Comparison of Memory Access Strategies in Multi-core Platforms Using MAST

Juan M. Rivas[1], J. Javier Gutiérrez[2], Julio L. Medina[2] and Michael González Harbour[2]
[1] PARTS Research Center, Université Libre de Bruxelles (ULB), Belgium.
[2] Software Engineering and Real-Time Group, University of Cantabria (UC), Spain.
[1]jrivasco@ulb.ac.be [2]{gutierjj, medinajl, mgh}@unican.es

***Abstract*—This paper reports solutions to the 2017 edition of the Formal Methods and Timing Verification (FMTV) challenge, which builds upon the 2016 FMTV challenge dealing with calculating latencies in a complex engine management system. The new challenge proposes two new strategies/semantics for task communication through local or global memory in a multi-core platform: (1) implicit communication, which allows access to global memory only at the task boundaries and (2) Logical Execution Time (LET), in which tasks only read global memory at the beginning of their activation interval and write at the end. Based on our previous experience in translating the provided Amalthea model into the MAST model, we propose using the latter and also the associated suite of tools for schedulability analysis and optimization to solve the challenge. Once the new memory access strategies have been properly modelled with MAST, we try to answer some of the questions raised by the challenge using response time analysis. Finally, we discuss the strengths and limitations of our approach according to the results obtained. The solutions are available to the public in electronic form to facilitate their assessment by the community.**

Keywords— Amalthea; MAST; engine management system; task communication; multi-core; response-time analysis; real-time.

I. INTRODUCTION

The 2017 FMTV Challenge [1] extends the 2016 version [2] by adding two new mechanisms for task communication through shared memory. Thus, the challenge asks for a qualitative and quantitative comparison of three different memory access strategies: (1) *explicit*, which is the 2016 strategy, allowing unrestricted access to both local or global memory from the tasks, (2) *implicit*, which allows access to global memory only at the task boundaries and (3) Logical Execution Time (*LET*), in which periodic tasks can only read global memory at the beginning of their activation interval and write at the end. An engine management system (EMS) is provided as a case study through an Amalthea [3] model.

For the 2016 challenge, we proposed in [4] the verification of this system by applying response time analysis (RTA) available in the MAST [5][6] analysis suite. Accordingly, we defined an Amalthea to MAST model transformation path dealing with the explicit memory access strategy. Once this equivalent MAST model was generated, the MAST analysis tools could be used to calculate latencies using common response-time analysis techniques, such as the offset-based analysis [7]. In our approach we had to make some assumptions when (1) interpreting and transforming the provided model, (2) selecting the most appropriate and less pessimistic analysis technique and (3) interpreting the results provided by the tools, especially to compute the latencies of the event chains.

In the previous challenge, we used an incorrect contention model for the access to global memory leading to safe but pessimistic results. We assumed a worst-case situation in which all cores could access global memory at the same time for each label, at a cost of 4*9=36 cycles. The crossbar interconnection network, which imposes 8 out of the 9 cycles required for a single global memory access [8], has no contention. Thus, the worst-case cost for each access to a label located in global memory is 4*1+8=12 cycles (instead of 36 cycles). This issue was clarified during the workshop. Taking into account our previous work on the 2016 FMTV challenge, we propose:

- To reevaluate the behavior of the explicit memory access strategy according to the right contention model for global memory.
- To extend the transformation of the Amalthea model into the MAST model for the new memory access strategies: implicit and LET.
- To reuse the event chain model proposed in our solution to the past challenge, based on the results of the response time analysis obtained by MAST.
- A fair comparison of the latencies calculated by MAST for the three memory access strategies in order to highlight their advantages and drawbacks.
- To use the mapping of labels that we proposed in [4], i.e., mapping into global memory only the labels shared among different cores, thus achieving absence of contention in local memory accesses.

The paper is organized as follows. Section II describes the MAST environment focusing on the most relevant elements used to solve the challenge. Section III summarizes the work done with MAST for the previous challenge. In Section IV, we propose the interpretation of the implicit and LET strategies, and how they are modelled using MAST. Section V, shows the results of the challenge. Finally, in Section VI the conclusions of this work are presented.

This work has been funded in part by the Spanish Government under grant number TIN2014-56158-C4-2-P (M2C2), and by the Walloon Region in Belgium with the BEWARE Project PARTITA (convention no. 1610375).

II. MAST TOOL SUITE

MAST consists of a model [5] and an open source set of tools to perform schedulability analysis and optimization of real-time systems [6]. The MAST model is aligned with MARTE (Modeling and Analysis of Real-Time Embedded systems) [9], a standard of the Object Management Group (OMG) for modeling and analysis of real-time and embedded systems.

