
(GSoC 2019) CPU-GPU Response Time and Mapping Analysis

Release 1.0

May 04, 2020

1	Motivation	3
2	Contents	5
2.1	Intention	5
2.2	Contribution & Benefits for The Community	5
2.3	Milestone with The Goal of Each Phase	6
2.3.1	Phase 1 (May 27 - June 24)	6
2.3.2	Phase 2 (June 25 - July 22)	6
2.3.3	Phase 3 (July 23 - August 25)	7
2.4	Approached Theories	7
2.4.1	Basic RTA	7
2.4.2	End-to-End Latency	9
2.5	Class Tree with Implemented Methods	12
2.5.1	Key Classes	13
2.5.2	Supplementary Classes (Out of scope)	18
2.6	User Interface (APP4RTA)	19
2.6.1	APP4RTA Location	19
2.6.2	Search Amalthea	20
2.6.3	Navigate to The Amalthea Folder	21
2.6.4	Select & Open Amalthea	22
2.6.5	Amalthea Model Loaded	23
2.6.6	Integer Mapping	24
2.6.7	Assign Tasks to Processing Units	25
2.6.8	Measure Response Time	26
2.6.9	Task Chain Analysis	27
2.6.10	Change The Model	28
2.6.11	Single-core RTA	29
2.6.12	Single-core Task Chain Analysis	30
2.7	Future Work	30
2.7.1	1. Reaction Update	31
2.7.2	2. Blocking	31
2.7.3	3. Scheduling mode: EDF	31
2.7.4	4. Read & Write latency setting feature	31
2.7.5	5. Data Age metrics should be organized	31
2.8	Repositories	31
2.8.1	Eclipse Contribution Tagged Repo	31

2.8.2	ReadTheDocs Repo	31
2.9	Reference	32
2.10	Contact	32



Google Summer of Code

CHAPTER 1

Motivation

Through one of the subjects in my Master's course, I carried on a project analyzing metrics of Software Models and visualizing it in APP4MC.

It was quite challenging as I was not very familiar with the Amalthea model and its APP4MC platform at first. But soon I was able to understand the concepts and started enjoying it. The project resulted in completing an application delivering performance and reliability metrics of a given Software Model. This is basically my motivation for participating in this Eclipse GSoC project, "CPU-GPU Response Time and Mapping Analysis".

Since the topic's ultimate goal is to achieve systems' real-time determinism for modern HPC (High Performance Computing) applications, analyzing response times is essential and my basic knowledge in regard to, e.g., timing constraints or end-to-end event chain latency values according to the different communication paradigms (direct, implicit, LET: Logical Execution Time) which I obtained through my Master's study were very helpful for me in order to apply for and luckily realize this project.

Now that the industry's interest has moved on to "Heterogeneous Systems", I do hope that my GSoC work would be helpful for other researchers in this regard and make a contribution to the further development of the platform.

Ki, Junhyung

2.1 Intention

The current APP4MC APIs provide several methods for getting execution time for a task, a runnable or ticks (pure computation) through the util package.

However, APIs for response time analysis do not exist yet. The reason why is that response time analysis results highly vary depending on the analyzed model properties such as the scheduling, the mapping, and others.

Since the trends are evolving from homogeneous to heterogeneous platforms, the analysis methodologies have become much more sophisticated. A generic form of CPU response time analysis, which can be used for different mapping models with different types of processing units (e.g., GPU), is though reasonable across modern analysis techniques.

Additionally, this project also aims to offer end-to-end event-chain latency analyses that incorporate a distinct concepts such as reaction & age which will be outlined in this documentation. Such analyses are intended to help users to analyze how much time would be taken for some data to be propagated from the beginning to the end of a given chain of tasks.

2.2 Contribution & Benefits for The Community

In this project, a [standardized response time analysis methodology](#) (Mathai Joseph and Paritosh Pandya, 1986) is used. Not only this, but a class, `CPURta` which can be used with various implementations (e.g. a Genetic Mapping Algorithm), is also provided.

Since a heterogeneous platform requires different analysis methodologies for processing units, a class that has a built-in response-time calculation algorithm is very helpful and makes the entire developing circle quicker.

Another class, `RTARuntimeUtil` supports the `CPURta` class by providing several ways to calculate the execution time of a task. The methodology for deriving execution time changes depending on the execution case (e.g., Worst Case, Best Case, Average Case), the offloading mechanism (e.g., Synchronous, Asynchronous), and the mapping model. This class can be modified and reused for other models under analysis simply by adjusting a single method which takes care of memory accessing time (because memory accessing time can be different according to the target hardware).

Furthermore, this GSoC project provides a small GUI implementation, which visually describes the mapping model with information about schedulability, the corresponding response times for each task, and E2E latency analysis results (E2ELatency) according to each task chain.

2.3 Milestone with The Goal of Each Phase

2.3.1 Phase 1 (May 27 - June 24)

1. Structuring classes based on the abstraction layers (Top: End-to-End latency / Mid Layer: Response Time / Low Layer: Task & Runnable Execution Time)

Layer	Responsibility
Top	End-to-End Latency
Mid	Task Response Time
Low	Task & Runnable Execution Time

2. Developing task and runnable level execution time methods with taking memory access cost and offloading mechanisms into account
3. Testing
4. Documenting

The main focus of phase 1 is to implement the basis framework and map each and every functionality to the classes. In this way, the entire system becomes organized which eases refactoring and debugging.

2.3.2 Phase 2 (June 25 - July 22)

1. Developing interfaces between classes
2. Implementation of response time analysis algorithms according to different communication paradigms, i.e., direct and implicit communication)
3. Structuring and developing basic user interface class
4. Testing
5. Documenting

The main focus of phase 2 is to provide a stable response time method which can be used for several models under various configuration settings.

2.3.3 Phase 3 (July 23 - August 25)

Refine Previous Phase and E2E Latency Foundation (IC, LET) / Documenting

1. Implementation of E2E latency analysis methodologies according to the concepts such as age, reaction, and propagation under different communication paradigms such as direct, implicit, and LET = Logical Execution Time.
2. Extend and finalize the UI part
3. Testing
4. Final documenting (Through Sphinx & readthedocs)

The main focus of phase 3 is to implement newly defined concepts of end-to-end latency methodologies in line with the implicit and LET communication paradigms. As a consequence, users gain much more task chain metrics in addition to data propagation only.

Moreover, by using the provided GUI, user can investigate mapping scenarios and analyze response times & E2E latency metrics without diving into java implementations.

2.4 Approached Theories

2.4.1 Basic RTA

- Table of Notation for **Basic RTA**

Description	Symbol
Task	i
WC Response time	R_i^+
WC Execution time	C_i^+
Period	T_i
Frequency in Hz	f_m
Latency	L
Read Latency	$L_{\uparrow m \rightarrow l}$
Write Latency	$L_{\downarrow m \rightarrow l}$
Read labels	\mathcal{R}_i
Written Labels	\mathcal{W}_i
Label	\mathcal{L}
Label Size	\mathcal{S}

Memory Access Cost

Memory access time is different depending on the target hardware. In this project, the memory access time is defined based on NVIDIA-TX2 platform. The equation for deriving this is referenced the WATERS19 projects namely [CPU-GPU Response Time and Mapping Analysis for High-Performance Automotive Systems](#)

