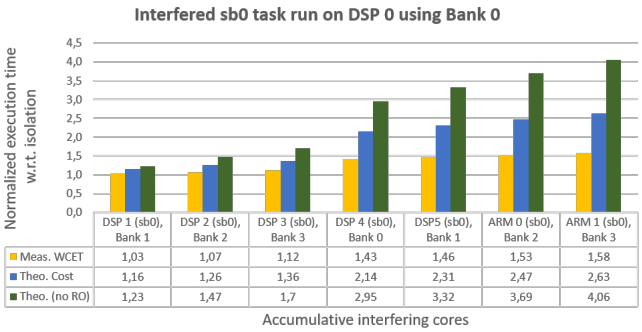


Fig. 14: Execution time of sb0 on ARM 0 as function of other cores. Intra-bank and inter-bank interference. Keystone II.

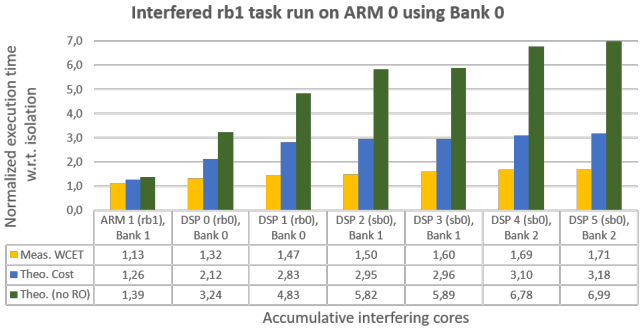


Fig. 15: Execution time of sb0 on DSP 0 as function of other cores. Intra-bank and inter-bank interference. Keystone II.

Figure 14 keeps the same type of task for all cores but changes the memory configuration to one where several banks are used. Intra-bank and inter-bank interference are found. The former is produced by DSP 4, which causes the notable interference rise from DSP 3 to DSP 4. The latter can be seen when adding DSP 1, DSP 2, DSP 3, DSP 5, ARM 0 and ARM 1 which has less impact. Overall, the measured WCET is well upper-bounded and the tendency well described. However, as previously seen for the intra-bank interference, the inter-bank interference calculation also overestimates. This is partially due to the assumption that every row active sequence (PRE + ACT) of the studied task always suffers an external bank row switch (Equations 8 and 10). This is very unlikely to happen as all the used banks would need to precharge and activate a new row almost at the same time repeatedly during the whole execution (see Figure 9).

A scenario where different type of tasks are used is shown in Figure 15. Intra-bank interference is produced by DSP 0 and DSP 1. The other cores introduce inter-bank interference. τ_i is running on an ARM resulting in a short execution time. This makes τ_i be interfered by fragments of the other tasks with longer execution times, except for ARM 1 which runs the same task type.

The previous results were all taken from the Keystone II SoC. Figures 16 and 17 show the behavior of the interference

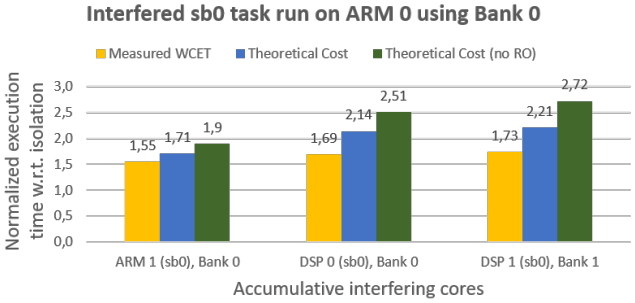


Fig. 16: Execution time of sb0 on DSP 0 as function of other cores. Intra-bank and inter-bank interference. Sitara AM5728.

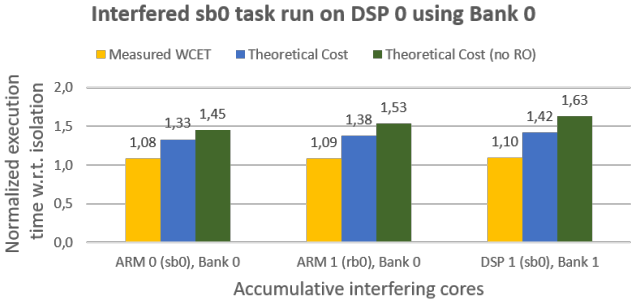


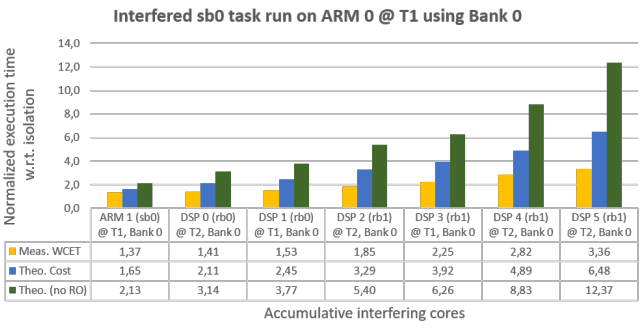
Fig. 17: Execution time of rb1 on ARM 0 as function of other cores. Intra-bank and inter-bank interference. Sitara AM5728.

on the Sitara AM5728 SoC. We remark again the impact difference due to the intra-bank interference of the two first core additions and the inter-bank interference of the last added core. Besides, we point out the fact that the DSPs on Sitara AM5728 has less interfering capacity due to a lower operation frequency compared to the ARM cores.