A. The MAST model

The MAST model follows an event-driven approach, and assumes a real-time distributed system with multiple processing resources (CPUs and communication networks). The system is composed of distributed end-to-end flows, which are released by periodic, sporadic or aperiodic sequences of external events. The relative phasing of the activations of different end-to-end flows is assumed to be arbitrary. An end-to-end flow is composed of a sequence of steps, which represent the execution of a thread in a processor, or the transmission of a message through a network. Each release of an end-to-end flow causes the execution of one instance of its sequence of steps. Each step is released when the preceding one in its end-to-end flow finishes its execution. We assume that the steps are statically mapped to processing resources. The model also allows mutual exclusion synchronization in the processors.

Fig. 1 shows an example of an end-to-end flow (Γ_i) with three steps (τ_{i1}, τ_{i2}, τ_{i3}), each executing in a different processing resource PR_k. The end-to-end flow is released by the arrival of the external event e_i. This external event has a period T_i, which can also represent the minimum inter-arrival time of a sporadic arrival pattern. Steps can have an initial offset (Φ_{ij}) associated, which is the minimum imposed release time of the step, relative to the arrival of the external event. Each step has a worst-case execution time (WCET), C_{ij}, and a best-case execution time (BCET), C_{ij}^b. MAST supports Fixed Priorities (FP) and Earliest Deadline First (EDF) scheduling. The timing requirements that we consider are end-to-end deadlines (D_i), which must be met by the completion of the last step in the end-to-end flow, relative to the arrival of the external event. The deadlines can be larger than or within the periods.

As a result of the response time analysis, each step τ_{ij} has a worst-case response time (or an upper bound on it) R_{ij}, and a best-case response time (or a lower bound on it) R_{ij}^b. These response times are relative to the arrival of the external event (global response times). The worst-case response time of an end-to-end flow (R_i) is the worst-case response time of its last step. The system is said to be schedulable if the worst-case response times of the end-to-end flows are lower than or equal to their end-to-end deadlines ($R_i \leq D_i$). The completion time of the steps can vary for different activations. As a consequence, the step activation time also varies. For a step τ_{ij}, we define its release jitter (J_{ij}) as its worst-case variation in activation times. The jitter is taken into account by the analysis techniques.

B. MAST analysis tools

MAST implements several analysis techniques that can be applied to an FP system with end-to-end flows, ranging from the holistic analysis, to various offset-based techniques [10]. In the same way as for the 2016 challenge [4], we use the technique called *offset_based_approx_w_pr* in MAST (Offset-Based Analysis with Precedence Relations [7]). This technique supports steps with offsets, and is capable of reducing the pessimism in the results, especially for end-to-end flows with sequential steps in the same processing resource, as it will be the case in this challenge.

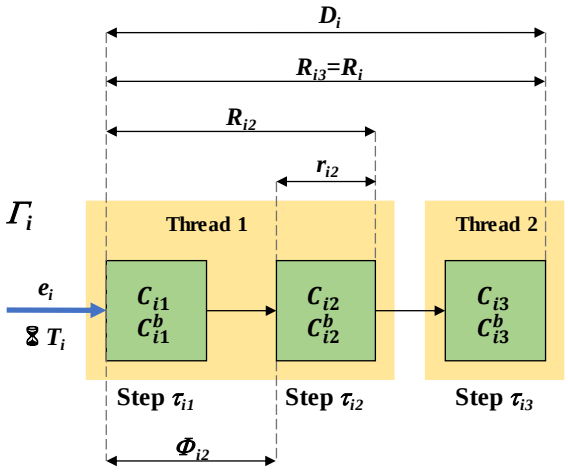

Fig. 1. Example of a simple MAST end-to-end flow with three steps.

Finally, for calculating latencies in event chains, we will need local response times of the steps. We will use the same modified tool as in [4] to provide these local response times according to [11], taking into account offsets. Local worst-case response times (r_{ij}), and local best-case response times (r_{ij}^b) are defined as upper and lower bounds, respectively, on the completion times of steps, relative to their own local activations (see Fig. 1). This custom version of MAST will be made available in addition to the transformation and generated models.

III. SOLUTIONS TO THE 2016 FMTV CHALLENGE WITH MAST

We have recalculated the solutions for the 2016 FMTV challenge [4], which uses the explicit memory access model, taking into account the right contention model for global memory, as indicated in the introduction. This section summarizes the assumptions made on the model transformation to MAST and also on the calculation of latencies of event chains. The complete information can be found in [4].

