DDR SDRAM cost function equation development for heterogeneous MPSoCs

Technical Report: CONCORDE_WP2_T2_REPORT1
Release date: 03/05/2022
Revision date: 03/05/2022
Version: 1.00

Alfonso Mascareñas González ✉
ISAE-SUPAERO, Université de Toulouse, France

Jean-Baptiste Chaudron ✉
ISAE-SUPAERO, Université de Toulouse, France

Frédéric Boniol ✉
ONERA, Université de Toulouse, France

Youcef Bouchebaba ✉
ONERA, Université de Toulouse, France

Jean-Loup Bussenot ✉
ONERA, Université de Toulouse, France

1 Introduction

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. Having the population-based heuristic algorithms in mind, this cost function must be time efficient and offer a correct description of the behavior of the SDRAM interference cost. For such a purpose, we consider the DDR SDRAM device operations and timings, the memory controller reordering 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 what is time consuming. The cost function evaluation is measurement-based using the heterogeneous multicore SoC Keystone II by Texas Instruments. The Keystone II implementation on Code Composer Studio, the DDR memory cost function equations on Jupyter Notebook and the put into practice of the previous in a task/memory mapping tool for heterogeneous platforms also on Jupyter Notebook, can be found in this repository[1]. The remaining sections of this technical report are structured in the following way. Section 2 introduces the theoretical basis for the comprehension of the equations development for the cost function. Section 3 lists all the important facts to have in mind for the experimental part of the report. Section 4 explains the development of the cost function equation. Section 5 compares the theoretical and the measured values on the Keystone II. Finally, Section 6 summarizes the entire work.

2 DDR SDRAM Background

DDR3 SDRAM devices must comply with the JEDEC standard JESD79-3E [7] which defines the functionalities, electrical characteristics, packaging, etc. This memory relies on three

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

busses in order to properly function: (1) the command bus which is used for transmitting the commands to the banks, (2) the address bus which is used for selecting the memory location to be accessed and (3) the data bus which is used for transferring the data.

2.1 DDR3 SDRAM Device Addressing

A convenient way to think about DDR SDRAM addressing is to see this memory as a set of logic units called 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 [3]).

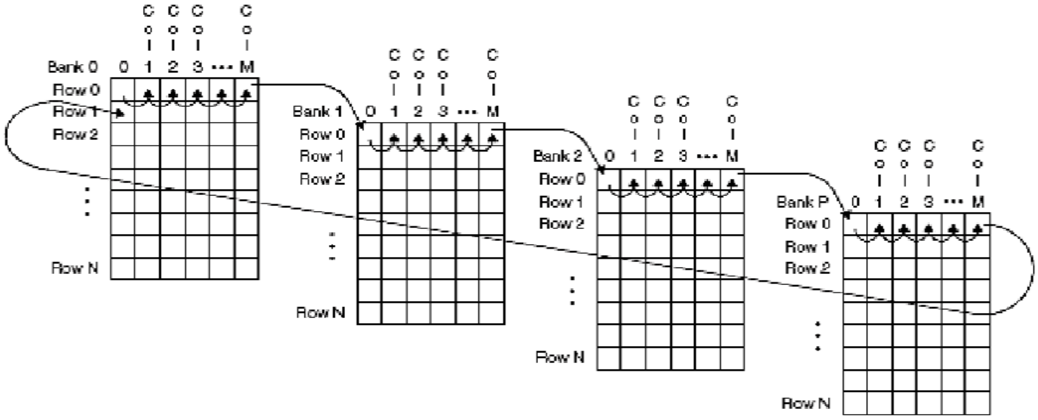


Figure 1 DDR memory organization: Columns, Rows and Banks. Source: [3]

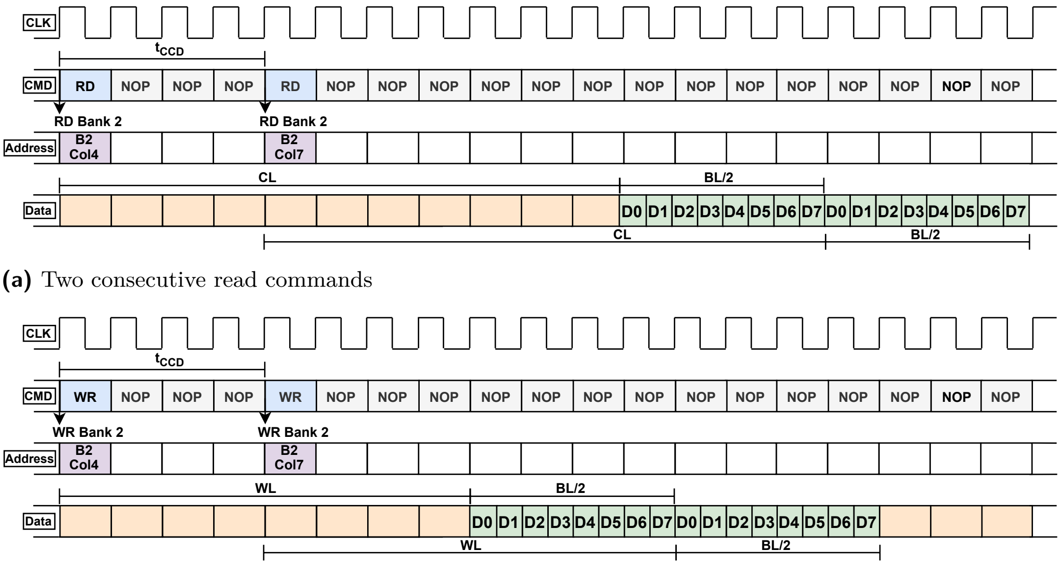
2.2 Device Operations

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 2.1) 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 command is performed. Associated to the command executions there are specific transition timings (see Table 1).

Note *CAS Latency* (CL) and the *CAS Write Latency* (WL) for the RD and WR operations respectively. These latencies define the particular delays between the command execution and the moment data is available on the data bus (see Figures 2a and 2b).