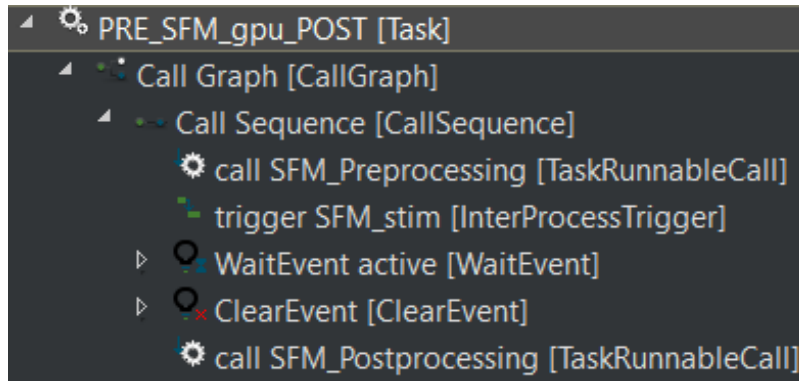
$$L_{a,i} = \sum_{x \in \mathcal{R}_i} \left(\left\lceil \frac{S_x}{64} \right\rceil \right) \cdot \frac{L_{\uparrow m \rightarrow l}}{f_m} + \sum_{y \in \mathcal{W}_i} \left(\left\lceil \frac{S_y}{64} \right\rceil \right) \cdot \frac{L_{\downarrow m \rightarrow l}}{f_m}$$

Here, the constant 64 is used as the baseline derived from the WATERS19 challenge description. ls denotes the label size and rl and wl define given read label and write label latencies specified in the given AMALTHEA model.

To find relevant methods, see [CPU Task Execution Time](#).

Synchronous & Asynchronous Mechanism

In the provided AMALTHEA WATERS19 model, some of the tasks that are mapped to CPU trigger tasks mapped to GPU. In this case, the execution or response time can be different according to the offloading mechanism.



- **Synchronous**

The triggering task triggers its target GPU task when it reaches `InterProcessTrigger` and actively waits until it receives the triggered task's result after the response from the triggered GPU task. Then it finishes the remaining job.

- **Asynchronous**

The triggering task triggers its target GPU task when it reaches `InterProcessTrigger` and passively waits for the response from the triggered GPU task and finishes the remaining job. During the passive waiting phase, other lower priority tasks can execute on the processor. The asynchronous methodology described here can be modified according to the user's interpretation.

This concept is used in two of the four execution cases introduced by a method, [CPU Task Execution Time](#).

Worst-case Response Time

The response time analysis approach implemented here is not only designed for Multi-core Systems but also for Heterogeneous Systems. Basically, the following classical response time analysis equation is used.

$$R_i^+ = C_i^+ + \sum_{j \in hp(i)} \left\lceil \frac{R_{i-1}^+}{T_j} \right\rceil C_j^+$$

The equation is based on RMS (Rate Monotonic Scheduling) which means that static priorities are assigned to tasks according to their period. A task with the shorter period results in a higher task priority. Here, R_i denotes the response time of task τ_i and $hp(i)$ is the set of tasks indexes (j) which have a priority higher than task i .

To find relevant methods, see [Response Time Sum](#).

Best-case Response Time

Unlike the worst-case analysis, considering all tasks arriving at the same point of time does not work since every new task instance can have a different response time after the first iteration as long as it is not the highest priority task. Hence, the response time analysis which takes this point into account is required.

$$R_i^{n+1} = C_i^- + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n - J_j - T_j}{T_j} \right\rceil_0 C_j^- \text{ for } n = 0, 1, 2, \dots$$

with $R_i^0 = R_i^+$
where $\lceil x \rceil_0 = \max(0, \lceil x \rceil)$

The equation is based on RMS (Rate Monotonic Scheduling) which means that static priorities are assigned to tasks according to their period. A task with the shorter period results in a higher task priority.

2.4.2 End-to-End Latency

The approach and its equations used here are referenced from a yet-unpublished paper, “Model-based Task Chain Latency and Blocking Analysis for Automotive Software” by the same authors who published [CPU-GPU Response Time and Mapping Analysis for High-Performance Automotive Systems](#).

- Table of Notation for **End-to-End Latency**

Symbol	Description
Task	τ
Response time	R
Execution time	C
Period	T
Task chain	γ
Latency	δ
implicit communication	ι
LET communication	λ
Age latency	α
Reaction latency	ρ
Reaction update	v

Task Chain Reaction

The time between the task chain’s first task release to the earliest task response of the last task in the chain.

Task Chain Reaction (Implicit Communication Paradigm)

- **Best-case Task-Chain Reaction (Implicit)**

$$\delta_{\gamma,\rho,\iota}^- = R_{\gamma_0}^- + \sum_{j=1}^{|\gamma|-1} (R_j^- - ip(\tau_j)^-)$$

$$ip(\tau_j)^- = R_j^- - C_j^-$$

The best-case task chain reaction latency with Implicit communication can be calculated by summing up the first chain element's best-case response time and the rest of all task's best-case response times which are subtracted by each task's best-case initial pending time. Here, γ refers to a task chain, ρ corresponds the reaction latency, ι is Implicit communication paradigm, and $ip(\tau_j)^-$ stands for the best-case initial pending time of τ_j .

- **Worst-case Task-Chain Reaction (Implicit)**

$$\delta_{\gamma,\rho,\iota}^+ = T_{\gamma_0} + R_{\gamma_0}^+ + \sum_{j=1}^{|\gamma|-1} (R_j^+ + f(j))$$

$$f(j) = \begin{cases} T_j - ip(\tau_j)^- & \text{if } j \neq |\gamma| - 1 \\ -ip(\tau_j)^+ & \text{else} \end{cases}$$

$$ip(\tau_j)^+ = f(0, R_{(j-1) \in hp(j)}^+)$$

$$f(k, R) = \begin{cases} R & \text{if } (k == |hp(j-1)|) \vee (R < T_{hp(j-1)_k}) \\ f(k+1, R) & \text{else if } (R \% T_{hp(j-1)_k}) \neq 0 \\ f(0, R + C_{hp(j-1)_k}^+) & \text{else} \end{cases}$$

To find relevant methods, see [Task Chain Reaction \(Implicit Communication Paradigm\)](#).

- **Best-case Task-Chain Initial Reaction (Implicit)**

$$\delta_{\gamma,\rho_0,\iota}^- = \delta_{\gamma,\rho,\iota}^-$$

The best-case reaction is always equal to the best-case initial reaction of a task chain with Implicit communication.

- **Worst-case Task-Chain Initial Reaction (Implicit)**

$$\delta_{\gamma,\rho_0,\iota}^+ = R_{\gamma_0}^+ + \sum_{j=1}^{|\gamma|-1} (R_j^+ + f(j))$$

$$f(j) = \begin{cases} T_{j-1} - ip(\gamma_j)^+ & \text{if } (T_j \geq T_{j-1}) \wedge (|\gamma| - 1 < 3) \\ -ip(\gamma_j)^+ & \text{else if } (T_j \geq T_{j-1}) \wedge (j == |\gamma| - 1) \\ T_{j-1} + T_j - ip(\gamma_j)^- & \text{else if } (T_j \geq T_{j-1}) \wedge (j == 1) \\ T_j - ip(\gamma_j)^- & \text{else} \end{cases}$$

Task Chain Reaction (Logical Execution Time Communication Paradigm)

- **Best-case Task Chain Reaction (LET)**

$$\delta_{\gamma,\rho,\lambda}^- = \sum_{j=0}^{|\gamma|-1} T_j \text{ with } \tau_j \in \gamma$$

The best-case task chain reaction latency for LET communication is the sum of all task's periods within task chain γ .

- **Worst-case Task Chain Reaction (LET)**

$$\delta_{\gamma,\rho,\lambda}^+ = \sum_{j=0}^{|\gamma|-2} (2 \cdot T_j) + T_{|\gamma|-1}$$

To find relevant methods, see [Task Chain Reaction \(Logical Execution Time Communication Paradigm\)](#).