A. Amalthea to MAST model transformation

Amalthea tasks represent the schedulable elements in the model. For the case of the challenge, they have the following characteristics:

- Tasks are activated by periodic or sporadic stimuli with minimum inter-arrival times. Stimuli are assumed to have arbitrary phasings. Timing constraints are given as deadlines that the tasks must meet. In this case, deadlines are equal to the periods.
- Tasks are statically assigned to cores, and are scheduled with a fixed priority policy. Tasks can be preemptive, or cooperative (they can preempt lower priority cooperative tasks only at the termination of runnables). In the provided model, cooperative tasks always have lower priority than preemptive tasks.

- Each Amalthea task in the model executes a sequential list of runnables. Each runnable is composed of three sequential stages: (1) label (memory) read accesses, (2) execution of instructions in the assigned processing core, and (3) label (memory) write accesses. Some runnables don't write or read from memory.

We interpret Amalthea tasks as MAST end-to-end flows, in which each runnable is transformed into a MAST step. For sporadic Amalthea tasks, the resulting MAST end-to-end flow will be periodic, with a period equal to the minimum inter-arrival time. This interpretation is only correct for flows with offsets within the periods [12]. Since in the Amalthea model the flow deadlines are within the periods so are the step offsets. If the offsets were larger than the periods, the MAST flows would need to be sporadic and the worst-case response times would be larger. The deadline of the Amalthea task is directly used as the end-to-end deadline of its corresponding MAST end-to-end flow.

We will model the memory accesses as execution time added to the MAST steps, accounting for the worst-case and best-case costs of accessing the memory. The worst-case cost of accessing a label pessimistically assumes that every core is accessing that memory at the same time. According to [8], the worst-case cost of accessing a label in global memory is 4*1+8=12 cycles. Similarly, the best-case cost of accessing a label assumes that no other core is in the queue for that memory, so this value is just 9 cycles (no contention). Thus, in the runnable to MAST step transformation, the worst-case execution time of the step (C_{ij}) is calculated as the sum of two elements: (1) the execution time of the upper bound of the number of instructions of the runnable (including local memory label accesses with their frequencies), and (2) the worst-case cost of reading the global labels at the beginning and writing them at the end of the runnable. If a runnable accesses N global labels (read and/or write), the worst-case cost would be N*12 cycles. Likewise, the best-case execution time (C_{ij}^b) of the step is calculated as the sum of the lower bound of the instructions of the runnable (also including local memory label accesses with their frequencies), and the best-case cost of accessing the global labels (N*9 cycles). Fig. 2 depicts the transformation of a simple Amalthea task (Fig. 2a) into a MAST end-to-end flow (Fig. 2b) for the explicit memory access model.

Additionally, we can also model the blocking effect in a thread accessing the memory due to a local label being accessed by a lower priority thread in the same core, although we will not consider it as its effect is negligible [4]. To model cooperative tasks, we will take into account that in the worst-case scenario, these tasks will be blocked by an amount equal to the longest cooperative runnable with lower priority. In MAST we can induce this blocking adding a dummy shared resource that is used by the longest runnable of each cooperative task. This resource uses the immediate ceiling protocol with a ceiling equal to the highest cooperative task's priority. MAST automatically finds the longest possible blocking that affects each task.

B. *Analysis of event chains*

We interpret event chains as a latency model for non-consecutive runnables communicating via shared memory. The first runnable in the event chain writes a result in a label. Then the next runnable in the chain reads this label, process it, and writes its result in another label, and so on. Runnables in an event-chain can belong to the same Amalthea task or not. MAST does not support this kind of "virtual" end-to-end flows, but it provides results that can be used to calculate bounds for the best and worst-case latencies of the event chains.

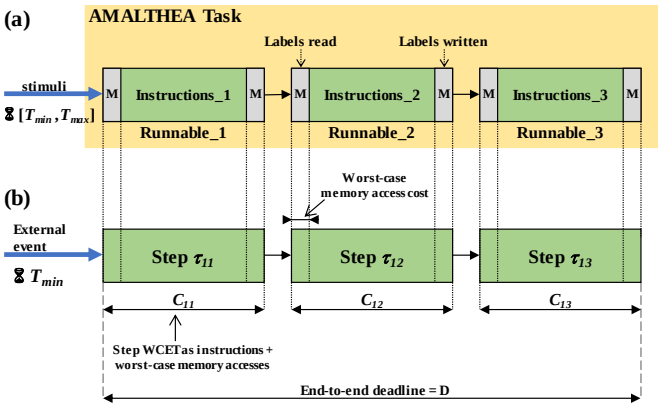

Fig. 2. (a) Example of a simple Amalthea task with three Runnables, and (b) its MAST end-to-end flow equivalent used in this work

We distinguish two types of event chains [4]:

- *Event chains that stay in the same Amalthea task (EffectChain_1).* We can notice that to go backwards, the chain requires an additional activation of the Amalthea task. The worst-case latency (L) occurs when the first label in the chain is read as soon as possible, so the chain has to wait the maximum amount of time until the next activation of the end-to-end flow. Then, the event chain must wait for the worst-case completion time of the last step. Since the end-to-end flow must finish before its next activation, the response times of middle steps are irrelevant in this calculation. Likewise, the best-case latency (L^b) of the event chain occurs when the first label is read as late as possible and the last step finishes as soon as possible.

