# Calculating Latencies in an Engine Management System Using Response Time Analysis with MAST

Juan M. Rivas, J. Javier Gutiérrez, Julio L. Medina and Michael González Harbour
Software Engineering and Real-Time Group, University of Cantabria, Spain.
{rivasjm, gutierjj, medinajl, mgh}@unican.es

***Abstract*—This paper reports solutions to the 2016 edition of the Formal Methods and Timing Verification (FMTV) challenge. The challenge requests calculating latencies in a complex engine management system, of which an Amalthea model is provided. We propose solving the challenge using MAST, which is a real-time systems model and also a suite of tools for schedulability analysis and optimization. The efforts to solve the challenge are mainly focused on translating the Amalthea model into the MAST model. Then, response time schedulability analysis tools are used. We discuss the strengths and limitations of our approach, and present the results obtained. Finally, we report the time needed to understand and complete the challenge. The solutions are available to the public in electronic form to facilitate their assessment by the community.**

***Keywords— Amalthea; MAST; engine management system; real-time, response-time analysis.***

## I. INTRODUCTION

This paper presents a solution to the 2016 FMTV Challenge [1] which asked calculating tight end-to-end latency bounds in a complex engine management software composed of a number of cause-effect chains. The system is provided as an Amalthea [2] model.

We propose the verification of this system by applying response time analysis (RTA) inside the MAST [3][4] analysis suite. Accordingly, the first effort that must be undertaken is to define an Amalthea to MAST model transformation path. Once an equivalent MAST model is generated, the MAST analysis tool can be used to calculate latencies, using common response-time analysis techniques, such as the offset-based analysis [5]. Using MAST enables the application of complex mathematical formulation to perform the response time analysis on an easy to understand high level abstraction model. This approach requires: (1) the correct interpretation and transformation of the provided model, (2) the selection of the most appropriate and less pessimistic analysis technique, and (3) the correct interpretation of the results provided by the tools.

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 deals with the interpretation of the provided Amalthea model, and how it is modelled using MAST. Section IV proposes an analysis for event chains. In Section V, the challenge results are presented. Finally, Section VI presents the conclusions of this work.

This work has been funded in part by the Spanish Government under grant number TIN2014-56158-C4-2-P (M2C2).

## II. MAST TOOL SUITE

The MAST environment provides an open source set of tools to perform schedulability analysis and optimization of real-time systems [4]. These tools operate on systems described using the MAST model [3], which is key to our solution of the challenge. This model is aligned with MARTE (Modeling and Analysis of Real-Time Embedded systems) [6], 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

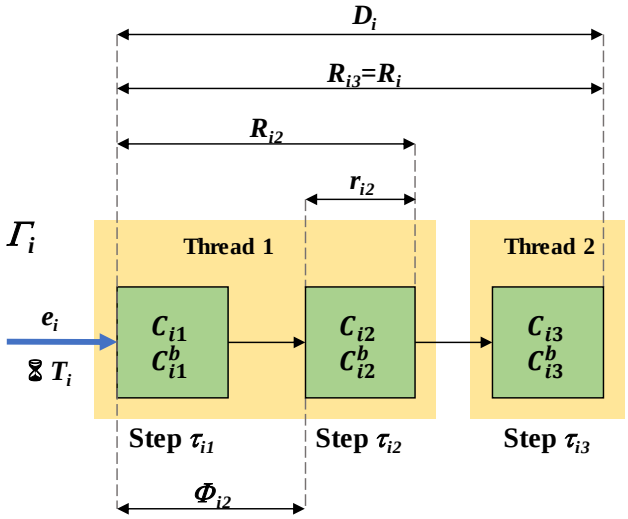


Fig. 1. Example of a simple MAST end-to-end flow with three steps.

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 ($\Gamma_i$) with three steps ($\tau_{i1}$, $\tau_{i2}$, $\tau_{i3}$), 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 ($\Phi_{ij}$) 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 the periods.

As a result of the response time analysis, each step $\tau_{ij}$ has a worst-case response time (or an upper bound of it) $R_{ij}$, and a best-case response time (or a lower bound of 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 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 $\tau_{ij}$, 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

To solve the challenge, we use the response-time analysis techniques included in MAST [4] on the equivalent MAST model generated from the Amalthea model. 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 [4].