Table 1 Keystone II DDR3 parameters - 800MHz Controller

Feature	**Value**	**Description**
Number of Ranks	1 rank	Number of available SDRAM ranks
Number of Banks	8 banks	Number of internal SDRAM bank
Page Size	10 columns	1024-word page
SDRAM Width	64 bits	64-bit bus width
Burst Length (BL)	8 columns	Number of columns that can be accessed during a read or a write
CAS to CAS command delay (t_{CCD})	4 cycles	Minimum time to leave between two CAS commands
CAS Latency (CL)	11 cycles	Elapsed time since the execution of a read command and the arrival of data
CAS Write Latency (WL)	8 cycles	Elapsed time since the execution of a write command and the arrival of data
Write Recovery Time (t_{WR})	12 cycles	Time from the last burst write cycle until a precharge can be issued to the same bank
Write to Read (t_{WTR})	5 cycles	Write to read turnaround time
Precharge (t_{RP})	11 cycles	Deactivation time of an opened row of a bank
Read to Precharge (t_{RTP})	6 cycles	Delay between a read command execution and a row bank precharge
Active (t_{RCD})	11 cycles	Activation time of a closed row of a bank
Active to Precharge (t_{RAS})	28 cycles	Minimum time from an active to a precharge command within the same bank
Inter-bank Active to Active (t_{RRD})	6 cycles	Minimum time separation of two consecutive active commands to different banks
Four Active Windows (t_{tFAW})	24 cycles	Time window where maximum four consecutive active commands can be issued
Intra-bank Active to Active (t_{RC})	39 cycles	Minimum time between two active command to the same bank

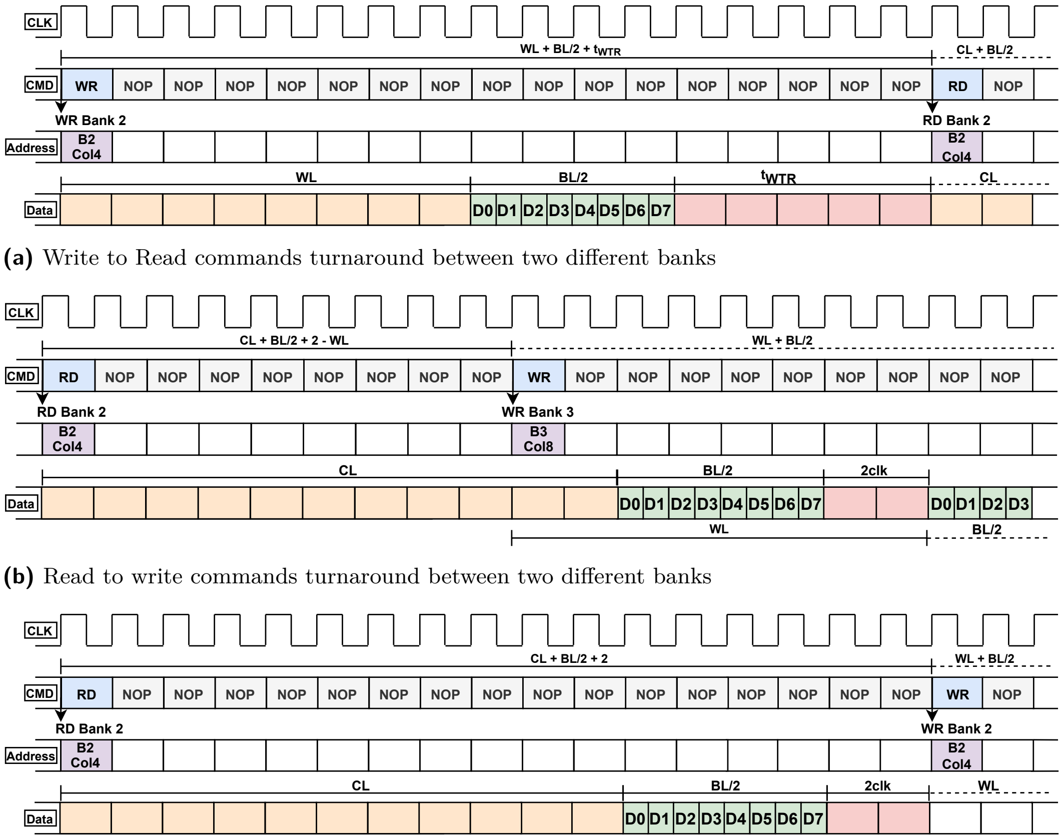

CLK
t_{CCD}
CMD
RD NOP NOP NOP RD NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP
RD Bank 2
RD Bank 2
Address
B2 Col4
B2 Col7
CL
BL/2
Data
D0 D1 D2 D3 D4 D5 D6 D7 D0 D1 D2 D3 D4 D5 D6 D7
CL
BL/2
CLK
t_{CCD}
CMD
WR NOP NOP NOP WR NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP
WR Bank 2
WR Bank 2
Address
B2 Col4
B2 Col7
WL
BL/2
Data
D0 D1 D2 D3 D4 D5 D6 D7 D0 D1 D2 D3 D4 D5 D6 D7
WL
BL/2

(a) Two consecutive read commands

(b) Two consecutive write commands

Figure 2 Consecutive read and write sequence diagrams

Another important delay to consider is the data bus turnaround time caused by RD to WR or WR to RD state transitions. Figure 3a depicts the timing transitions for the WR to RD case when using two different banks. Figures 3c and 3b show the transitions from a read to a write in the same and different banks.



(a) Write to Read commands turnaround between two different banks

(b) Read to write commands turnaround between two different banks

(c) Read to write commands turnaround in the same bank

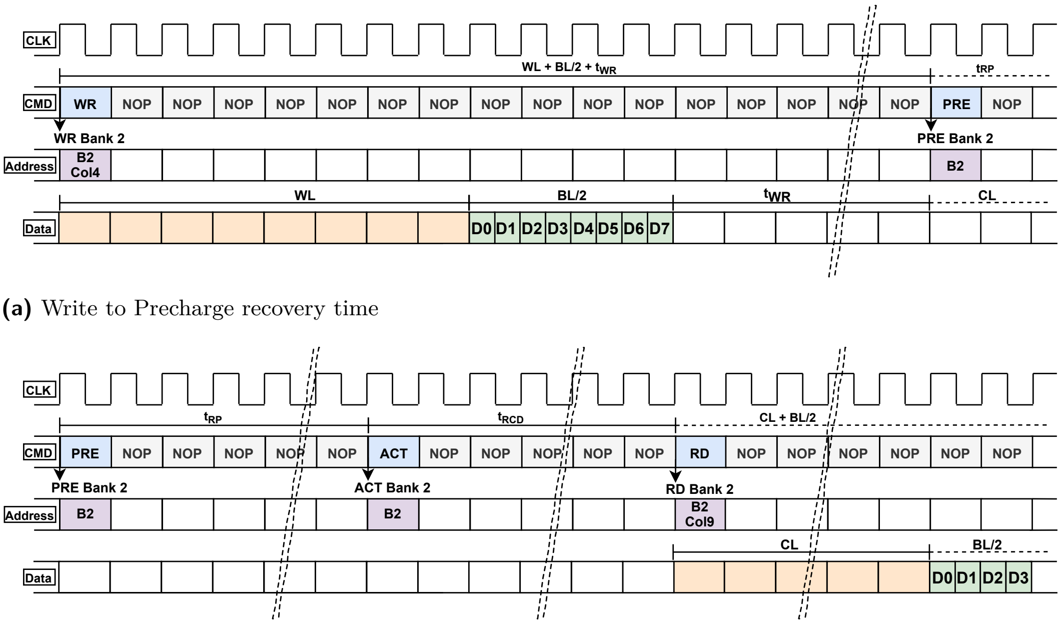
Figure 3 Read/Write commands turnaround sequence diagrams