2) Biperiodic: The tests for the biperiodic cases consider a fixed period *T1* of 900μs and a variable *T2*. The aim of the *T2* variation is to show the importance of the period of a task in terms of interference as depicted in Figures 10a and 10b.

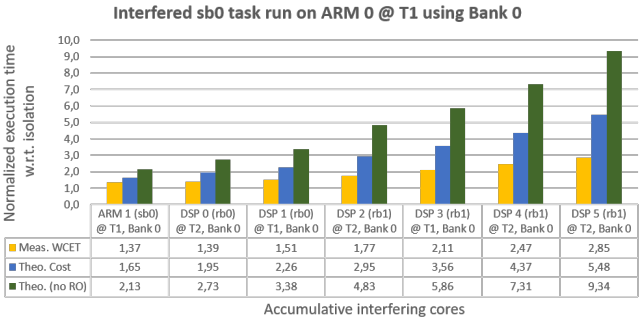
Figures 18 and 19 show the interference cost that is caused by changing the period *T2* value from 60μs to 90μs respectively for the tagged tasks. Both scenarios use the same task mapping and memory configuration (a single bank). The increase of period *T2* reduces the amount of time *T2* tasks interfere *T1* tasks. For example, this can be clearly appreciated looking at the values when enabling DSP 4 and DSP 5, where the measured values for $T2 = 60\mu s$ are 2.82 and 3.36 (equivalent to 102408 and 122018 cycles) and for $T2 = 90\mu s$ are 2.47 and 2.85 respectively (89698 and 103497 cycles). This reduction in interference is well captured, having a theoretical impact of 4.89 and 6.48 for $T2 = 60\mu s$ (177580 and 235321 cycles) and 4.37 and 5.48 for $T2 = 90\mu s$ (158697 and 199006 cycles).

Figure 20 shows the scenario where different banks are used. The period *T2* is 30μs. However, *T2* is not low enough to make the corresponding τ_j re-execution have an effect on



	ARM 1 (sb0) @ T1, Bank 0	DSP 0 (rb0) @ T2, Bank 0	DSP 1 (rb0) @ T1, Bank 0	DSP 2 (rb1) @ T2, Bank 0	DSP 3 (rb1) @ T1, Bank 0	DSP 4 (rb1) @ T2, Bank 0	DSP 5 (rb1) @ T2, Bank 0
Meas. WCET	1,37	1,41	1,53	1,85	2,25	2,82	3,36
Theo. Cost	1,65	2,11	2,45	3,29	3,92	4,89	6,48
Theo. (no RO)	2,13	3,14	3,77	5,40	6,26	8,83	12,37

Fig. 18: Execution time of sb0 on ARM 0 as function of other cores (T1 = 900μs, T2 = 60μs). Keystone II.



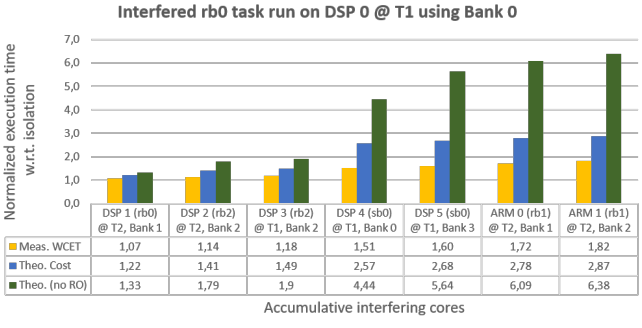
	ARM 1 (sb0) @ T1, Bank 0	DSP 0 (rb0) @ T2, Bank 0	DSP 1 (rb0) @ T1, Bank 0	DSP 2 (rb1) @ T2, Bank 0	DSP 3 (rb1) @ T1, Bank 0	DSP 4 (rb1) @ T2, Bank 0	DSP 5 (rb1) @ T2, Bank 0
Meas. WCET	1,37	1,39	1,51	1,77	2,11	2,47	2,85
Theo. Cost	1,65	1,95	2,26	2,95	3,56	4,37	5,48
Theo. (no RO)	2,13	2,73	3,38	4,83	5,86	7,31	9,34

Fig. 19: Execution time of sb0 on ARM 0 as function of other cores (T1 = 900μs, T2 = 90μs). Keystone II.