Of particular interest for this work is the Offset-Based Analysis with Precedence Relationships [5] (*offset_based_approx_w_pr* in MAST). This technique supports steps with offsets, and is capable of reducing the pessimism in the results by eliminating scenarios that would be impossible when taking into account the precedence relationships inside end-to-end flows. This characteristic is particularly helpful with end-to-end flows that don't traverse different processing resources, as it will be the case in this challenge.

Additionally, MAST can also perform sensitivity analysis by calculating the system slack, which, if positive, is defined as the percentage by which the execution times of all the steps in the system may be increased while still keeping the system schedulable. If negative, the system slack corresponds to the percentage by which WCET's would have to be decreased to make the system schedulable. Similarly, slacks for each processor can be calculated too.

MAST provides global worst-case and best-case response times of the steps in the system. For a part of this challenge we will need local response times of the steps. While these are not usually provided by MAST, we have modified the tool so it could handle local response-times too, according to [7] taking into account offsets. Then, we define local worst-case response times ($r_{ij}$), and local best-case response times ($r_{ij}^b$) 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. AMALTHEA TO MAST MODEL TRANSFORMATION

The 2016 FMTV Challenge provides an Amalthea model of a full blown engine management system. The complexity of the system is made apparent just by looking at the model file, which has approximately 71000 lines. In this section, we will describe how we interpret the Amalthea model, and how the equivalent MAST description of the engine management system is created. While Amalthea defines a vast meta-model supporting many types of elements and use-cases, we will limit our transformation to the elements relevant for this challenge.

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 phasing (property "Clock" of the stimuli is undefined). Timing constraints are given as deadlines that the tasks must meet. In this case, deadlines are equal to the periods (tasks must finish before their next activation).
- Tasks are statically assigned to a core, and are scheduled with a fixed priority policy. Tasks can be preemptive (they can preempt any lower priority task at any moment), 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 [8]. 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.

MAST lacks a specialized element to model memories. Additionally, it also doesn't implement any mechanism to model the blocking of a processor while it is accessing a memory, thus disallowing us to model memory as a general purpose device. With these limitations in mind, we will model the memory accesses as execution times 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. Therefore, if we consider that only the global memory is used (second question of the challenge), the worst-case cost of accessing a label is 4*9 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).

Accordingly, 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, and (2) the worst-case cost of accessing the labels. If a runnable accesses N labels (read and/or write), the worst-case cost would be N*4*9 cycles if we assume that only global memory is used. 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, and the best-case cost of accessing the labels (N*9 cycles).

Additionally, we also take into account the blocking effect in a thread accessing the memory due to a label being accessed by a lower priority thread in the same core, even though this is almost negligible. This is modeled by including in each core a shared resource protected by the Immediate Ceiling protocol that is accessed by each step during 9 cycles. This produces one blocking of 9 cycles to each higher priority thread, which is the intended effect.

Fig. 2 depicts the transformation of a simple Amalthea task (Fig. 2a) into a MAST end-to-end flow (Fig. 2b). If memory accesses are ignored, as stated in the first question of the challenge, the executions times of the resulting MAST steps only include the execution times produced by the instructions.

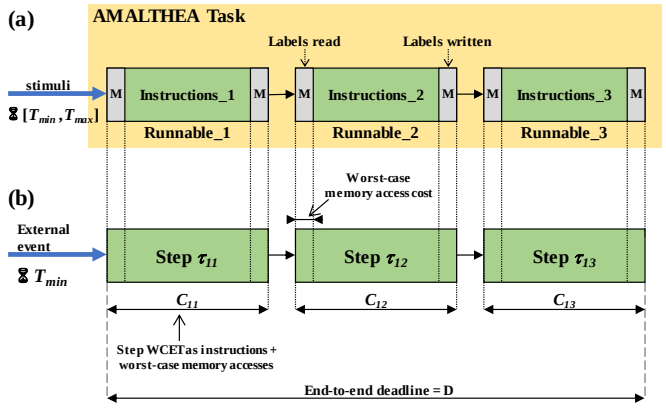


Fig. 2. (a) Example of a simple Amalthea task with three Runnables, and (b) its MAST end-to-end flow equivalent used in this work