- *Event chains that traverse different Amalthea tasks (EffectChain_2 and EffectChain_3).* The worst-case latency of the event-chain (L) comprises the sum of the worst-case local response times of the steps in the chain (r_{ij}), and the periods of all the end-to-end flows involved in the chain except the one hosting the initial step in the chain. The periods should be added because in the worst-case situation it is assumed that at the time a label is written, the next runnable in the chain has just executed, so the chain cannot continue until the next period. For sporadic stimuli, the period added must be its upper bound. Similarly, the best-case latency (L^b) is calculated by adding the best-case local response times (r_{ij}^b). In this case periods are not added, because the best case is built when a label is read immediately after the previous runnable in the chain updated its value.

The formulation for the worst and best case latencies for the event chains as well as a further explanation can be found in [4].

IV. Implicit and LET memory access in MAST

The current challenge introduces new memory access models that group read and write memory operations of tasks to be performed at the beginning or at the end of the task execution (implicit model) or at the beginning or at the end of the period (LET model). This organization of read and write memory operations affects the transformation of the original Amalthea model to the MAST model, as well as the interpretation of the event chains according to the information obtained by the schedulability analysis for these new models.

A. *Amalthea to MAST model transformation*

The interpretation of an Amalthea task for the implicit and LET memory access models as MAST end-to-end flows differs from the explicit model only in the organization of steps representing the execution of runnables and read and write operations, and in the specific instants at which some of these steps can be executed. Fig. 3 shows the equivalent MAST model for the implicit memory access model (Fig. 3a) and for the LET memory access model (Fig. 3b).

As it can be seen, for simplicity, the implicit model considers all runnables of a task (including global read and write operations) grouped into a single step (Fig. 3a). This is possible because all these actions are executed at the same priority. In the same way done for the explicit memory access model, the memory accesses are considered as execution times added to this step according to the mapping of labels. In this model, read and write operations are executed at the priority of the task and they can be preempted or not depending on the kind of task (non-cooperative or cooperative). This interpretation does not compromise data coherence within the task itself. Inter-task data coherence is not assured as it is possible to have concurrent data access from different cores and there is no mutual exclusion mechanism specified.

The LET memory access model needs to control the instants at which read and write operations should be executed. For this reason, the equivalent MAST model of an Amalthea task is an end-to-end flow composed of three steps (Fig. 3b): (1) an initial step grouping all the global-memory read operations, (2) a middle step grouping all the activities performed by the runnables (except global reads and writes) and (3) a final step grouping all the global-memory write operations. The execution of the first step should be as close as possible to the beginning of the period, and the final step is delayed with an offset trying to make it coincide with the end of the period. To accomplish these requirements it is necessary to achieve low jitter in the execution of the initial and final step, which requires executing them at high priority levels. In our model, these steps are executed by different tasks making it possible to assign different priorities to them. Please note that modeling these steps with different tasks does not require having different tasks in the actual implementation. It is possible to have a single task for each end-to-end flow, dynamically setting its priority according to the step that is being executed. To assign priorities to each step we propose the use of priority bands in the following way:

- 2 priority bands: a low priority band for the execution of runnables at the priority assigned in the Amalthea model, and a high priority band for the execution of read and write operations under the following conditions: (1) reads and writes of the same end to-end flow are executed at the same priority and (2) the original priority order is preserved in this band.
- 3 priority bands: a low priority band for the execution of runnables, a medium priority band for the execution of read operations and a high priority band for the execution of write operations. The same criteria as for the previous case is used inside each band.