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 4a and 4b show the previous transitions respectively. The dashed lines in both figures represent a time skip. The memory controller and the DDR device itself pack commands optimally by considering their latencies and the accessed banks in order to maximize the data output (see Figures 2a to 5).

2.2.1 Timing Constraints

Apart from the previous timings related to the execution of commands or transitions, there are timing constraints to respect in order to achieve a well-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 5 describes this constraint.



(a) Write to Precharge recovery time

(b) Precharge to Read commands transition time

Figure 4 Precharge and Active commands sequence diagrams

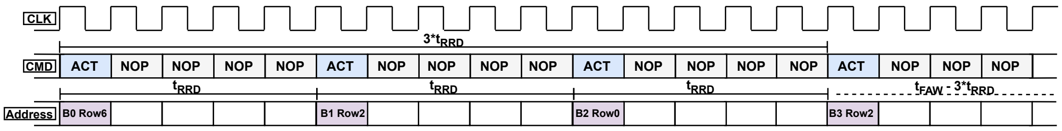


Figure 5 Multiple ACT commands execution behavior

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 $RD \rightarrow PRE$ spacing is given by t_{RTP}. Likewise, the $WR \rightarrow PRE$ minimum spacing time is t_{WR}. Figure 4b shows how this constraints is respected by waiting the write recovery time t_{WR} before performing a PRE.

C5 **Refresh interval**: Eight refresh commands within a time window of size $8 * t_{REFI}$ must be issued.

2.3 Memory Controller Arbitration

In this work, the used DDR3 memory controller 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 [3]:

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 come, first served 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 command) and the activation of the new row (ACT command) 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 [9]. 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 DDR3 controllers are: RD, WR, ACT and PRE. RD and WR commands have the same priority after being reordered according to the arbitration rules.

3 Considerations

Along the development of this work, the following points always apply:

- **Platform**: The heterogeneous platform considered for carrying out measurement-based evaluations of the cost functions is the **Keystone II** model TCI6636K2H by Texas Instruments [4]. This SoC is made up of 4 ARM Cortex A15 cores [1] and 8 C66x DSPs [2]. However, in this work, we evaluate the results with 2 ARM cores and 6 DSPs, i.e., a total of 8 cores in parallel.
- **DDR SDRAM**: The memory used is a DDR3 SDRAM version DDR3-1600K. It has a single rank and 8 banks. During the evaluation, 4 banks are used. The main properties of this memory are found in Table 1.
- **SDRAM Refresh**: SDRAM refreshing effects are not considered.
- **No data caches**: We disable the L1D and L2 cache to increase the task's access frequency 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 effect, e.g., preemption, frequency throttling (see [6]).
- **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 core is temporarily increased. DDR3 SDRAM command priority raise counter (anti-starvation mechanism) is left at default setting.
- **Periodicity**: Tasks are periodically executed using a non-preemptive Start-Finish-Idle pattern in a predefined order.
- **FPUs**: Floating Point Units are disabled.

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

4 Formal Description

4.1 Tasks Model

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

$$\tau_i := (C_i, A_i, SP_i, S_i, ACOR_i, PE_i, B_i, T_i)$$

where:

- C_i: The WCET in isolation. $C_i \,\epsilon\, \mathbb{N}^+$.
- A_i: The number of DDR3 memory accesses. $A_i \,\epsilon\, \mathbb{N}^+$.
- SP_i: The *Store Proportion* (SP) indicates the share of stores in A_i. $SP_i = 1 - LP_i$ where SP_i and $LP_i \,\epsilon\, [0, 1]$. LP_i is the *Load Proportion* of A_i.
- 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 detailed in Section 4.1.1.
- 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 the task set. To denote the total number of tasks sharing a specific bank b, we define $tsb(b) = |\{\tau_i \epsilon \mathcal{T} | B_i = b\}|$.

4.1.1 ACOR

The task's internal composition 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 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 $ACOR_i$ for each task. An over-pessimistic solution would be to assume a close-row policy ($ACOR_i = 1$), as the row switch reduction is not considered. Another 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 ($S_{i_{||}}$). Note that as we are working with a heterogeneous platform, the saturating benchmark should be executed in all the core types available (e.g., ARM Cortex A15, DSP C66) and pick the worst-case result. $S_{i_{||}}$ is obtained using the DDR3 controller counters. Thus, the switches imposed by the interfering task are obtained. The τ_i DDR 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. Equation 1 describes the previous:

$$ACOR_i = \frac{A_i}{S_{i_{||}}} \tag{1}$$

The idea behind Expression 1 may be better understood through an example. Listing 1 shows a fragment of benchmark *sb0* (see Table 2). In terms of execution, we can distinguish some 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 (see ACOR value for *sb0* in Table 2). This happens due to the open-row policy of the controller, that for efficiency purposes, executes those groups commands together. The difference between both methods is very small. Hence, we consider equivalent to use the measurement-based result.