- **Best-case Task Chain Initial Reaction (LET)**

$$\delta_{\gamma, \rho_0, \lambda}^- = \delta_{\gamma, \rho, \lambda}^-$$

The best-case reaction is always the initial reaction of a task chain.

- **Worst-case Task Chain Initial Reaction (LET)**

$$\delta_{\gamma, \rho_0, \lambda}^+ = T_0 + \sum_{j=1}^{|\gamma|-1} (T_j + f(j))$$

$$f(j) = \begin{cases} T_{j-1} & \text{if } (T_j > T_{j-1}) \\ T_j & \text{else} \end{cases}$$

Task Chain Age

“The time a task chain result is initially available until the next task chain instance’s initial results are available. In other words, the task chain age latency is the maximal time a task chain’s results based on the same input persist in memory.”

- **Best-case Task Chain Age (Implicit)**

$$\delta_{\gamma, \alpha, \ell}^- = T_{\gamma_0} + \delta_{\gamma, \rho_0, \ell}^- - \delta_{\gamma, \rho, \ell}^+$$

- **Worst-case Task Chain Age (Implicit)**

$$\delta_{\gamma, \alpha, \ell}^+ = T_{\gamma_0} + \delta_{\gamma, \rho_0, \ell}^+ - \delta_{\gamma, \rho_0, \ell}^-$$

- **Worst-case Task Chain Age (LET)**

$$\delta_{\gamma, \alpha, \lambda}^- = T_{|\gamma|-1}$$

- **Worst-case Task Chain Age (LET)**

$$\delta_{\gamma, \alpha, \lambda}^+ = T_{\gamma_0} + \sum_{j=1}^{|\gamma|-1} f(j)$$

$$f(j) = \begin{cases} 2 \cdot T_j - f(j-1) & \text{if } (T_j == f(j-1)) \\ T_j - f(j-1) & \text{else if } (T_j > f(j-1)) \\ T_j \cdot \left\lceil \frac{f(j-1)}{T_j} \right\rceil - f(j-1) & \text{else if } (T_j < f(j-1)) \wedge \\ & (f(j-1) \% T_j \neq 0) \\ T_j \cdot \left(\frac{f(j-1)}{T_j} + 1 \right) - f(j-1) & \text{else} \end{cases}$$

$$\text{with } f(0) = T_0$$

Task Age

The Task Age refers to the time from a task instance’s result is available until the next instance’s result from the same task appears.

- **Best-case Task Age (Implicit)**

$$\delta_{j, \alpha}^- = (T_j - R_j^+) + R_j^- = T_j - R_j^+ + R_j^-$$

- **Worst-case Task Age (Implicit)**

$$\delta_{j,\alpha}^+ = (T_j - R_j^-) + R_j^+ = T_j - R_j^- + R_j^+$$

Data Age

It describes the longest time some data version persists in memory. This is independent of task chains and simply depends on the period of entities writing a particular label (i.e. data).

- **Best-case Data Age**

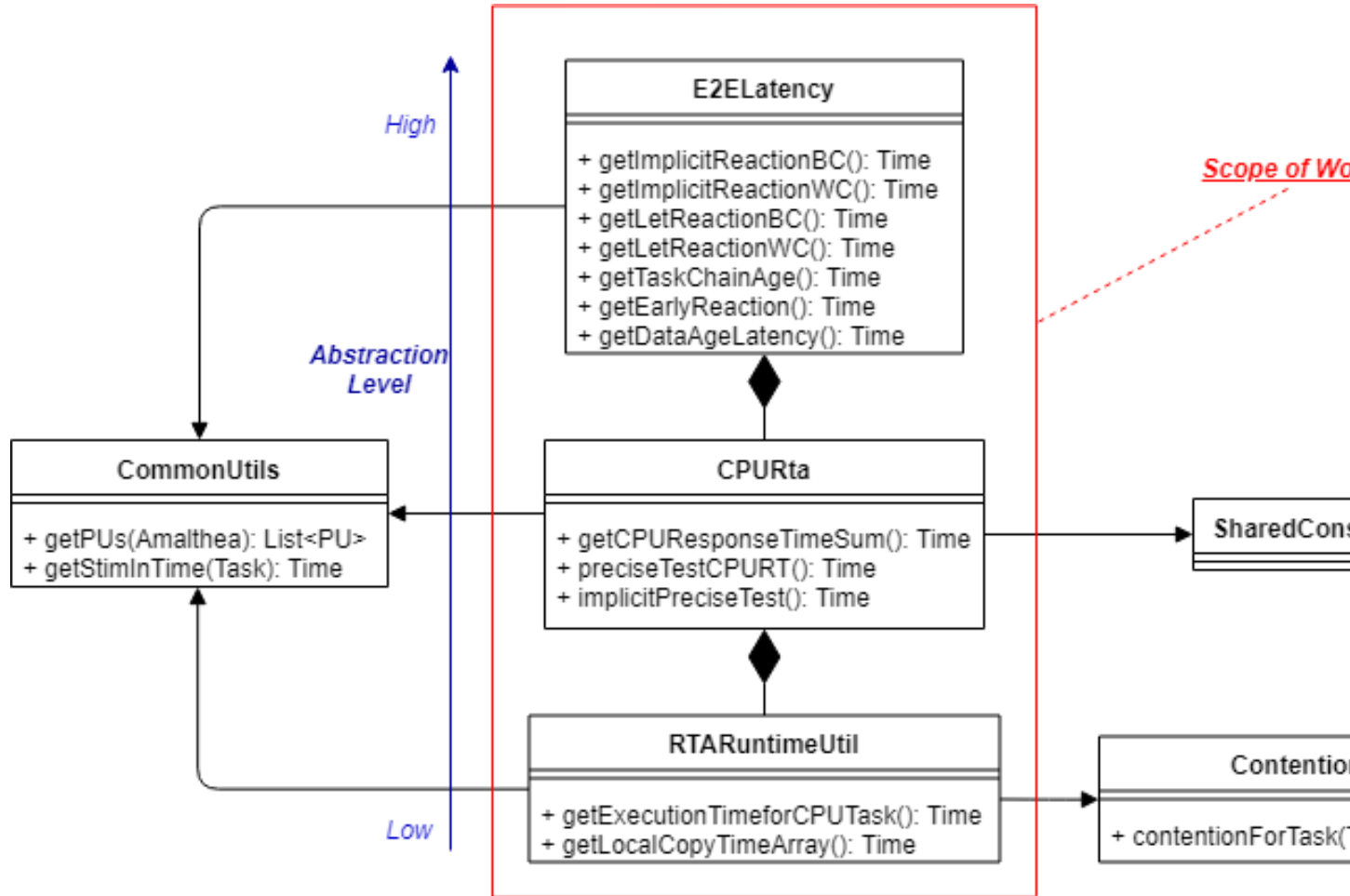
$$\delta_{l,\alpha}^- = \min_{\gamma_l} \delta_{\gamma_l,\alpha}^- \text{ with } \gamma_l \text{ being all tasks that access label } l.$$

- **Worst-case Data Age**

$$\delta_{l,\alpha}^+ = \max_{\gamma_l} \delta_{\gamma_l,\alpha}^+ \text{ with } \gamma_l \text{ being all tasks that access label } l.$$

To find relevant methods, see *Data Age*.

2.5 Class Tree with Implemented Methods



The above UML class diagram describes the project's implementation in a hierarchical way.

2.5.1 Key Classes

E2ELatency

The top layer takes care of the end-to-end latency calculation of the observed task-chain based on the analyzed response time from the CPURta class. It includes calculating E2E latency values according to the concepts stated in the theory part (e.g., Reaction, Age).