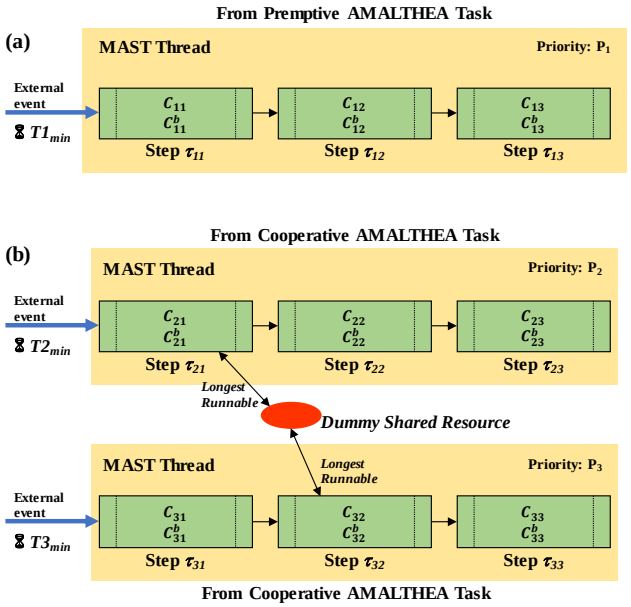


Fig. 3. Equivalent Amalthea tasks as MAST end-to-end flows, for (a) preemptive , and (b) cooperative Amalthea tasks.

MAST supports non-preemptive tasks, but they cannot be preempted by any task. This is not aligned with the behavior of Amalthea cooperative tasks, which can be preempted by preemptive tasks. 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. MAST automatically finds the longest possible blocking that affects each task. Fig. 3a depicts a MAST end-to-end flow transformed from a preemptive Amalthea task, while Fig. 3b shows the transformation of two Amalthea cooperative tasks.

## IV. 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. Even though MAST does not support this kind of "virtual" end-to-end flows, it provides results that can be used to calculate bounds for the best and worst-case latencies of the event-chains.

We distinguish two types of event-chains: event-chains that stay in the same Amalthea task; and event-chains that traverse different Amalthea tasks. Each kind requires a different formulation to calculate the end-to-end latencies.

### A. *Event-chains that traverse different Amalthea tasks*

Fig. 4 shows the MAST equivalent model of a simple event-chain that traverses three Amalthea tasks. This is the behavior that follows *EffectChain_2* and *EffectChain_3* event-chains in the challenge. Let us use the simple example shown in Fig. 4 to

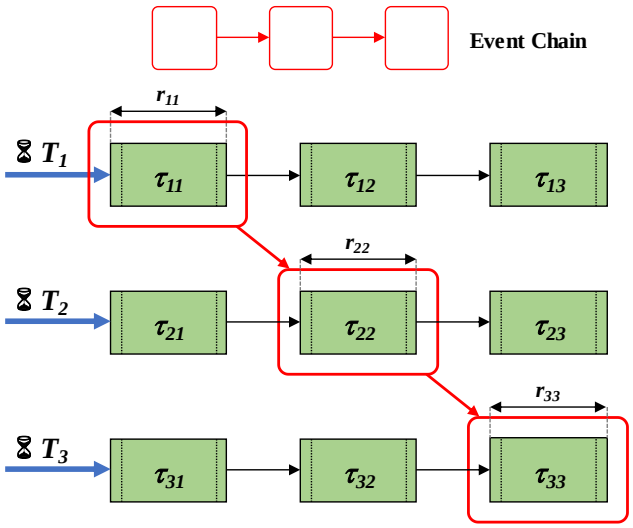


Fig. 4. Interpretation of an event-chain traversing different MAST end-to-end flows.

explain how to formulate the latencies for this kind of event-chain.