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
```

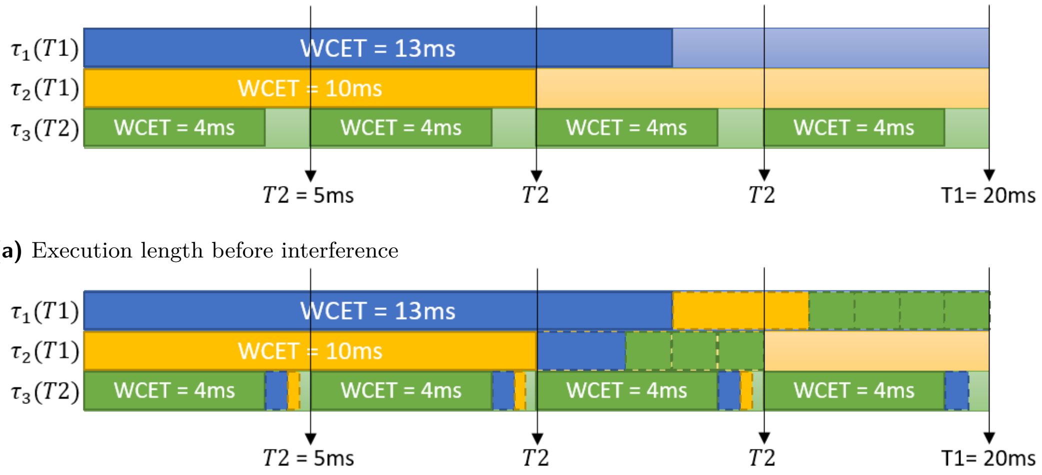
4.2 DDR3 Memory Interference Cost Function

This cost function design relies on four aspects: (1) **DDR3 SDRAM device**, (2) **DDR3 Memory Controller**, (3) **Platform Architecture** and (4) **Application**. These parts must be considered by the cost function. As a result of (1), we permanently differentiate between the interference originated from inside the bank (intra-bank interference) and outside it (inter-bank interference).

4.2.1 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 make use of the interfering DDR3 memory accesses and calculate the proportion of time in which a task is interfered by another. In order to consider the tasks period differences, it is computed how many 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 6a and 6b 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 problem, we consider a recursive implementation described by Algorithm 1. The Relative Interference Exposition (RIE) shown in Equation 2 calculates the time length τ_i is exposed to τ_j. The function recursively calls the DDR3 interference cost



(a) Execution length before interference

(b) Execution length after interference

Figure 6 Biperiodical DDR3 interference example

function IC (Equation 11) 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 memory controller cycles and C_i in core cycles. Then, the former has to be converted to the later by multiplying by a factor of $\alpha = freq_{core}/freq_{DDR}$ (i.e., the bus clock multiplier).

4.2.2 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, it is assumed that the DDR3 accesses are uniformly distributed along the tasks execution. By using this equation, we make the assumption that all the interfering accesses affect the task under analysis.

$$PGNCL^X(\tau_i) = \sum_{\tau_j, j\neq i} A_j * RIE(\tau_i,\tau_j) \tag{3}$$

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.

4.2.3 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 $t_{CCD} = \frac{BL}{2}$. The DDR3 memory is designed in such a way that there is not idle time between two consecutive WRs or RDs (see Figures 2a and 2b). Nevertheless, moving from one RD to a WR and vice-versa entails a penalty which vary according to the previous command (RD/WR) and in which bank it takes place (intra/inter). This is the turnaround time penalty (see Figures 3a, 3b and 3c). 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 1. We don't consider the effect of ranks as only one is present in the platforms of this work. Please refer to [8, 11] for more information on the effect of ranks on turnaround transitions and the t_{RTRS} rank switch delay.

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

4.2.4 Data Transmission Cost

The memory controller 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 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 by using the expression *CBS = min(N_{ec}, Thresh)*, where N_{ec} is the number of concurrent cores pointing to external banks and $Thresh$ is the maximum batch size set in the RD/WR threshold register. CBS increases as function of N_{ec} but it is limited by the RD/WR batch threshold[3]. The total Data Transmission Cost (DTC) is expressed in Equation 5, where X is *intra* or *inter*.

$$\begin{aligned} DTC^X(\tau_i) \ &= \frac{1}{CBS} * PGNCL^X(\tau_i) * \left(\bar{SP}^X * WT^X + \bar{LP}^X * RT^X\right) + \\ (1 - \frac{1}{CBS}) &* PGNCL^X(\tau_i) * \frac{BL}{2} \end{aligned} \tag{5}$$

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 proportion respectively of $\mathcal{T}$. To calculate (2), we multiply the number of consecutive reads and writes by its burst length ($(1 - \frac{1}{CBS}) * PGNCL^X(\tau_i) * \frac{BL}{2}$).

4.2.5 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

[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 1 to 32 (see [3, 5]).

by another. Equation 6 is used when a row switch is produced in the same bank as τ_i. Otherwise, Equation 7 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 4a and 4b 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 5), when 4 or less banks are used, t_{RRD} is applied. Otherwise, $t_{FAW} - 3 * t_{RRD}$ is used.

$$RSC^{intra} = max(t_{RTP}, t_{WR}) + t_{RP} + t_{RCD} + max(CL, WL) \tag{6}$$

$$RSC^{inter} = \begin{cases} 1 + t_{RRD} & \text{if } NB \leq 4 \\ 1 + t_{FAW} - 3 * t_{RRD} & \text{otherwise} \end{cases} \tag{7}$$

where NB is the number of banks used.

4.2.6 Number of Row Switches

For the intra-bank interference, we start by assuming that for every interfering A_j 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 A_i as the number of actives can't be higher than the number of accesses. To consider the open-row policy, A_i and A_j are divided by $ACOR_i$ and $ACOR_j$ respectively.

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 A_j by its $ACOR_j$. It is assumed that the maximum interfering row switches per bank is limited by the number of ACTs of τ_i, i.e., $\frac{A_i}{ACOR_i}$.

These quantities are calculated with the Number of Row Switches (NRS) function (Equations 8 and 9).

$$NRS^{intra}(\tau_i) = \sum_{\tau_j, j \neq i, B_j = B_i} min\left(\frac{A_i}{ACOR_i}, \frac{A_j * RIE(\tau_i, \tau_j)}{ACOR_j}\right) \tag{8}$$

$$NRS^{inter}(\tau_i) = \sum_{b=0, b \neq B_i}^{N_B - 1} min\left(\frac{A_i}{ACOR_i}, \sum_{\substack{\tau_j, j \neq i \\ tsb(B_j)=1}} S_j + \sum_{\substack{\tau_j, j \neq i, B_j = b \\ tsb(B_j) > 1}} \frac{A_j * RIE(\tau_i, \tau_j)}{ACOR_j}\right) \tag{9}$$

4.2.7 Row Switch Cost

The Total Row Switch Cost (TRSC) (Equation 10) calculates the total impact the interfering row switches have on τ_i. This is done through the multiplication of the number of switches calculated by Equation 8 and 9 by their respective switching cost returned by Equations 6 and 7.

$$\begin{aligned} TRSC^{intra}(\tau_i) &= NRS^{intra}(\tau_i) * RSC^{intra} \\ TRSC^{inter}(\tau_i) &= NRS^{inter}(\tau_i) * RSC^{inter} \end{aligned} \tag{10}$$

4.2.8 Interference Cost Function

Finally, the global Interference Cost (IC) expression is shown in Equation 11. It is the sum of the data transmission cost and the row switch for both cases, the intra-bank and inter-bank interference.

$$\begin{aligned} IC(\tau_i) &= IC(\tau_i)^{intra} + IC(\tau_i)^{inter} = \\ &\quad TRSC^{intra}(\tau_i) + DTC^{intra}(\tau_i) + TRSC^{inter}(\tau_i) + DTC^{inter}(\tau_i) \end{aligned} \tag{11}$$

Algorithm 1 shows how IC(τ_i) is repeatedly called until the interference value converges. This is necessary as explained in Section 4.2.1. 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
$\forall \tau \in \mathcal{T}, IC^n[\tau] = 0$
/* Recursive loop */
while true do
    for each $\tau_i, \tau_j \in \mathcal{T}, \tau_j \neq \tau_i$, compute $RIE^{n+1}(\tau_i, \tau_j)$ by Eq. 2 using $IC^n$
    for each $\tau_i \in \mathcal{T}$, compute $IC^{n+1}(\tau_i)$ by Eq. 3 to 11 using $RIE^{n+1}$
    if $\forall \tau_i \in \mathcal{T}, IC^{n+1}(\tau_i) == IC^n(\tau_i)$ then
        return ($IC^n$, true)
    if $\exists \tau_i \in \mathcal{T}$ such that $C_i + IC^{n+1}(\tau_i) > T_i$ then
        return ($IC^n$, false)
    $n = n + 1$
```