Task Chain Reaction (Implicit Communication Paradigm)

```
public Time getTCReactionBC(final EventChain ec, final ComParadigm paradigm, final  
    ↳ CPURta cpurta)
```

This method derives the given event-chain's best-case end-to-end latency based on the reaction concept for the direct and implicit communication paradigms.

Code Reference

```
public Time getTCReactionWC(final EventChain ec, final ComParadigm paradigm, final  
    ↳ CPURta cpurta)
```

This method derives the given event-chain's worst-case end-to-end latency value based on the reaction concept for the direct and implicit communication paradigms.

Code Reference

For the details, see *Task Chain Reaction (Implicit Communication Paradigm)* and features-e2elatency.

Task Chain Reaction (Logical Execution Time Communication Paradigm)

```
public Time getLetReactionBC(final EventChain ec, final CPURta cpurta)
```

This method derives the given event-chain's best-case end-to-end latency value based on the reaction concept for LET communication.

Code Reference

```
public Time getLetReactionWC(final EventChain ec, final CPURta cpurta)
```

This method derives the given event-chain's worst-case end-to-end latency based on the reaction concept for LET communication.

[Code Reference](#)

For the details, see *Task Chain Reaction (Logical Execution Time Communication Paradigm)* and features-e2elatency.

Task Chain Age

```
public Time getTaskChainAge(final EventChain ec, final TimeType executionCase, final ↵  
↵ComParadigm paradigm, final CPURta cpurta)
```

This method derives the given event-chain latency based on the age concept. By changing `TimeType executionCase` parameter, the latency in the best-case or the worst-case can be derived.

[Code Reference](#)

For the details, see *Task Chain Age* and features-e2elatency.

Task Chain Early Reaction

```
public Time getEarlyReaction(final EventChain ec, final TimeType executionCase, final ↵  
↵ComParadigm paradigm, final CPURta cpurta)
```

This is a method to be pre-executed for getting the reaction-update latency values. The best-case and worst-case early-reaction latency values should be derived first and then the reaction update latency can be calculated. By changing `TimeType executionCase` parameter, the latency in the best-case or the worst-case can be derived.

[Code Reference](#)

For the details, see early-reaction and features-e2elatency.

Data Age

```
public Time getDataAge(final Label label, final EventChain ec, final TimeType ↵  
↵executionCase, final ComParadigm paradigm, final CPURta cpurta)
```

This method derives the given label's age latency. If the passed event-chain does not contain the observed label, `null` is returned. By changing `TimeType executionCase` parameter, the latency in the best-case or the worst-case can be derived.

[Code Reference](#)

For the details, see [Data Age](#) and [features-e2elatenacy](#).

CPURta

The middle layer takes care of analyzing task response times. It is responsible for calculating response times according to the communication paradigm (Direct or Implicit communication paradigm).

Response Time Sum

```
public Time getCPUResponseTimeSum(final TimeType executionCase)
```

This method derives the sum of all the tasks' response times according to the given mapping model (which is described as an integer array). The method can be used as a metric to assess a mapping model.

[Code Reference](#)

Response Time (Direct Communication Paradigm)

```
public Time preciseTestCPURT(final Task task, final List<Task> taskList, final ↵  
↵TimeType executionCase, final ProcessingUnit pu)
```

This method derives the response time of the observed task according to the classic response time equation. The response time can be different depending on the passed taskList which is derived from the mapping model. Here, we are concerning response time for RMS (Rate Monotonic Scheduling). It means that a task with the shorter period obtains a higher priority. Before the taskList is passed to the method, it should be sorted in the order of shortest to longest and this job is done by `taskSorting(List<Task> taskList)` which is a private method.

[Code Reference](#)

Response Time (Implicit Communication Paradigm)

```
public Time implicitPreciseTest(final Task task, final List<Task> taskList, final ↵  
↵TimeType executionCase, final ProcessingUnit pu, final CPURta cpurta)
```

This method derives the response time of the task parameter according to the classic response time equation but in the implicit communication paradigm. In the implicit communication paradigm which is introduced by AUTOSAR. A task copies in its required data (labels) to its local memory at the beginning of its execution, computes in the local

memory and finally copies out the result to the shared memory. Due to these copy-in & copy-out costs, extra time must be added to the task's execution time which is done by `getLocalCopyTimeArray` (for the details, see [Local Copy Cost for the Implicit Communication Paradigm](#)) which is a method from the `RTARuntimeUtil` class. As a result, the task's execution time gets longer while its period should stay the same. Once the local-copy cost is taken into account, the remaining process is the same as [Response Time \(Direct Communication Paradigm\)](#)

Code Reference

For the details, see `response-time` and `features-rta`.

RTARuntimeUtil

The bottom layer takes care of task and runnable execution time. It is responsible for calculating memory access costs, execution ticks or execution needs, and computation time.

CPU Task Execution Time

```
public Time getExecutionTimeforCPUTask(final Task task, final ProcessingUnit pu,
    ↪ final TimeType executionCase, final CPURta cputa)
```

This method derives the execution time of the task parameter under one of the following cases:

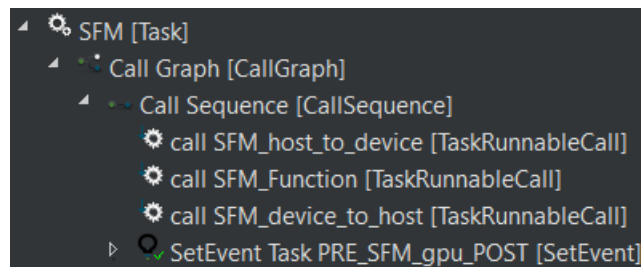
- The CPU task triggers a GPU task in the synchronous offloading mode
- The CPU task triggers a GPU task in the asynchronous offloading mode

(For the details, see [Synchronous & Asynchronous Mechanism](#).)

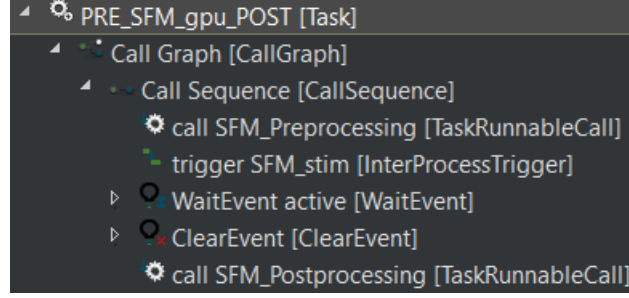
- The GPU task is mapped to a CPU

According to the WATERS challenge, a triggering task (`PRE_..._POST`) can be ignored if the triggered task is mapped to a CPU.

For example, the following Figure shows the `SFM` task which is mapped to the GPU by default.



If the task is mapped to CPU, the offloading runnables (`SFM_host_to_device`, `SFM_device_to_host`) which are in charge of offloading workload to GPU and copying back to CPU are obsolete.



Instead, the labels from runnables before (Pre-processing) & after (Post-processing) the InterProcessTrigger are considered. For the runnable, Pre-processing, read labels and read latency values are taken into account. For the runnable, Post-processing, write labels and write latency values are taken into account. This job is done by the private method `getExecutionTimeForGPUPTaskOnCPU()`.

- Task with only Ticks (pure computation)

When a CPU task without any triggering behavior is passed, only the execution time that corresponds to the task's ticks is considered.

[Code Reference for getExecutionTimeforCPUTask](#)

Except for the very last case (Task with only Ticks), the task execution time calculation always includes memory accessing costs. Calculating memory accessing costs is taken care of by methods such as `getExecutionTimeForRTARunnable`, `getRunnableMemoryAccessTime` which are defined as private.

[Code Reference for getExecutionTimeForRTARunnable](#) [Code Reference for getRunnableMemoryAccessTime](#)

For the details, see [Memory Access Cost](#).