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 but the first one. 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 summing 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-chain shown in Fig. 4 is formalized as follows:

$$L = r_{11} + T_2 + r_{22} + T_3 + r_{33}$$

$$L^b = r_{11}^b + r_{22}^b + r_{33}^b$$

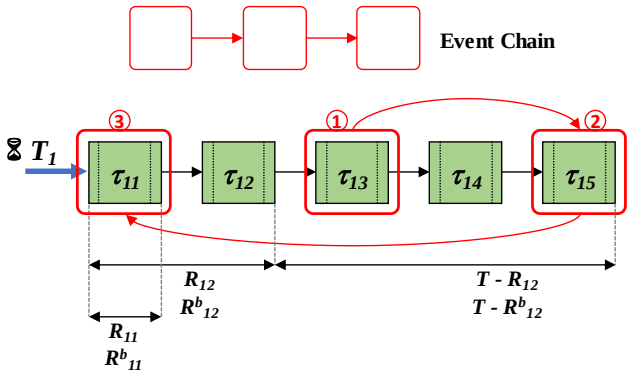


Fig. 5. Interpretation of an event-chain going backwards in the same MAST end-to-end flow.

### B. *Event-chains that go back in the same Amalthea task*

Fig. 5 shows the MAST equivalent model of a simple event-chain that traverses the same Amalthea task backwards. This is the behavior of *EffectChain_1* in the challenge. For this kind of event-chains it is trivial to see that to go backwards, the chain requires an additional activation of the Amalthea task.

Using the simple example shown in Fig. 5 as reference, for this type of event-chains the worst-case latency ($L$) occurs when the first label in the chain is read as soon as possible ($R_{12}^b$), 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 step $\tau_{11}$ ($R_{11}$). Since the end-to-end flow must finish before its next activation, the response time of step $\tau_{15}$ is irrelevant in this calculation. The total worst-case latency for this type of event-chain is formalized with the following equation:

$$L = (T_1 - R_{12}^b) + R_{11}$$

Likewise, the best-case latency ($L^b$) of the event-chain occurs when the first label is read as late as possible ($R_{12}$) and step $\tau_{11}$ finishes as soon as possible ($R_{11}^b$). The best-case latency for these kind of event-chains can be calculated with the following formula:

$$L^b = (T_1 - R_{12}) + R_{11}^b$$

## V. Evaluation

To transform the provided Amalthea model to MAST we developed an *ad-hoc* tool written in Java, consisting on less than 400 lines of code. This tool reads the challenge model using the Eclipse EMF framework [9], and builds an equivalent MAST model piece by piece using the interpretations described in Section III. The transformation of the given Amalthea model to MAST takes approximately 10 minutes, most of which are spent by the EMF framework loading the Amalthea model. The generated MAST model has approximately 23000 lines.

We proceed to solve the questions raised in the challenge, that is, to calculate end-to-end latencies that are as tight as possible. The challenge doesn't explicitly specify which are the end-to-end latencies that must be calculated. We provide end-to-end latencies for the Amalthea tasks (since they all have timing requirements), and for the event-chains described in the model. The analysis technique used has been the Offset-Based Analysis with Precedence Relationships [5]. This is the less pessimistic technique for end-to-end flows that only traverse one processor. The analysis tool takes from 1 to 5 minutes to execute, depending on the utilization of the system. The calculations of the slacks took up to 2 hours, since they involve iterative executions of the analysis tool.

In a first attempt to get analytical worst-case latencies, we used the upper bounds of the number of instructions of the runnables as the WCET of the MAST steps. The total utilization of that system goes above 100%. Using response time analysis in such situation automatically yields unbounded (infinite) worst-case response times. While utilizations over 100% can be handled by other techniques (e.g., simulators), they are not appropriate when applying response time analysis. After knowing that all upper-bounds in the original Amalthea model

can never occur at the same time, and not having the realistic models for each relevant real-time situation, we decided to consider two scenarios: Scn-ACET, and Scn-WCET.

In Scn-ACET, the worst-case execution times of the steps are calculated using the mean value of the number of instructions of the runnables. In Scn-WCET the worst-case execution times of the steps are calculated with the upper bound of the number of instructions (as described in Section III). In both scenarios we calculate latencies for different CPU clock frequencies, from the default 200Mhz and above (233Mhz, 266Mhz, etc.), until the timing requirements in the system are met. We essayed common CPU frequencies only. Additionally, for each analyzed case, we also calculate the system slack, and the slack of each core.