5 Experimentation

5.1 Measurement framework

The measurement framework plays an important role in this work. We need it for: (1) retrieving the properties C_i, A_i, SP_i, LP_i, S_i and $ACOR_i$ of the task (see Section 4.1), (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 available events in [1]). We access these by using a Start-Stop pattern to avoid parasitic overheads due to concurrent management of counters.
- **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 memory controller is able to monitor events regarding the SDRAM device use. 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, SDRAM activates) and filter by master (e.g., ARM Cortex A15, C66x DSP, whole system). The counters registers are accessed using a Start-Read pattern.

5.2 Experimentation Results

Equation 11 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 in our configuration) and normalized with respect to τ_i. The result is added to the normalized τ_i itself, i.e., plus one.

The different 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 [10]. To carry out the cost function equation evaluation each active core runs either a synthetic or a real task. Different cores can execute the same type of task. Table 2 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 considered for the tasks execution. The graphs depicting the normalized execution time (Y axis) do it as function of interfering cores (X axis), which are accumulative.

5.2.1 Monoperiodic

The tests carried out consider a unique period T1 of 900μs. Figures 7, 8 and 9 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 7) we end up having 3.37 and 5.22 units while for the DSP case (Figure 9) 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, it may be noticed that the theoretical value is far from the worst-case measured value, which is normal taking into account the pessimistic assumptions made in Equation 11, 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). Figure 8 shows

Table 2 Tasks main properties on Keystone II

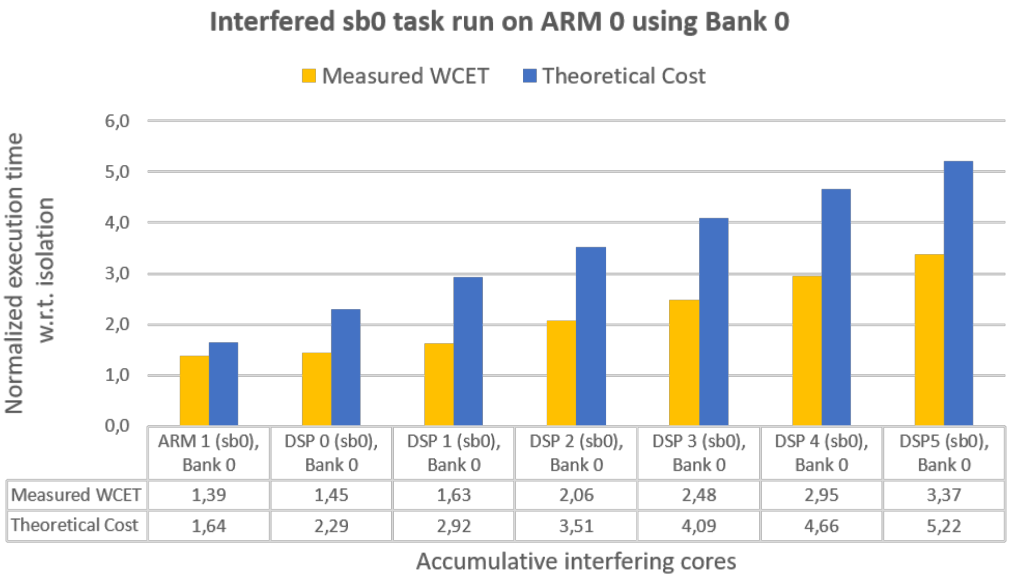
Task	Type	Description	C (Cycles)	A (Accesses)	SP	LP	S (Actives)	ACOR
sb0 (ARM)	Synthetic benchmark	Integer vectors operations.	36315	437	0.312	0.688	0	2.46
sb0 (DSP)			33202	443	0.298	0.702	0	1.70
rb0 (ARM)	Real benchmark	ROSACE engine management.	3107	51	0.628	0.372	16	3.40
rb0 (DSP)			12698	197	0.518	0.482	16	2.20
rb1 (ARM)	Real benchmark	ROSACE altitude filtering.	5110	84	0.631	0.369	14	3.36
rb1 (DSP)			18927	304	0.514	0.486	23	2.10
rb2 (ARM)	Real benchmark	ROSACE elevator management.	6229	96	0.667	0.333	25	3.31
rb2 (DSP)			21708	356	0.520	0.480	32	2.21

the total number of accesses and row switches evolution for the scenario in Figure 7 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 2). 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 8 and 9 by using Equation 1.

Figure 10 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 is also pessimistically modelled. This behavior 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 7 and 9). 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 5). Recall that a Column-First scheduling policy is used by the controller so RD and WR commands of the task under analysis have priority over the PRE or ACT commands from other banks. A scenario where different tasks are used is shown in Figure 11. Intra-bank interference is produced by DSP 0 and DSP 1. The rest of cores introduces inter-bank interference. τ_i is running on an ARM resulting in a short execution time. This makes τ_i be interfered during all its execution by fragments of the other tasks with longer execution times, except for ARM 1 which runs the same task type.