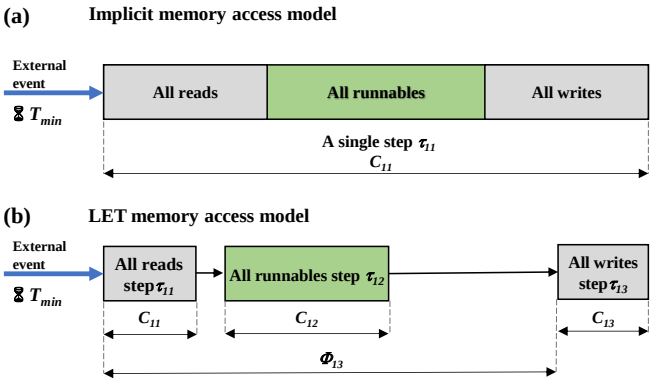

Fig. 3. MAST end-to-end flow equivalent to an Amalthea task for (a) the implicit and (b) the LET memory access models

The use of priority bands allows minimizing jitter. In addition, it is necessary to use offsets to control the timing of the write operations, so that their execution is postponed to the end of the period. We consider two different offset assignment strategies:

- *LET-Static*. Offsets are set equal to periods, thus write operations are always performed as the first operation of the next activation, except for the first one.
- *LET-Dynamic*. The objective is that the worst case response times of the write operations equal the corresponding periods, thus ensuring that these operations finish as close as possible to the end of their instance period. An algorithm is needed for assigning offsets, as response times depend on offsets and the assignment of offsets depends on the response times. We propose the following one:

```
Offsets calculation for LET-Dynamic
begin
   offset=T for all end-to-end flows;
   loop
      Calculate response times;
      exit when R=T for all end-to-end flows;
      for each end-to-end flow
      loop
         if R>=T then
            offset=offset-(R-T);
         else
            offset=T-(R-offset);
         end if;
      end loop;
   end loop;
end;
```

For the challenge example this algorithm requires only one iteration for the model with 3-priority bands, and three iterations for 2-priority bands.

B. Analysis of event chains

In the explicit memory access model the execution of each runnable includes read-execute-write operations, and thus the analysis of the chain can focus on the individual local response times of each runnable. However, in the implicit or LET models reads and writes are only performed at the beginning or at the end of the task, and thus the complete task global response times must be considered. Based on this observation, we reformulate the equations proposed in [4] for the two types of event chains:

- *Event chains that stay in the same Amalthea task*. For the implicit memory access model, we use the same formulation than for the explicit model to calculate latencies, but changing local response times into global response times:

$$L = (T - R^b) + R$$

$$L^b = (T - R) + R^b$$

  For the LET memory access model, independently of whether the task finishes within the period or just after the period, the second activation to conclude the event chain always finishes with the worst or the best case response times after the first period, i.e.:

$$L = T + R$$

$$L^b = T + R^b$$

- *Event chains that traverse different Amalthea tasks*. In this case, both the implicit and LET memory access models share the equations to calculate latencies, which can be expressed as for the explicit model changing local response times into global response times, as follows:

$$L = R_1 + T_2 + R_2 + T_3 + R_3$$

$$L^b = R_1^b + R_2^b + R_3^b$$

V. Evaluation

To transform the Amalthea model to MAST we use an updated version of the *ad-hoc* tool written in Java developed for the previous challenge [4]. This tool reads Amalthea models using the Eclipse EMF framework [13], and builds equivalent MAST models using the interpretations described in Section III and Section IV. We provide end-to-end latencies and jitters for the Amalthea tasks and for the event-chains described in the model, as well as offsets and jitters for the write operations of the LET-Dynamic case. The analysis technique used has been the Offset-Based Analysis with Precedence Relationships [7]. This is the least pessimistic technique for end-to-end flows that only traverse one processor.

We test two cases: (1) with contention, i.e., considering the contention effect in the crossbar when accessing global memory (12 cycles in the worst case) and (2) without contention, i.e., assuming that access to global memory is always performed without waiting other cores (9 cycles in the worst case). The latter leads to an optimistic calculation of latencies, but it serves as an estimation of the influence of the contention in the calculation of latencies.

TABLE I. CORE UTILZATIONS (%) FOR THE ORIGIAL FREQUENCY AND THE ONE USE IN OUR TESTS.

	With contention		**Without contention**	
	200 MHz	***300 MHz***	***200 MHz***	***300 MHz***
CORE0 Util. (%)	97.35	64.90	97.31	64.88
CORE1 Util. (%)	135.69	90.46	135.39	90.26
CORE2 Util. (%)	107.51	71.68	107.45	71.63
CORE3 Util. (%)	119.43	79.62	119.22	79.48

As we pointed out in [4], the total utilization of the system is above 100% using the 200 MHz processor specified in the challenge, and thus this workload cannot be managed by using response time analysis. We will assume that we have hardware that is fast enough to execute the code below 100% utilization even in the worst-case scenario. We explore common CPU clock frequencies and we find that 300 MHz is the first one that allows the schedulability of the proposed application. Table I shows the utilization of the different cores for the original and the selected frequencies, and also for the cases with and without contention. All the experiments reported have been done for a 300 MHz processor. We can also notice a low variation (less than 0.2% for 300 MHz) in the core utilizations between the cases with and without contention. The reason for this low variation is that our mapping allocates most of the labels in local memory and the access to a global label is only reduced from 12 to 9 cycles between one case and the other.