Local Copy Cost for the Implicit Communication Paradigm

```
public Time[] getLocalCopyTimeArray(final Task task, final ProcessingUnit pu, final ↵
↵TimeType executionCase, final CPURta cputa)
```

As it is introduced in [Response Time \(Implicit Communication Paradigm\)](#), label copy-in and copy-out costs should be calculated and added to the total execution time of the target task.

The following equation from [End-To-End Latency Characterization of Implicit and LET Communication Models](#) is used to calculate these costs.

$$C_i^0 = \sum_{l \in I_i} \xi_l(x)$$

Where C_i^0 denotes the execution time of the runnable `tau_0`, I_i represents the inputs (read labels) of the considered task and $\xi_l(x)$ denotes the time it takes to access a shared label l from memory x .

$$C_i^{last} = \sum_{l \in O_i} \xi_l(x)$$

Where C_i^{last} denotes the execution time of the runnable `tau_last`, O_i represents the outputs (write labels) of the considered task and $\xi_l(x)$ denotes the time it takes to access a shared label l from memory x .

For the copy-in cost, only read labels should be taken into account. The copy-in cost time is stored on index 0 of the return array. This will later be considered as the execution time of the copy-in runnable which is added to the beginning of the task execution.

For the copy-in cost, only write labels should be taken into account. The copy-in cost time is stored on index 1 of the return array. This will later be considered as the execution time of the copy-out runnable which is added to the end of the task execution.

[Code Reference](#)

2.5.2 Supplementary Classes (Out of scope)

SharedConsts

This class is in charge of setting configuration variables. The user can set the offloading mechanism and the execution case (WC, AC, BC) by changing `synchronousOffloading` and `timeType` respectively. Also, all file paths for every Amalthea model can be saved as `String` type constants here so that the user can change the target Amalthea model by switching these constants.

CommonUtils

```
public static List<ProcessingUnit> getPUs(final Amalthea amalthea)
```

This method derives a list of processing units of the target Amalthea model. It places CPU type processing units in the front and that of GPU type in the tail (end) of the list.

[Code Reference](#)

```
public static Time getStimInTime(final Task t)
```

This method returns the periodic recurrence time of the target task. If the passed task is not a periodic task (e.g., GPU task), the recurrence time of a task which is periodic and triggers the target task is returned. Otherwise time 0 is returned.

[Code Reference](#)

Contention

```
public Time contentionForTask(final Task task)
```

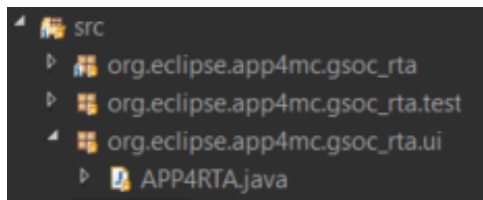
This method derives a memory contention time which represents the delay when more than one CPU core and/or the GPU is accessing memory at the same time.

Code Reference

For the details, see [Memory Contention Model](#).


2.6 User Interface (APP4RTA)

2.6.1 APP4RTA Location



Run `APP4RTA.java` in `org.eclipse.app4mc.gsoc_rta.ui` package.

2.6.2 Search Amalthea

 APP4RTA

Amalthea Model

Search Amalthea

Task Name

PU Num

EVENT CHAIN MODEL

Calculate

Reset

Reaction (Implicit)

Best-Case

Worst-Case

Initial Reaction (Implicit)

Best-Case

Worst-Case

Age (Implicit)

Best-Case

Worst-Case

Single-core Initial Reaction

Critical-Case

Best-Case

Worst-Case

Reaction (LET)

Best-Case

Worst-Case

Initial Reaction (LET)

Best-Case

Worst-Case

Age (LET)