5.2.2 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



	ARM 1 (sb0), Bank 0	DSP 0 (sb0), Bank 0	DSP 1 (sb0), Bank 0	DSP 2 (sb0), Bank 0	DSP 3 (sb0), Bank 0	DSP 4 (sb0), Bank 0	DSP5 (sb0), Bank 0
Measured WCET	1,39	1,45	1,63	2,06	2,48	2,95	3,37
Theoretical Cost	1,64	2,29	2,92	3,51	4,09	4,66	5,22

Figure 7 Execution time of sb0 on ARM 0 as function of other cores. Intra-bank interference.

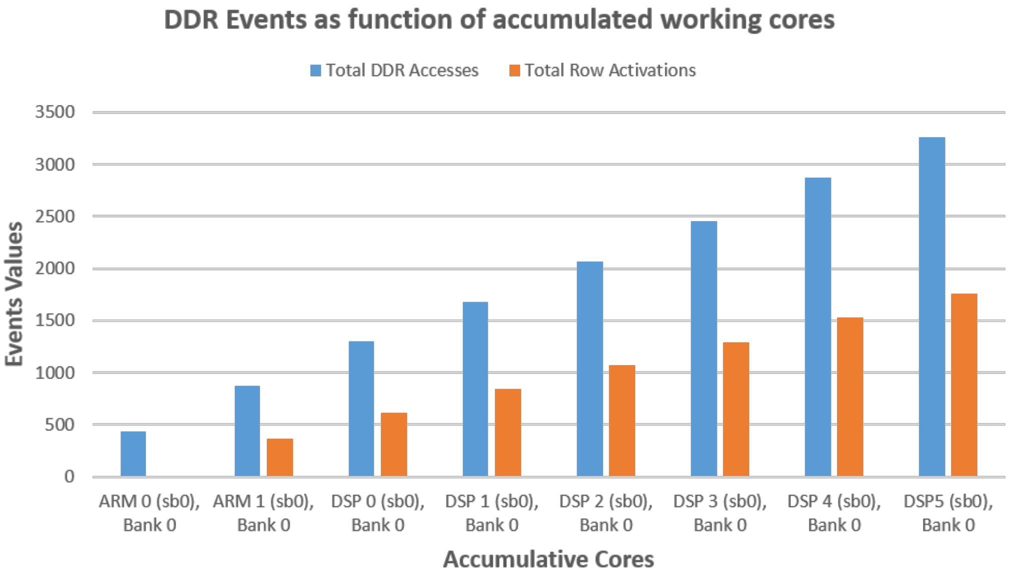
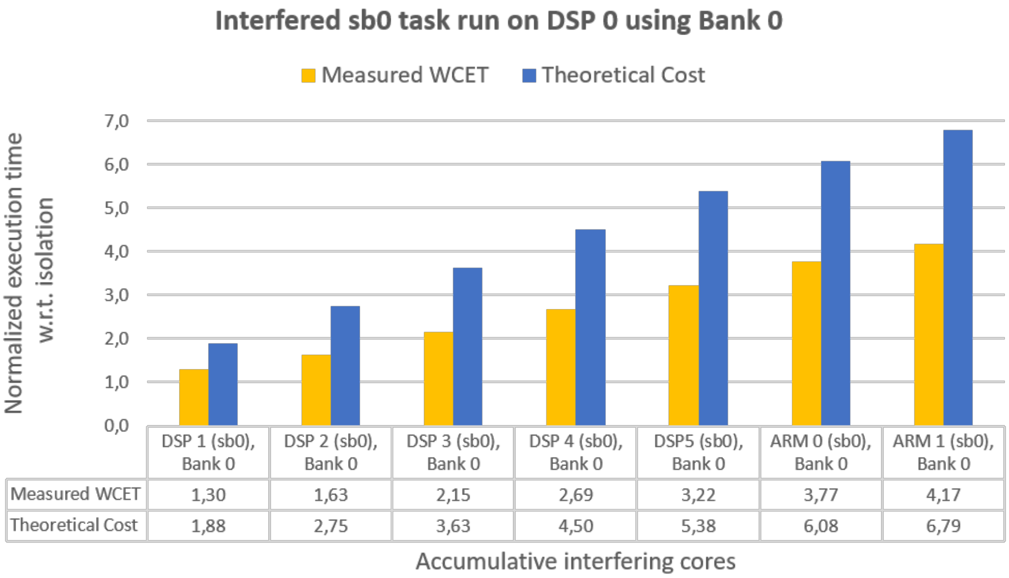
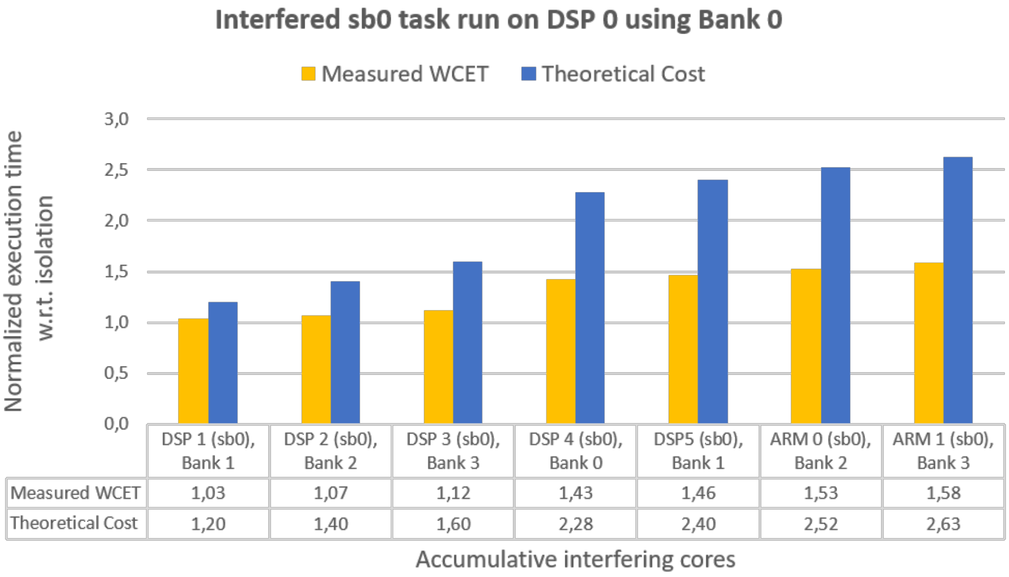


Figure 8 Total system DDR accesses and row buffer activations evolution as function of cores. Intra-bank interference.