### A. Ignoring Memory Accesses

Table I shows the results when memory accesses are ignored. Shadowed cells indicate tasks that don't meet their deadlines. We can see that for Scn-ACET, 200Mhz is enough to make the system schedulable, with a system slack of 9.77%. If the clock frequency is increased to 233Mhz, system slack increases to 27.73%. For Scn-WCET, schedulability is achieved at 300 Mhz, with a system slack of 8.98%. If we observe the slack in each core, we can see that CORE1 is always the most constrained (lowest positive slack). This is to be expected, as this core has higher utilization among all cores.

### B. Adding Memory Accesses, using Global Memory Only

We repeat the process, but this time considering the memory accesses. As a reminder, the memory accesses are modelled as additional WCET of the steps, considering the worst-case cost of accessing each label. The results are shown in Table II. As can be expected, the core utilizations now increase compared to the case without memory accesses (Table I). The increase in utilization is between 3% and 12%, depending on the core. As a consequence, there is a system-wide increase in latencies too.

In this situation, Scn-ACET is not schedulable at 200 Mhz (Angle_Sync task misses its deadline in its worst-case). In this scenario, schedulability is achieved at 233 Mhz, with a system slack of 13.67%. On the other hand, Scn-WCET is schedulable at 300 Mhz, although with a marginal system slack of just 0.78%. At 333 Mhz, this system slack increases to 11.72%.

### C. Re-mapping Labels

The final question of the challenge asks for an optimization of the label-to-memory mapping to minimize the latencies. MAST does not provide a model for mapping memories, so we propose a reasonable solution. We identify that the majority of the labels are only accessed from a single core. As a first step, we map those labels into their local memories. Now the problem is reduced to determining where to map the labels shared by more than one core.

TABLE I. END-TO-END LATENCIES (MILLISECONDS.) AND SLACKS (%), **IGNORING MEMORY ACCESSES.**

| | **Scn-ACET** | | **Scn-WCET** | | | |
|---|---|---|---|---|---|---|
| | ***200 Mhz*** | ***233 Mhz*** | ***200 Mhz*** | ***300 Mhz*** | ***333 Mhz*** | ***D*** |
| **CORE0 Util. (%)** | 71.47 | 61.35 | 97.02 | 64.68 | 58.27 | |
| **CORE1 Util. (%)** | 88.38 | 75.86 | 133.57 | 89.05 | 80.22 | |
| **CORE2 Util. (%)** | 71.36 | 61.26 | 106.85 | 71.24 | 64.18 | |
| **CORE3 Util. (%)** | 77.19 | 66.25 | 117.94 | 78.62 | 70.83 | |
| **System Slack (%)** | 9.77 | 27.73 | -27.34 | 8.98 | 21.09 | |
| **CORE0 Slack (%)** | 31.08 | 52.89 | -98.44 | 45.12 | 60.52 | |
| **CORE1 Slack (%)** | 10.29 | 28.46 | -98.44 | 9.35 | 21.2 | |
| **CORE2 Slack (%)** | 40.37 | 63.58 | -98.44 | 40.37 | 55.66 | |
| **CORE3 Slack (%)** | 29.1 | 50.21 | -98.44 | 26.56 | 40.37 | |
| **Angle_Sync** | 5.54 | 3.86 | ∞ | 5.59 | 4.58 | **6.66** |
| **ISR_1** | 0.03 | 0.02 | ∞ | 0.02 | 0.02 | **9.5** |
| **ISR_10** | 0.02 | 0.02 | ∞ | 0.02 | 0.02 | **0.7** |
| **ISR_11** | 1.45 | 1.23 | ∞ | 1.29 | 1.16 | **5** |
| **ISR_2** | 0.04 | 0.03 | ∞ | 0.04 | 0.03 | **9.5** |
| **ISR_3** | 0.06 | 0.05 | ∞ | 0.05 | 0.05 | **9.5** |
| **ISR_4** | 0.50 | 0.43 | ∞ | 0.46 | 0.41 | **1.5** |
| **ISR_5** | 0.21 | 0.18 | ∞ | 0.19 | 0.17 | **0.9** |
| **ISR_6** | 0.23 | 0.20 | ∞ | 0.21 | 0.19 | **1.1** |
| **ISR_7** | 1.21 | 0.86 | ∞ | 0.90 | 0.81 | **4.9** |
| **ISR_8** | 0.75 | 0.63 | ∞ | 0.66 | 0.59 | **1.7** |
| **ISR_9** | 2.46 | 1.48 | ∞ | 2.20 | 1.39 | **6** |
| **Task_1000ms** | 31.18 | 17.63 | ∞ | 31.14 | 18.63 | **1000** |
| **Task_100ms** | 31.01 | 17.48 | ∞ | 30.97 | 18.47 | **100** |
| **Task_10ms** | 7.72 | 6.62 | ∞ | 7.86 | 7.08 | **10** |
| **Task_1ms** | 0.52 | 0.45 | ∞ | 0.51 | 0.46 | **1** |
| **Task_200ms** | 31.09 | 17.55 | ∞ | 31.05 | 18.55 | **200** |
| **Task_20ms** | 9.55 | 7.95 | ∞ | 9.78 | 8.81 | **20** |
| **Task_2ms** | 0.29 | 0.25 | ∞ | 0.27 | 0.24 | **2** |
| **Task_50ms** | 12.77 | 9.91 | ∞ | 12.99 | 11.46 | **50** |
| **Task_5ms** | 0.93 | 0.80 | ∞ | 0.89 | 0.80 | **5** |
| **EffectChain_1 (*L*)** | 12.63 | 12.25 | ∞ | 12.67 | 12.40 | |
| **EffectChain_2 (*L*)** | 25.23 | 23.10 | ∞ | 25.44 | 23.86 | |
| **EffectChain_3 (*L*)** | 63.38 | 60.72 | ∞ | 63.85 | 62.44 | |