Tables II, III and IV show the latencies and jitters for all the tasks and also the worst and best case latencies for the event chains, for the explicit, implicit and LET-Static memory access

TABLE II. END-TO-END LATENCIES AND JITTERS (MILLISECONDS) FOR TASKS AND EVENT CHAINS IN **THE EXPLICIT MEMORY ACCESS MODEL**.

	With contention		**Without contention**		
	L	***J***	***L***	***J***	***D***
Angle_Sync	5.684	4.766	5.671	4.760	**6.66**
ISR_1	0.024	0.013	0.024	0.013	**9.5**
ISR_10	0.021	0.009	0.020	0.009	**0.7**
ISR_11	1.297	1.205	1.297	1.204	**5**
ISR_2	0.036	0.029	0.036	0.029	**9.5**
ISR_3	0.052	0.043	0.052	0.043	**9.5**
ISR_4	0.459	0.347	0.458	0.347	**1.5**
ISR_5	0.193	0.107	0.193	0.107	**0.9**
ISR_6	0.214	0.204	0.214	0.204	**1.1**
ISR_7	0.899	0.783	0.899	0.783	**4.9**
ISR_8	0.662	0.574	0.661	0.574	**1.7**
ISR_9	2.227	2.108	2.226	2.107	**6**
Task_1000ms	31.346	31.293	31.324	31.272	**1000**
Task_100ms	31.162	29.031	31.142	29.016	**100**
Task_10ms	7.960	5.201	7.946	5.201	**10**
Task_1ms	0.516	0.343	0.515	0.343	**1**
Task_200ms	31.252	31.198	31.231	31.178	**200**
Task_20ms	9.629	7.179	9.623	7.178	**20**
Task_2ms	0.271	0.177	0.271	0.177	**2**
Task_50ms	13.068	12.182	13.060	12.176	**50**
Task_5ms	0.895	0.649	0.894	0.649	**5**
	L	***L^b***	***L***	***L^b***	
EffectChain_1	10.305	3.761	10.312	3.769	
EffectChain_2	25.517	0.033	25.509	0.033	
EffectChain_3	63.920	0.030	63.913	0.030	

TABLE III. END-TO-END LATENCIES AND JITTERS (MILLISECONDS) FOR TASKS AND EVENT CHAINS IN **THE IMPLICIT MEMORY ACCESS MODEL**.

	With contention		Without contention		
	L	***J***	***L***	***J***	***D***
Angle_Sync	5.684	4.766	5.671	4.760	**6.66**
ISR_1	0.024	0.013	0.024	0.013	**9.5**
ISR_10	0.021	0.009	0.020	0.009	**0.7**
ISR_11	1.297	1.205	1.297	1.204	**5**
ISR_2	0.036	0.029	0.036	0.029	**9.5**
ISR_3	0.052	0.043	0.052	0.043	**9.5**
ISR_4	0.459	0.347	0.458	0.347	**1.5**
ISR_5	0.193	0.107	0.193	0.107	**0.9**
ISR_6	0.214	0.204	0.214	0.204	**1.1**
ISR_7	0.899	0.783	0.899	0.783	**4.9**
ISR_8	0.662	0.574	0.661	0.574	**1.7**
ISR_9	2.227	2.108	2.226	2.107	**6**
Task_1000ms	31.345	31.292	31.324	31.271	**1000**
Task_100ms	31.161	29.031	31.141	29.016	**100**
Task_10ms	7.959	5.201	7.945	5.201	**10**
Task_1ms	0.516	0.343	0.515	0.343	**1**
Task_200ms	31.252	31.197	31.231	31.177	**200**
Task_20ms	9.629	7.179	9.623	7.178	**20**
Task_2ms	0.271	0.177	0.271	0.177	**2**
Task_50ms	13.068	12.182	13.060	12.175	**50**
Task_5ms	0.895	0.649	0.894	0.649	**5**
	L	***L^b***	***L***	***L^b***	
EffectChain_1	15.201	4.799	15.201	4.799	
EffectChain_2	51.391	4.982	51.357	4.963	
EffectChain_3	65.360	0.992	65.351	0.990	

TABLE IV. END-TO-END LATENCIES AND JITTERS (MILLISECONDS) FOR TASKS AND EVENT CHAINS IN **THE LET-STATIC MODEL WITH CONTENTION**.