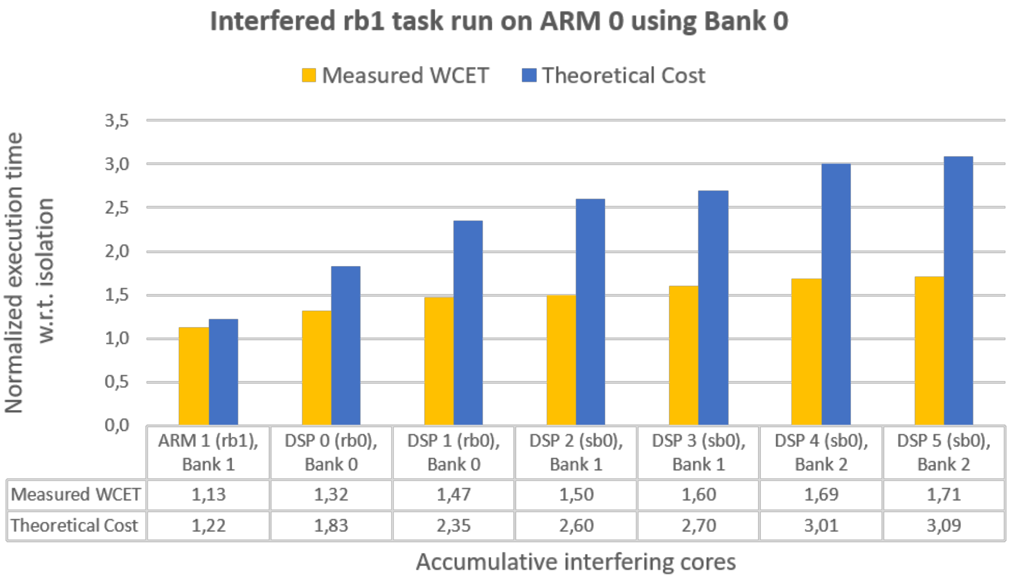
	DSP 1 (sb0), Bank 0	DSP 2 (sb0), Bank 0	DSP 3 (sb0), Bank 0	DSP 4 (sb0), Bank 0	DSP5 (sb0), Bank 0	ARM 0 (sb0), Bank 0	ARM 1 (sb0), Bank 0
Measured WCET	1,30	1,63	2,15	2,69	3,22	3,77	4,17
Theoretical Cost	1,88	2,75	3,63	4,50	5,38	6,08	6,79

Figure 9 Execution time of sb0 on DSP 0 as function of other cores. Intra-bank interference.



	DSP 1 (sb0), Bank 1	DSP 2 (sb0), Bank 2	DSP 3 (sb0), Bank 3	DSP 4 (sb0), Bank 0	DSP5 (sb0), Bank 1	ARM 0 (sb0), Bank 2	ARM 1 (sb0), Bank 3
Measured WCET	1,03	1,07	1,12	1,43	1,46	1,53	1,58
Theoretical Cost	1,20	1,40	1,60	2,28	2,40	2,52	2,63

Figure 10 Execution time of sb0 on DSP 0 as function of other cores. Intra-bank and inter-bank interference.



	ARM 1 (rb1), Bank 1	DSP 0 (rb0), Bank 0	DSP 1 (rb0), Bank 0	DSP 2 (sb0), Bank 1	DSP 3 (sb0), Bank 1	DSP 4 (sb0), Bank 2	DSP 5 (sb0), Bank 2
Measured WCET	1,13	1,32	1,47	1,50	1,60	1,69	1,71
Theoretical Cost	1,22	1,83	2,35	2,60	2,70	3,01	3,09

Figure 11 Execution time of rb1 on ARM 0 as function of other cores. Intra-bank and inter-bank interference.

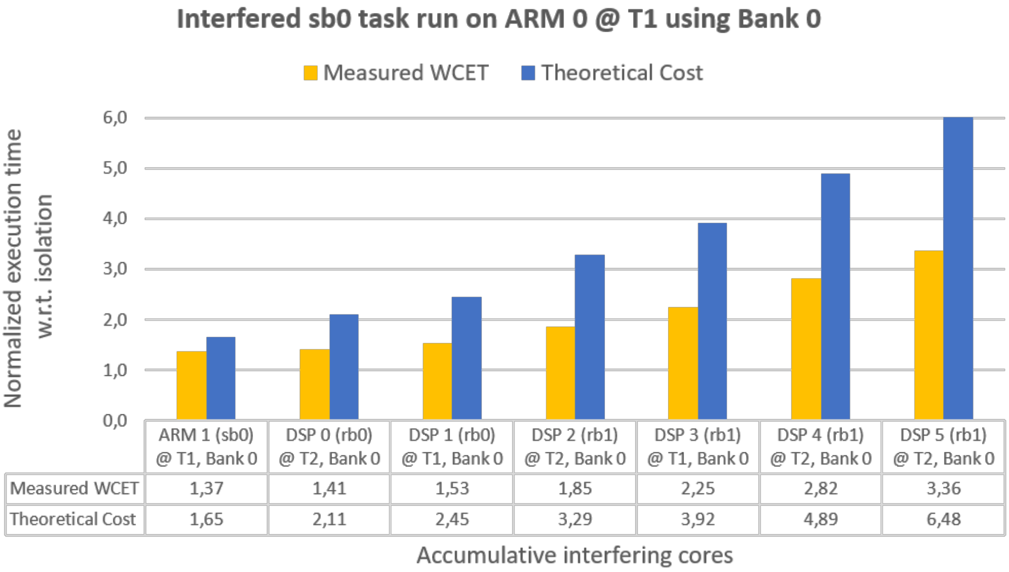
interference as depicted in Figures 6a and 6b. *T1* is considerably high to always assure that $C_i + \alpha * IC_i < T_i$.

Figures 12 and 13 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 with *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 (equivalent to 89698 and 103497 cycles). This reduction in interference is well captured, having a theoretical impact of 4.89 and 6.48 for $T2 = 60\mu s$ (equivalent to 177580 and 235321 cycles) and 4.37 and 5.48 for $T2 = 90\mu s$ (equivalent to 158697 and 199006 cycles).

Figure 14 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 have a direct effect on τ_i as the interference overhead suffered by the task under analysis is not making its execution time exceed *T2*. The highest impact is produced by the DSP 4 addition which is accessing the same bank as τ_i.