τ_i as the interference overhead suffered by τ_i is not making its execution time exceed *T2*. The highest impact is produced by DSP 4 which is accessing the same bank as τ_i.

VII. Conclusions and Future Work

In this work, we show the development and evaluation of a DDR SDRAM interference cost function equation. The results show that the cost function always yields values above the measured WCET for different monoperiodic and biperiodic scenarios. As well, it correctly describes the behavior of the SDRAM interference when adding the interfering core.



	DSP 1 (rb0) @ T2, Bank 1	DSP 2 (rb2) @ T2, Bank 2	DSP 3 (rb2) @ T1, Bank 2	DSP 4 (sb0) @ T1, Bank 0	DSP 5 (sb0) @ T1, Bank 3	ARM 0 (rb1) @ T2, Bank 1	ARM 1 (rb1) @ T2, Bank 2
Meas. WCET	1,07	1,14	1,18	1,51	1,60	1,72	1,82
Theo. Cost	1,22	1,41	1,49	2,57	2,68	2,78	2,87
Theo. (no RO)	1,33	1,79	1,9	4,44	5,64	6,09	6,38

Fig. 20: Execution time of sb0 on ARM 0 as function of other cores (T1 = 900μs, T2 = 30μs). Keystone II.

Besides, we can appreciate the overpessimistic theoretical results when not considering the MC reordering optimization. Indeed, the key is to make the cost function be able to model the DDR device (e.g., inter and intra bank interference timing difference, RD/WR distinction), the MC (e.g., open-row policy, command burst), the heterogeneity of the platform (e.g., ARM Cortex A15 and C66x DSP distinction) and the task properties (e.g., measured WCET in isolation, SDRAM accesses).

As future work, we plan to test this cost function equation while having the data caches enabled. This would allow us to reduce the DDR3 interference further in exchange of L2 shared cache interference. A priori, if private caches are assumed, the procedure would be the same, just requiring the tasks profiling with the cache memories enabled. However, if the caches are shared, inter-core interference in form of evictions could take place. In this case, cache locking or cache partitioning should be applied to avoid modelling this kind of interference.

Acknowledgment

This work was supported by the Defense Innovation Agency (AID) of the French Ministry of Defense (research project CONCORDE N° 2019 65 0090004707501) and the French Civil Aviation Authority (DGAC) (research project PHY-LOG).

References

[1] ARM. *Cortex A15 MPCore Technical Reference Manual*, March 2012.

[2] Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio C. Buttazzo. A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2020, Sydney, Australia, April 21-24, 2020*, pages 239–252. IEEE, 2020.

[3] Mohamed Hassan and Rodolfo Pellizzoni. Bounding dram interference in cots heterogeneous mpsocs for mixed criticality systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2323–2336, 2018.

[4] Mohamed Hassan and Rodolfo Pellizzoni. Analysis of Memory-Contention in Heterogeneous COTS MPSoCs. In *32nd Euromicro Conference on Real-Time Systems*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:24, 2020.

[5] Texas Instruments. *Keystone II Architecture DDR3 Memory Controller User's Guide*, March 2015.

[6] Texas Instruments. *66AK2Hxx Multicore DSP+ARM® KeyStone™ II System-on-Chip (SoC)*, October 2017.

[7] Texas Instruments. *AM572x Technical Reference Manual*, August 2019.

[8] JEDEC ASSOCIATION. *DDR3 SDRAM. JESD79-3C*, 2008.

[9] Hyoseung Kim, Dionisio de Niz, Björn Andersson, Mark H. Klein, Onur Mutlu, and Ragunathan Raj Rajkumar. Bounding and reducing memory interference in cots-based multi-core systems. *Real-Time Systems*, 2016.

[10] Alfonso Mascareñas González, Frédéric Boniol, Youcef Bouchebaba, Jean-Loup Bussenot, and Jean-Baptiste Chaudron. *Heterogeneous Multicore SDRAM Interference Analysis*, page 12–23. RTNS'2021. 2021.

[11] Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The rosace case study: From simulink specification to multi/many-core execution. In *IEEE 19th Real-Time and Embedded Technology and Applications Symposium*, pages 309–318, 2014.

[12] Zheng Wu, Rodolfo Pellizzoni, and Danlu Guo. A composable worst case latency analysis for multi-rank dram devices under open row policy. *Real-Time Systems*, 52, 11 2016.

[13] Heechul Yun, Rodolfo Pellizzon, and Prathap Kumar Valsan. Parallelism-aware memory interference delay analysis for cots multicore systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 184–195, 2015.