	2 priority bands		3 priority bands		
	L	***J***	***L***	***J***	***D***
Angle_Sync	6.694	0.023	6.672	0.001	**6.66**
ISR_1	9.500	0.000	9.500	0.000	**9.5**
ISR_10	0.700	0.000	0.700	0.000	**0.7**
ISR_11	5.002	0.001	5.002	0.001	**5**
ISR_2	9.500	0.000	9.500	0.000	**9.5**
ISR_3	9.500	0.000	9.500	0.000	**9.5**
ISR_4	1.501	0.001	1.501	0.001	**1.5**
ISR_5	0.900	0.000	0.900	0.000	**0.9**
ISR_6	1.101	0.000	1.101	0.000	**1.1**
ISR_7	4.901	0.001	4.901	0.001	**4.9**
ISR_8	1.701	0.001	1.701	0.001	**1.7**
ISR_9	6.002	0.002	6.002	0.002	**6**
Task_1000ms	1.000.056	0.055	1.000.027	0.025	**1000**
Task_100ms	100.049	0.037	100.024	0.012	**100**
Task_10ms	10.059	0.028	10.032	0.000	**10**
Task_1ms	1.004	0.003	1.001	0.000	**1**
Task_200ms	200.054	0.052	200.025	0.024	**200**
Task_20ms	20.023	0.014	20.010	0.001	**20**
Task_2ms	2.001	0.000	2.000	0.000	**2**
Task_50ms	50.028	0.026	50.012	0.010	**50**
Task_5ms	5.002	0.001	5.001	0.000	**5**
	L	***L^b***	***L***	***L^b***	
EffectChain_1	20.059	20.032	20.032	20.032	
EffectChain_2	124.109	112.044	124.056	112.044	
EffectChain_3	104.729	52.703	104.712	52.702	

models, respectively. Table V presents the offsets calculated for the LET-Dynamic memory access model using the algorithm proposed in Section IV, as well as jitters for tasks and the worst and best case latencies for the event chains. In Tables II and III

TABLE V. OFFSETS, JITTERS AND END-TO-END LATENCIES (MILLISECONDS) FOR TASKS AND EVENT CHAINS IN **THE LET-DYNAMIC MODEL WITH CONTENTION**.

	2 priority bands		3 priority bands		
	Φ	***J***	***Φ***	***J***	***D***
Angle_Sync	6.645	0.004	6.648	0.001	**6.66**
ISR_1	9.500	0.000	9.500	0.000	**9.5**
ISR_10	0.700	0.000	0.700	0.000	**0.7**
ISR_11	4.998	0.001	4.998	0.001	**5**
ISR_2	9.500	0.000	9.500	0.000	**9.5**
ISR_3	9.500	0.000	9.500	0.000	**9.5**
ISR_4	1.499	0.001	1.499	0.001	**1.5**
ISR_5	0.900	0.000	0.900	0.000	**0.9**
ISR_6	1.099	0.000	1.099	0.000	**1.1**
ISR_7	4.899	0.001	4.899	0.001	**4.9**
ISR_8	1.699	0.001	1.699	0.001	**1.7**
ISR_9	5.998	0.002	5.998	0.002	**6**
Task_1000ms	999.945	0.054	999.973	0.025	**1000**
Task_100ms	99.960	0.028	99.976	0.012	**100**
Task_10ms	9.968	0.000	9.968	0.000	**10**
Task_1ms	0.999	0.000	0.999	0.000	**1**
Task_200ms	199.949	0.049	199.975	0.024	**200**
Task_20ms	19.989	0.002	19.990	0.001	**20**
Task_2ms	2.000	0.000	2.000	0.000	**2**
Task_50ms	49.975	0.023	49.988	0.010	**50**
Task_5ms	4.999	0.001	4.999	0.000	**5**
	L	***L^b***	***L***	***L^b***	
EffectChain_1	20.000	20.000	20.000	20.000	
EffectChain_2	124.000	112.000	124.000	112.000	
EffectChain_3	104.700	52.700	104.700	52.700	

the results are presented for the cases with and without contention for the implicit and explicit memory access models. As there are no significant variations in latencies or jitters for these cases, the results for LET-Static and LET-Dynamic included in Tables IV and V show only the case with contention, focusing on the differences for the cases with 2 or 3 priority bands.

The worst-case latencies for the explicit memory access model (Table II) are slightly lower than those obtained in [4] due to the more precise model used for the contention in the crossbar. Compared to the implicit memory access model (Table III), we can see that similar latencies and jitters are obtained for the tasks, but the worst case latencies of the event chains are higher in the implicit memory access model. Looking at the temporal behavior, this implicit model does not seem to be a good solution as it has similar or worst latencies than the explicit model without having any significant reduction of jitter.