TABLE II. END-TO-END LATENCIES (MILLISECONDS.) AND SLACKS (%), **INCLUDING MEMORY ACCESSES, USING GLOBAL MEMORY ONLY**

| | **Scn-ACET** | | **Scn-WCET** | | | |
|---|---|---|---|---|---|---|
| | ***200 Mhz*** | ***233 Mhz*** | ***200 Mhz*** | ***300 Mhz*** | ***333 Mhz*** | ***D*** |
| **CORE0 Util. (%)** | 73.75 | 63.31 | 99.30 | 66.20 | 59.64 | |
| **CORE1 Util. (%)** | 99.21 | 85.16 | 144.41 | 96.27 | 86.73 | |
| **CORE2 Util. (%)** | 76.40 | 65.58 | 111.89 | 74.59 | 67.20 | |
| **CORE3 Util. (%)** | 86.10 | 73.91 | 126.85 | 84.57 | 76.19 | |
| **System Slack (%)** | -2.34 | 13.67 | -32.81 | 0.78 | 11.72 | |
| **CORE0 Slack (%)** | -98.44 | 48.47 | -98.44 | 41.92 | 57.57 | |
| **CORE1 Slack (%)** | -1.92 | 14.20 | -98.44 | 1.18 | 12.21 | |
| **CORE2 Slack (%)** | -98.44 | 52.89 | -98.44 | 33.80 | 48.47 | |
| **CORE3 Slack (%)** | -98.44 | 34.50 | -98.44 | 17.87 | 30.41 | |
| **Angle_Sync** | 6.95 | 4.85 | ∞ | 6.60 | 4.96 | **6.66** |
| **ISR_1** | 0.03 | 0.03 | ∞ | 0.03 | 0.02 | **9.5** |
| **ISR_10** | 0.03 | 0.02 | ∞ | 0.02 | 0.02 | **0.7** |
| **ISR_11** | 2.26 | 1.27 | ∞ | 1.32 | 1.19 | **5** |
| **ISR_2** | 0.05 | 0.04 | ∞ | 0.04 | 0.04 | **9.5** |
| **ISR_3** | 0.07 | 0.06 | ∞ | 0.06 | 0.05 | **9.5** |
| **ISR_4** | 0.52 | 0.45 | ∞ | 0.47 | 0.42 | **1.5** |
| **ISR_5** | 0.22 | 0.19 | ∞ | 0.20 | 0.18 | **0.9** |
| **ISR_6** | 0.24 | 0.21 | ∞ | 0.22 | 0.20 | **1.1** |
| **ISR_7** | 1.25 | 0.89 | ∞ | 1.09 | 0.83 | **4.9** |
| **ISR_8** | 0.78 | 0.65 | ∞ | 0.67 | 0.61 | **1.7** |
| **ISR_9** | 2.53 | 2.15 | ∞ | 2.27 | 1.44 | **6** |
| **Task_1000ms** | 33.91 | 19.32 | ∞ | 33.03 | 19.64 | **1000** |
| **Task_100ms** | 33.55 | 19.02 | ∞ | 32.74 | 19.37 | **100** |
| **Task_10ms** | 8.61 | 7.39 | ∞ | 8.45 | 7.62 | **10** |
| **Task_1ms** | 0.58 | 0.50 | ∞ | 0.54 | 0.49 | **1** |
| **Task_200ms** | 33.71 | 19.15 | ∞ | 32.87 | 19.49 | **200** |
| **Task_20ms** | 11.21 | 8.79 | ∞ | 11.15 | 9.22 | **20** |
| **Task_2ms** | 0.32 | 0.27 | ∞ | 0.29 | 0.26 | **2** |
| **Task_50ms** | 13.63 | 11.42 | ∞ | 13.57 | 11.96 | **50** |
| **Task_5ms** | 0.97 | 0.84 | ∞ | 0.92 | 0.83 | **5** |
| **EffectChain_1 (*L*)** | 12.93 | 12.52 | ∞ | 12.87 | 12.59 | |
| **EffectChain_2 (*L*)** | 26.17 | 23.89 | ∞ | 26.07 | 24.67 | |
| **EffectChain_3 (*L*)** | 64.20 | 62.20 | ∞ | 64.40 | 62.91 | |