Best-Case

Worst-Case

Single-core Worst-case Reaction

Critical-Case

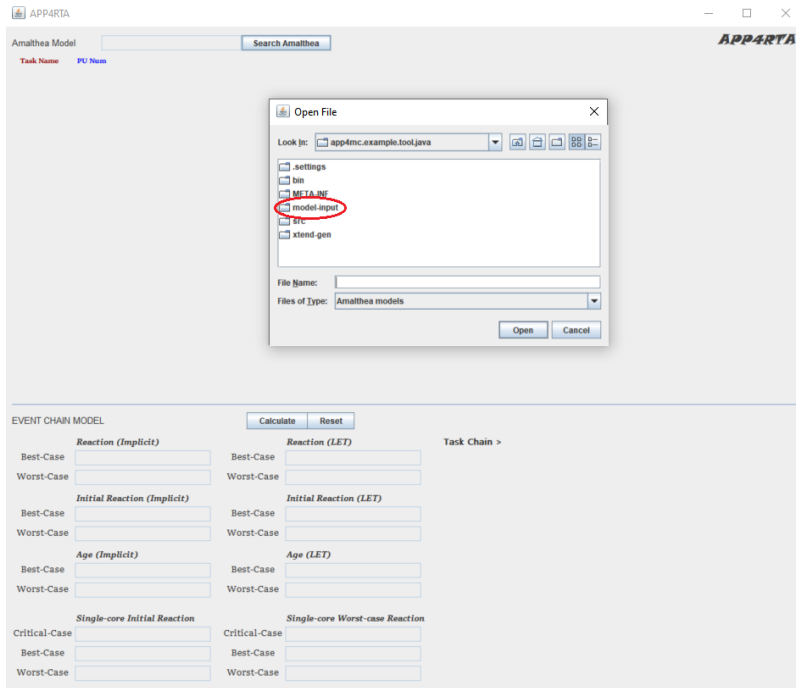
Best-Case

Worst-Case

Task Chain >

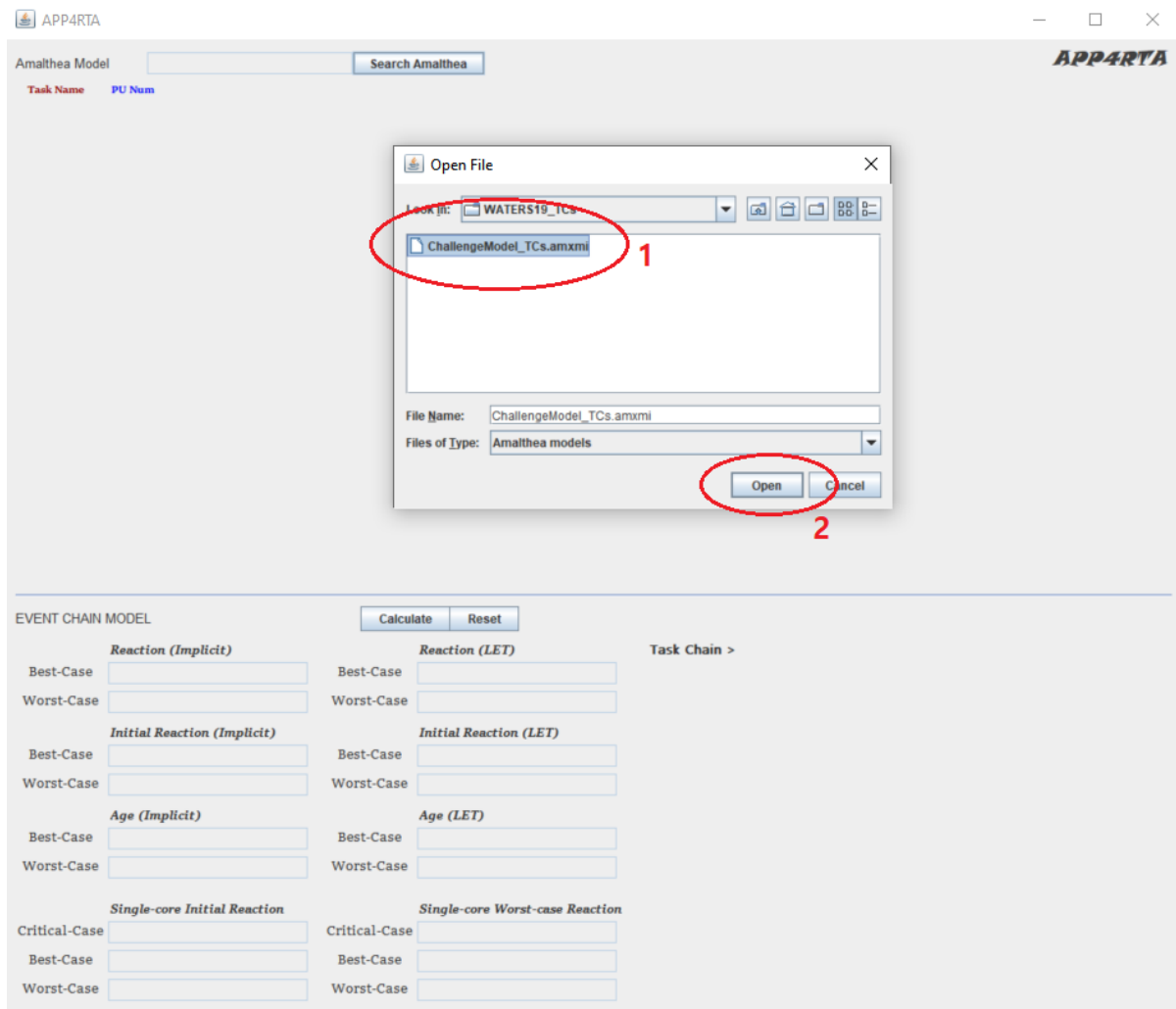
Based on the horizontal line on the middle, the upper part is for response time and mapping analysis and the lower part is for end-to-end event chain latency analysis. The first thing to do is deciding a target Amalthea model. Click the Search Amalthea button.

2.6.3 Navigate to The Amalthea Folder



Navigate to the folder where the target Amalthea model file is located.

2.6.4 Select & Open Amalthea



Select and open an Amalthea file. In this example, a multi-core Amalthea model is chosen.

2.6.5 Amalthea Model Loaded

APP4RTA

Amalthea Model ChallengeModel_TCs.amxml Search Amalthea

Task Name	PU Num	0: Denver	Response Time	1: Denver	Response Time	2: A57	Response Time	3: A57	Response Time
OS_Overhead	<input type="checkbox"/>								
Lidar_Grabber	<input type="checkbox"/>								
ASM	<input type="checkbox"/>								
CANbus_polling	<input type="checkbox"/>								
EKF	<input type="checkbox"/>								
Planner	<input type="checkbox"/>								
PRE_SFM_gpu...	<input type="checkbox"/>								
PRE_Localizati...	<input type="checkbox"/>								
PRE_Lane_det...	<input type="checkbox"/>								
PRE_Detection...	<input type="checkbox"/>								
SFM	<input type="checkbox"/>								
Localization	<input type="checkbox"/>								
Lane_detection	<input type="checkbox"/>								
Detection	<input type="checkbox"/>								

Default IA
Enter IA

☐ Synchronous
☐ Asynchronous

☐ Worst-Case
☐ Average-Case
☐ Best-Case

Calculate
Reset

Schedulability
Cumulated Memory-Access Cost
Cumulated Contention
Computation
Response Time Sum

0: Denver Response Time 1: Denver Response Time 2: A57 Response Time 3: A57 Response Time 4: A57 Response Time 5: A57 Response Time 6: GPU_def Response Time

1 2

EVENT CHAIN MODEL

Reaction (Implicit)		Reaction (LET)		Task Chain >	
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		
Initial Reaction (Implicit)		Initial Reaction (LET)			
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		
Age (Implicit)		Age (LET)			
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		
Single-core Initial Reaction		Single-core Worst-case Reaction			
Critical-Case	<input type="text"/>	Critical-Case	<input type="text"/>		
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		

After a model is loaded, it shows all the tasks (1) and processing units (2) that the selected model has.

2.6.6 Integer Mapping

APP4RTA

Amalthea Model: ChallengeModel_TCs.amxmi

Task Name	PU Num	Default IA	Enter IA	0: Denver	Response Time	1: Denver	Response Time	2: A57	Response Time	3: A57
OS_Overhead	4	Default IA	<input type="button" value="Enter IA"/>							
Lidar_Grabber	1									
DASM	1									
CANbus_polling	3	<input type="radio"/> Synchronous								
EKF	4	<input type="radio"/> Asynchronous								
Planner	0	<input type="radio"/> Worst-Case								
PRE_SFM_gpu...	3	<input type="radio"/> Average-Case								
PRE_Localizati...	3	<input type="radio"/> Best-Case								
PRE_Lane_det...	3	<input type="button" value="Calculate"/>								
PRE_Detection...	0	<input type="button" value="Reset"/>								
SFM	6									
Localization	2	Schedulability								
Lane_detection	5	Cumulated Memory-Access Cost								
Detection	6	Cumulated Contention								
		Computation								
		Response Time Sum								

EVENT CHAIN MODEL

Reaction (Implicit)		Reaction (LET)		Task Chain >	
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		
Initial Reaction (Implicit)		Initial Reaction (LET)			
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		
Age (Implicit)		Age (LET)			
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		
Single-core Initial Reaction		Single-core Worst-case Reaction			
Critical-Case	<input type="text"/>	Critical-Case	<input type="text"/>		
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		

When the Default IA (1) button is clicked, each task's box (2) is automatically filled with an integer number. This indicates that a task is about to be mapped to the corresponding identity number of processing unit. One can also write an integer number in each box manually. The Default IA means an integer array to map all the tasks to processing units and that is specifically designed to make the ChallengeModel_TCs.amxmi model schedulable. Therefore it is always possible that it does not serve for other multi-core models. However, the Default IA would only contain numbers of 0 when a single-core model is loaded.

2.6.7 Assign Tasks to Processing Units

APP4RTA

Amalthea Model: ChallengeModel_TCs.amxmi

Task Name	PU Num	Default IA	Enter IA	1
OS_Overhead	4	<input type="button" value="Default IA"/>	<input type="button" value="Enter IA"/>	
Lidar_Grabber	1			
DASM	1			
CANbus_polling	3	<input type="radio"/> Synchronous		
EKF	4	<input type="radio"/> Asynchronous		
Planner	0	<input type="radio"/> Worst-Case		
PRE_SFM_gpu...	3	<input type="radio"/> Average-Case		
PRE_Localizati...	3	<input type="radio"/> Best-Case		
PRE_Lane_det...	3	<input type="button" value="Calculate"/>		
PRE_Detection...	0	<input type="button" value="Reset"/>		
SFM	6			
Localization	2	<input type="text" value="Schedulability"/>		
Lane_detection	5	<input type="text" value="Cumulated Memory-Access Cost"/>		
Detection	6	<input type="text" value="Cumulated Contention"/>		
		<input type="text" value="Computation"/>		
		<input type="text" value="Response Time Sum"/>		

2

EVENT CHAIN MODEL

Reaction (Implicit)		Reaction (LET)		Task Chain >
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>	
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>	
Initial Reaction (Implicit)		Initial Reaction (LET)		
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>	
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>	
Age (Implicit)		Age (LET)		
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>	
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>	
Single-core Initial Reaction		Single-core Worst-case Reaction		
Critical-Case	<input type="text"/>	Critical-Case	<input type="text"/>	
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>	
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>	

When the Enter IA (1) button is clicked, each task is mapped to the corresponding processing unit (2). Since there are 7 processing units in the ChallengeModel_TCs.amxmi model, it shows 7 pairs of lists. The list on the left side of each pair is for listing names of the tasks that are mapped to the corresponding processing unit while one on the right side is for listing response times of the corresponding tasks.