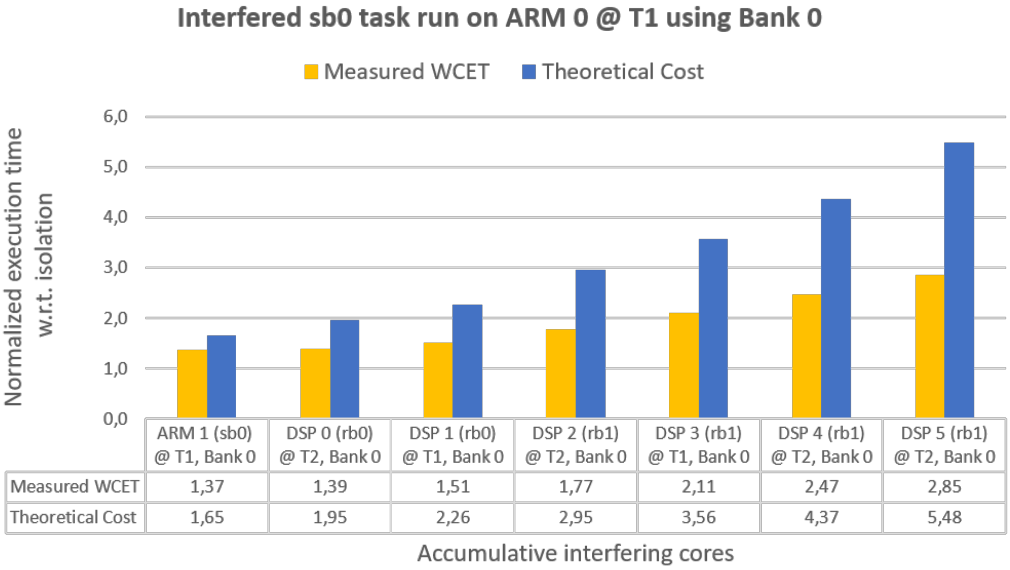
6 Conclusions and Caveats

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. 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 memory controller (e.g.,



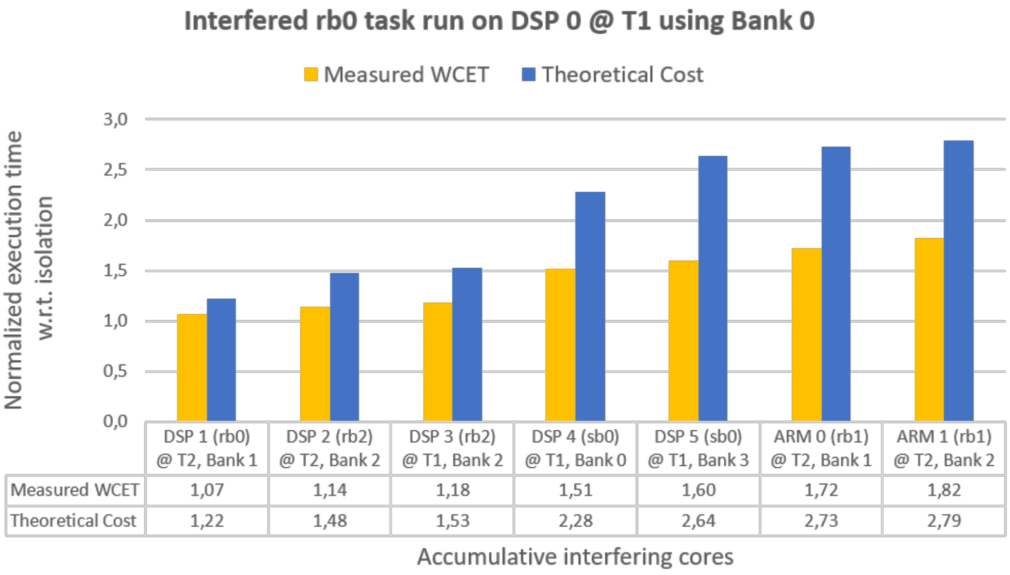
	ARM 1 (sb0) @ T1, Bank 0	DSP 0 (rb0) @ T2, Bank 0	DSP 1 (rb0) @ T1, Bank 0	DSP 2 (rb1) @ T2, Bank 0	DSP 3 (rb1) @ T1, Bank 0	DSP 4 (rb1) @ T2, Bank 0	DSP 5 (rb1) @ T2, Bank 0
Measured WCET	1,37	1,41	1,53	1,85	2,25	2,82	3,36
Theoretical Cost	1,65	2,11	2,45	3,29	3,92	4,89	6,48

Figure 12 Execution time of sb0 on ARM 0 as function of other cores (T1 = 900μs, T2 = 60μs).



	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
Measured WCET	1,37	1,39	1,51	1,77	2,11	2,47	2,85
Theoretical Cost	1,65	1,95	2,26	2,95	3,56	4,37	5,48

Figure 13 Execution time of sb0 on ARM 0 as function of other cores (T1 = 900μs, T2 = 90μs).



	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
Measured WCET	1,07	1,14	1,18	1,51	1,60	1,72	1,82
Theoretical Cost	1,22	1,48	1,53	2,28	2,64	2,73	2,79

Figure 14 Execution time of sb0 on ARM 0 as function of other cores (T1 = 900μs, T2 = 30μs).

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, Nº of accesses). For this reason, it is important to analyze beforehand these previous elements. For instance, in this work, there were no ranks. In case of having them, they should be considered to avoid unnecessary over-estimations. Another element is, for example, the command batch technique, which may vary depending on the controller. Here, it was modeled in an effective but simplistic way. Therefore, it is susceptible to be improved. In the same line, we have the tasks access pattern, which were assumed to be uniformly distributed. Instead, the real distribution could be retrieved and applied, leading to a more accurate interference cost estimation.

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 PHYLOG).

References

1 ARM. *Cortex™-A15 MPCore - Technical Reference Manual*, March 2012.

2 Texas Instruments. *TMS320C66x DSP CPU and Instruction Set*, November 2010.

3 Texas Instruments. *Keystone II Architecture DDR3 Memory Controller - User's Guide*, March 2015.

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

5 Texas Instruments. *AM572x Sitara Processors - Technical Reference Manual*, August 2019.

6 D. Iorga, T. Sorensen, J. Wickerson, and A. F. Donaldson. Slow and steady: Measuring and tuning multicore interference. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

7 JEDEC ASSOCIATION. *JEDEC STANDARD - DDR3 SDRAM - JESD79-3C*, 2008.

8 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*, 52:356–395, 2016.

9 Alfonso Mascareñas González, Frédéric Boniol, Youcef Bouchebaba, Jean-Loup Bussenot, and Jean-Baptiste Chaudron. *Heterogeneous Multicore SDRAM Interference Analysis*, page 12–23. Association for Computing Machinery, New York, NY, USA, 2021.

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

11 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.