In our pessimistic approach for modeling the memory accesses, even if just one label in the local memory is accessed from a non-local core, every label in that local memory would be impacted. For example, consider a local memory with labels that are accessed from two cores: the local core and a non-local core. In this case, and regardless of from which core the memory is accessed, the worst-case cost assumes that both cores are accessing the memory at the same time, and thus that cost for reading or writing any of its labels would be 1 cycle + 9 cycles = 10 cycles.

To preserve the advantage of local memory accesses, we map into global memory every label shared among different cores. Therefore, local labels are assured to be accessed without contention (1 cycle access only), and the worst-case cost for shared labels is modelled as in Section III; that is, assuming that all cores are accessing global memory at the same time (a cost of 4*9 cycles for each label access). Table III shows the slacks and latencies obtained using this mapping, which confirms that the new mapping improves the results. It is also worth noting that with this new mapping, the results are closer to the case ignoring memory accesses (Table I), than to the case in which all labels are mapped to the global memory (Table II).

TABLE III. END-TO-END LATENCIES (MILLISECONDS.) AND SLACKS (%), **RE-MAPPING LABELS TO LOCAL AND GLOBAL MEMORIES**

| | **Scn-ACET** | | **Scn-WCET** | | | |
|---|---|---|---|---|---|---|
| | ***200 Mhz*** | ***233 Mhz*** | ***200 Mhz*** | ***300 Mhz*** | ***333 Mhz*** | ***D*** |
| **CORE0 Util. (%)** | 71.98 | 61.78 | 97.53 | 65.02 | 58.57 | |
| **CORE1 Util. (%)** | 92.14 | 79.09 | 137.33 | 91.56 | 82.48 | |
| **CORE2 Util. (%)** | 72.28 | 62.04 | 107.77 | 71.84 | 64.72 | |
| **CORE3 Util. (%)** | 79.85 | 68.54 | 120.6 | 80.4 | 72.43 | |
| **System Slack (%)** | 5.08 | 22.66 | -29.3 | 5.86 | 17.58 | |
| **CORE0 Slack (%)** | 30.41 | 51.98 | -98.44 | 44.31 | 60.52 | |
| **CORE1 Slack (%)** | 5.75 | 22.94 | -98.44 | 6.19 | 17.87 | |
| **CORE2 Slack (%)** | 38.11 | 61.52 | -98.44 | 38.86 | 54.72 | |
| **CORE3 Slack (%)** | 24.72 | 45.12 | -98.44 | 24.12 | 37.38 | |
| **Angle_Sync** | 5.78 | 4.50 | ∞ | 5.75 | 4.71 | **6.66** |
| **ISR_1** | 0.03 | 0.02 | ∞ | 0.02 | 0.02 | **9.5** |
| **ISR_10** | 0.02 | 0.02 | ∞ | 0.02 | 0.02 | **0.7** |
| **ISR_11** | 1.47 | 1.24 | ∞ | 1.30 | 1.17 | **5** |
| **ISR_2** | 0.04 | 0.03 | ∞ | 0.04 | 0.03 | **9.5** |
| **ISR_3** | 0.06 | 0.05 | ∞ | 0.05 | 0.05 | **9.5** |
| **ISR_4** | 0.51 | 0.44 | ∞ | 0.46 | 0.41 | **1.5** |
| **ISR_5** | 0.21 | 0.18 | ∞ | 0.19 | 0.17 | **0.9** |
| **ISR_6** | 0.23 | 0.20 | ∞ | 0.21 | 0.19 | **1.1** |
| **ISR_7** | 1.22 | 0.87 | ∞ | 1.07 | 0.81 | **4.9** |
| **ISR_8** | 0.76 | 0.63 | ∞ | 0.66 | 0.60 | **1.7** |
| **ISR_9** | 2.47 | 1.49 | ∞ | 2.23 | 1.39 | **6** |
| **Task_1000ms** | 31.63 | 17.89 | ∞ | 31.43 | 18.81 | **1000** |
| **Task_100ms** | 31.42 | 17.71 | ∞ | 31.24 | 18.64 | **100** |