2.6.8 Measure Response Time

APP4RTA

Amalthea Model: ChallengeModel_TCs.amxmi

Task Name	PU Num	Default IA	Enter IA	0: Denver	Response Time	1: Denver	Response Time	2: A57	Response Time	3: A57	Response Time
OS_Overhead	4	<input type="button" value="Default IA"/>	<input type="button" value="Enter IA"/>	Planner	13358534500	DASM	1302430000 p	Localization	392590097500	CANbus_polling	
Lidar_Grabber	1			PRE_Detectio	73565439500	Lidar_Grabber	18265272000			PRE_SFM_g	
DASM	1									PRE_Lane_d	
CANbus_polling	3	<input checked="" type="radio"/> Synchronous								PRE_Localiz	
EKF	4	<input type="radio"/> Asynchronous									
Planner	0	<input checked="" type="radio"/> Worst-Case									
PRE_SFM_gpu...	3	<input type="radio"/> Average-Case									
PRE_Localizati...	3	<input type="radio"/> Best-Case									
PRE_Lane_det...	3	<input type="button" value="Calculate"/>									
PRE_Detection...	0	<input type="button" value="Reset"/>									
SFM	6										
Localization	2										
Lane_detection	5										
Detection	6										

4: A57 Response Time: EKF 4788430000 p, OS_Overhead 73942150000

5: A57 Response Time: Lane_detectio 56045200000

6: GPU_def Response Time: SFM Detection 2000000000 p

5: 5361668000 ps
24795710000 ps
635075050500 ps
665232428500 ps

EVENT CHAIN MODEL

Reaction (Implicit)		Reaction (LET)		Task Chain >	
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		
Initial Reaction (Implicit)		Initial Reaction (LET)			
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		
Age (Implicit)		Age (LET)			
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		
Single-core Initial Reaction		Single-core Worst-case Reaction			
Critical-Case	<input type="text"/>	Critical-Case	<input type="text"/>		
Best-Case	<input type="text"/>	Best-Case	<input type="text"/>		
Worst-Case	<input type="text"/>	Worst-Case	<input type="text"/>		

- (1) Choose the offloading mode between Synchronous case and Asynchronous case.
- (2) Choose the execution case between Worst-, Average-, and Best-Case.
- (3) By clicking the Calculate button, each task's response time is calculated and printed on the right list of each list pair
- (4) All analysis results appear in (5) which include: Schedulability, Cumulated Memory-Access Cost, Cumulated Contention, Computation, and Response Time Sum.

2.6.9 Task Chain Analysis

APP4RTA

Amalthea Model: ChallengeModel_TCs.amxmi

Task Name	PU Num	Default IA	Enter IA	0: Denver	Response Time	1: Denver	Response Time	2: A57	Response Time	3: A57	Response Time
OS_Overhead	4	<input type="button" value="Default IA"/>	<input type="button" value="Enter IA"/>	Planner	13358534500	DASM	1302430000 p	Localization	392590097500	CANbus_poll	
Lidar_Grabber	1			PRE_Detectio	73565439500	Lidar_Grabber	18265272000			PRE_SFM_g	
DASM	1									PRE_Lane_d	
CANbus_polling	3	<input checked="" type="radio"/> Synchronous	<input type="radio"/> Asynchronous							PRE_Localiz	
EKF	4	<input type="radio"/> Worst-Case	<input type="radio"/> Average-Case								
Planner	0	<input type="radio"/> Best-Case									
PRE_SFM_gpu...	3	<input type="button" value="Calculate"/>									
PRE_Localizati...	3	<input type="button" value="Reset"/>									
PRE_Lane_det...	3										
PRE_Detection...	0										
SFM	6										
Localization	2										
Lane_detection	5										
Detection	6										

Schedulability:

Cumulated Memory-Access Cost: 5361668000 ps

Cumulated Contention: 24795710000 ps

Computation: 635075050500 ps

Response Time Sum: 665232428500 ps

EVENT CHAIN MODEL: Li-Lo-EK-P-DA

Reaction (Implicit)		Reaction (LET)		Initial Reaction (Implicit)		Initial Reaction (LET)		Age (Implicit)		Age (LET)	
Best-Case	402348334000 ps	Best-Case	468000000000 ps	Best-Case	402348334000 ps	Best-Case	468000000000 ps	Best-Case	47500000000 ps	Best-Case	50000000000 ps
Worst-Case	893304764000 ps	Worst-Case	931000000000 ps	Worst-Case	898304764000 ps	Worst-Case	536000000000 ps	Worst-Case	528956430000 ps	Worst-Case	425000000000 ps

Task Chain >

- 1: Core1 (Denver)
- Lidar_Grabber
- 2: Core2 (A57)
- Localization
- 3: Core4 (A57)
- EKF
- 4: Core0 (Denver)
- Planner
- 5: Core1 (Denver)
- DASM

Single-core Initial Reaction: Critical-Case: Not Single-Core. Best-Case: Not Single-Core. Worst-Case: Not Single-Core.

Single-core Worst-case Reaction: Critical-Case: Not Single-Core. Best-Case: Not Single-Core. Worst-Case: Not Single-Core.

Now that every task's response time is measured, it is possible to measure end-to-end task chain latency with the derived task response times. (1) To analyze end-to-end task chain latency, a task chain in the combo-box should be selected first. (2) Click the Calculate button, then the selected task chain would be illustrated (3) and all measurement results would also be printed out (4)(5). Since the observed Amalthea model is a multi-core model here, the single-core analysis results are not available (5).

2.6.10 Change The Model

The screenshot shows the APP4RTA interface with the 'Amalthea Model' section. The current model is 'ChallengeModel_TCs.amxmi'. The 'Search Amalthea' button is circled in red and labeled '1'. An 'Open File' dialog is open, showing the file 'ChallengeModel_SingleTCs.amxmi' selected in the file list, circled in red and labeled '2'. The 'Open' button in the dialog is circled in red and labeled '3'.

Amalthea Model

ChallengeModel_TCs.amxmi **Search Amalthea** 1

Task Name **PU Num**

OS_Overhead 4 **Default IA**

Lidar_Grabber 1 **Enter IA**

DASM 1

CANbus_polling 3 ☒ **Synchronous**

EKF 4 ☐ **Asynchronous**

Planner 0 ☒ **Worst-Case**

PRE_SFM_gpu... 3 ☐ **Average-Case**

PRE_Localizati... 3 ☐ **Best-Case**

PRE_Lane_det... 3 **Calculate**

PRE_Detection... 0 **Reset**

SFM 6

Localization 2

Lane_detection 5

Detection 6

Schedulability

Scheduleable! :) ☐

Cumulated Memory-Access Cost

5361668000 ps

Cumulated Contention

24795710000 ps

Computation

635075050500 ps

Response Time Sum

665232428500 ps

Open File

Look In: WATERSTIS_SingleTCs

ChallengeModel_SingleTCs.amxmi 2

File Name: ChallengeModel_SingleTCs.amxmi

Files of Type: Amalthea models

Open **Cancel** 3

EVENT CHAIN MODEL Li-Lo-EK-P-DA **Calculate** **Reset**

Reaction (Implicit)

Best-Case 402348334000 ps

Worst-Case 893304764000 ps

Initial Reaction (Implicit)

Best-Case 402348334000 ps

Worst-Case 898304764000 ps

Age (Implicit)

Best-Case 4750000000 ps

Worst-Case 528956430000 ps

Reaction (LET)

Best-Case 468000000000 ps

Worst-Case 931000000000 ps

Initial Reaction (LET)

Best-Case 468000000000 ps

Worst-Case 536000000000 ps

Age (LET)

Best-Case 5000000000 ps

Worst-Case 425000000000 ps

Single-core Initial Reaction

Critical-Case Not Single-Core.