Finally, the LET memory access model (Table IV and Table V) minimizes jitter at the cost of increasing of the worst and best case latencies for both tasks and event chains. Using 3 priority bands allows a better control of writing activities. The LET-Dynamic model allows tasks to finish their executions within the deadlines through the configuration of offsets, while in the LET-Static model the tasks always finish behind the deadlines.

VI. CONCLUSIONS

This paper builds upon the work done in the previous FMTV challenge to provide solutions to the two new memory access models proposed for the multicore platform: implicit and LET. We extend the general guidelines to transform an Amalthea timing model into a MAST equivalent model that can be used in

the MAST Analysis Tool Suite. Thus, response time analysis has been applied to calculate latencies and jitters of tasks and event chains for the different memory access models (explicit, implicit and LET). To guarantee LET behavior, offsets have been obtained for the write operations.

The paper partially answers the five main questions of the challenge: (1) we have proposed solutions for the implicit and LET memory access models with the objective of minimizing jitter, (2) we have modeled and computed the overheads for the three different memory access models; additionally we have calculated latencies, jitters and offsets for the tasks, (3) we have calculated the latencies (worst and best) for the event chains, (4) we have used a mapping of labels that reduces overheads in memory access and (5) we have studied the effects of contention in the crossbar.

From the evaluation of the temporal behavior done in this work for the three memory access models, we can extract the following conclusions: (1) none of the strategies is a clear winner, i.e., there is not a best one that allows the reduction of latencies and jitters at the same time, (2) the implicit model has similar latencies than the explicit one for the tasks and higher latencies for the event chains without getting a reduction of jitter; so there is a penalty to keep data consistency and (3) the LET model has a good control of jitter at the cost of a significant increase of latencies for both tasks and event chains.

A straightforward modification of the LET model using shorter offsets would allow keeping a small jitter on the write operations while minimizing the worst-case response time at the same time. The offsets would just need to be calculated equal to the worst-case response times of the tasks executing the runnables.

The workspace used in this paper can be downloaded from [14].

REFERENCES

[1] 2017 Formal Methods for Timing Verification (FMTV) challenge, co-located with the 8th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS). https://waters2017.inria.fr/challenge/

[2] 2016 Formal Methods for Timing Verification (FMTV) challenge, co-located with the 7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS). https://waters2016.inria.fr/challenge/

[3] AMALTHEA: An Open Platform Project for Embedded Multicore Systems, http://www.amalthea-project.org/

[4] J.M. Rivas, J.J. Gutiérrez, J.L. Medina, and M. González Harbour, "Calculating Latencies in an Engine Management System Using Response Time Analysis with MAST," 7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), FMTV Challenge 2016, Toulouse (France), 2016.

[5] M. González Harbour, J.J. Gutiérrez García, J.C. Palencia Gutiérrez, and J.M. Drake Moyano, "MAST: Modeling and Analysis Suite for Real Time Applications," Proceedings of 13th ECRTS conference, Delft, The Netherlands, IEEE Computer Society Press, pp. 125-134, June 2001.

[6] MAST web-page, http://mast.unican.es/

[7] J. C. Palencia and M. González Harbour, "Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems," Proceedings of the 20th Real-Time Systems Symposium, IEEE Computer Society Press, pp 328-339, December 1999.

[8] I. Stierand, P. Reinkemeier, S. Gerwinn, and T. Peikenkamp, "Computational Analysis of Complex Real-Time Systems - FMTV 2016 Verification Challenge," 7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), FMTV Challenge 2016, Toulouse (France), 2016.

[9] Object Management Group, "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems," 2011 OMG Document, v1.1 formal/2011-06-02.

[10] MAST Analysis Techniques, http://mast.unican.es/mast_analysis_techniques.pdf

[11] J.C. Palencia, J.J. Gutiérrez, and M. González Harbour. "On the Schedulability Analysis for Distributed Hard Real-Time Systems," Proc. of the 9th Euromicro Workshop on Real-Time Systems, pp. 136-143, June 1997.

[12] J. C. Palencia. "Análisis de planificabilidad de sistemas distribuidos de tiempo real basados en prioridades fijas", Phd Thesis, University of Cantabria, July 1999.

[13] Eclipse Modeling Framework (EMF), https://eclipse.org/modeling/emf/

[14] Amalthea workspace used for this solution, www.istr.unican.es/members/rivasjm/workspace_fmtv17_public.html