| **Task_10ms** | 7.98 | 6.85 | ∞ | 8.04 | 7.24 | **10** |
| **Task_1ms** | 0.54 | 0.47 | ∞ | 0.52 | 0.47 | **1** |
| **Task_200ms** | 31.52 | 17.80 | ∞ | 31.34 | 18.73 | **200** |
| **Task_20ms** | 9.68 | 8.31 | ∞ | 9.86 | 8.88 | **20** |
| **Task_2ms** | 0.30 | 0.25 | ∞ | 0.27 | 0.24 | **2** |
| **Task_50ms** | 12.93 | 10.84 | ∞ | 13.10 | 11.56 | **50** |
| **Task_5ms** | 0.94 | 0.80 | ∞ | 0.90 | 0.81 | **5** |
| **EffectChain_1 (*L*)** | 12.71 | 12.33 | ∞ | 12.73 | 12.46 | |
| **EffectChain_2 (*L*)** | 25.40 | 23.25 | ∞ | 25.56 | 23.97 | |
| **EffectChain_3 (*L*)** | 63.53 | 60.83 | ∞ | 63.95 | 62.52 | |

## VI. CONCLUSIONS

This paper provides general guidelines to transform an Amalthea timing model into a MAST equivalent model that can be used in the MAST Analysis Tool Suite. Using them, response time analysis has been applied to calculate worst case latencies of tasks in a complex engine management system.

To understand the Amalthea model, we relied on the documentation of the tool [2], and the document describing the challenge [1]. While the basics of the model (e.g., tasks and runnables) can be easily understood with these materials, special elements of the model such as the event-chains required additional inquiries in the workshop forum. The total amount of time needed to completely digest the model can be approximated to about 12-14 hours divided in several days. Once the model was understood, the process of building the Amalthea to MAST transformation in Java required approximately 5 man-hours to a person familiar with MAST and EMF. The workspace used in this paper can be downloaded from [10].

The paper answers the three main questions of the challenge, (1) providing latencies when memory accesses are ignored, (2) providing latencies when all labels are mapped to the global memory, and (3) finding a new optimized mapping. Safer CPU frequencies as well as indicators of the most loaded tasks and cores in the system are provided. The main weakness we identify in our proposal is its pessimism in the modelling of global memory accesses. It uses an upper bound that cannot occur in reality. This is done to overcome the limitations of MAST which does not currently model the memory and the blocking of the processor while the memory is accessed. These two shortcomings have flagged interesting developments that we will explore in the future.

## REFERENCES


[1] 2016 Formals Methods and 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/

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

[3] 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.

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

[5] 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.

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

[7] 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.

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

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

[10] Amalthea workspace used for this solution: www.istr.unican.es/members/rivasjm/workspace_fmtv16_public.zip