Best-Case Not Single-Core.

Worst-Case Not Single-Core.

Single-core Worst-case Reaction

Critical-Case Not Single-Core.

Best-Case Not Single-Core.

Worst-Case Not Single-Core.

Task Chain >

1: Core1 (Denver)

Lidar_Grabber

2: Core2 (A57)

Localization

3: Core4 (A57)

EKF

4: Core0 (Denver)

Planner

5: Core1 (Denver)

DASM

It is possible to change the observed model without clicking the **Reset** buttons. Apply the same process but this time with the `ChallengeModel_SingleTCs.amxmi` file that is a single-core Amalthea model (1) (2) (3).

2.6.11 Single-core RTA

APP4RTA

Amalthea Model: ChallengeModel_SingleTCs.amxmi

Task Name	PU Num	Default IA	Enter IA	0: A57	Response Time
Task0	0	<input type="button" value="Default IA"/>	<input type="button" value="Enter IA"/>	Task0	100000000000
Task1	0			Task1	200000000000
Task2	0			Task2	300000000000
Task3	0			Task3	900000000000

☒ Synchronous
☐ Asynchronous

☒ Worst-Case
☐ Average-Case
☐ Best-Case

Schedulability:

Cumulated Memory-Access Cost:

Cumulated Contention:

Computation:

Response Time Sum:

EVENT CHAIN MODEL:

Reaction (Implicit)		Reaction (LET)		Initial Reaction (Implicit)		Initial Reaction (LET)		Age (Implicit)		Age (LET)	
Best-Case	402348334000 ps	Best-Case	468000000000 ps	Best-Case	402348334000 ps	Best-Case	468000000000 ps	Best-Case	4750000000 ps	Best-Case	5000000000 ps
Worst-Case	893304764000 ps	Worst-Case	931000000000 ps	Worst-Case	898304764000 ps	Worst-Case	536000000000 ps	Worst-Case	528956430000 ps	Worst-Case	425000000000 ps

Single-core Initial Reaction		Single-core Worst-case Reaction	
Critical-Case	Not Single-Core.	Critical-Case	Not Single-Core.
Best-Case	Not Single-Core.	Best-Case	Not Single-Core.
Worst-Case	Not Single-Core.	Worst-Case	Not Single-Core.

Task Chain >

- 1: Core1 (Denver)
 - Lidar_Grabber
- 2: Core2 (A57)
 - Localization
- 3: Core4 (A57)
 - EKF
- 4: Core0 (Denver)
 - Planner
- 5: Core1 (Denver)
 - DASIM

The ChallengeModel_SingleTCs.amxmi model only has one processing unit with four tasks. As it is already mentioned, the Default IA only contains numbers of 0 because a single-core model is loaded this time. The process is the same.

2.6.12 Single-core Task Chain Analysis

APP4RTA

Amalthea Model: ChallengeModel_SingleTCs.amxml Search Amalthea

Task Name PU Num

Task0 0 Default IA

Task1 0 Enter IA

Task2 0

Task3 0

☒ Synchronous ☐ Asynchronous

☒ Worst-Case ☐ Average-Case ☐ Best-Case

Calculate

Reset

Schedulability: Scheduleable! :)

Cumulated Memory-Access Cost: 0 ps

Cumulated Contention: 0 ps

Computation: 15000000000000 ps

Response Time Sum: 15000000000000 ps

0: A57

Response Time

Task0: 10000000000000

Task1: 20000000000000

Task2: 30000000000000

Task3: 90000000000000

EVENT CHAIN MODEL: EC_10-5-6-3 Calculate Reset

Reaction (Implicit)

Best-Case: Not Multi-Core.

Worst-Case: Not Multi-Core.

Initial Reaction (Implicit)

Best-Case: Not Multi-Core.

Worst-Case: Not Multi-Core.

Age (Implicit)

Best-Case: Not Multi-Core.

Worst-Case: Not Multi-Core.

Reaction (LET)

Best-Case: Not Multi-Core.

Worst-Case: Not Multi-Core.

Initial Reaction (LET)

Best-Case: Not Multi-Core.

Worst-Case: Not Multi-Core.

Age (LET)

Best-Case: Not Multi-Core.

Worst-Case: Not Multi-Core.

Single-core Initial Reaction

Critical-Case: 160000000000000 ps

Best-Case: 80000000000000 ps

Worst-Case: 160000000000000 ps

Single-core Worst-case Reaction

Critical-Case: 190000000000000 ps

Best-Case: 150000000000000 ps

Worst-Case: 230000000000000 ps

Task Chain >

1: Core0 (A57)

Task3

2: Core0 (A57)

Task1

3: Core0 (A57)

Task2

4: Core0 (A57)

Task0

Now that every task's response time is measured, it is possible to measure end-to-end task chain latency with the derived task response times. The process is the same. However, a single-core model is analyzed this time. Therefore, latency results regarding single-core are only available while multi-core results are not in this case.

Download PDF file to see offline.

2.7 Future Work

Many implementations and tests have been left for the future due to the limited time but the topic has so much potential to be developed further. The future work concerns the followings:

2.7.1 1. Reaction Update

The current implementation covers `Early Reaction` but does not cover `Reaction Update`. To calculate `Reaction Update`, the number of sampled task-chain entity instances should be taken into account first, and then `Early Reaction` can finally be utilized to get `Reaction Update`. For the details, see `reaction-update`.

2.7.2 2. Blocking

The current implementation only focuses on preemptive tasks but does not cover cooperative tasks. Preemptive tasks preempt each other at any moment in time while cooperative tasks preempt each other at runnable boundaries. Therefore, the preempting task should be blocked until the currently executing runnable of the preempted task to finish.

2.7.3 3. Scheduling mode: EDF

The type of real-time scheduling algorithm used in this project is RMS (Rate Monotonic Scheduling). Under RMS, a task with the shorter period obtains a higher priority. To analyze different response times and mapping scenarios, extending the current scheduling algorithm further to EDF (Earliest Deadline First) can be done. Under EDF, tasks are sorted by using their deadlines. Therefore, a task which has the earliest deadline runs first.

2.7.4 4. Read & Write latency setting feature

The current implementation derives `memory access costs` with `read` and `write` latency attribute values from the processing unit. If the selected model does not describe these attributes, the default latency value is assigned to the processing unit and then the `memory access costs` is calculated. Therefore, having a GUI feature for assigning these attribute values is reasonable and useful for users to analyze with different processing unit configurations.

2.7.5 5. Data Age metrics should be organized

Currently, the GUI features for `Data Age` latency are not well-designed because the list for label names and the rest of the lists for latency values are not synchronized. Therefore, this should be restructured in a more tidy way to prevent possible confusions.

With these extensions, APP4RTA users can analyze response times under more various configuration settings with the better quality of GUI features.

2.8 Repositories

2.8.1 Eclipse Contribution Tagged Repo

Click [Eclipse Contribution Tagged Repository](#)

2.8.2 ReadTheDocs Repo

Click [ReadTheDocs Documentation Repository](#)

2.9 Reference

- [1] Finding Response Times in a Real-Time System (Mathai Joseph and Paritosh Pandya, 1986)
- [2] End-To-End Latency Characterization of Implicit and LET Communication Models (Jorge Martinez, Ignacio Sanudo, Paolo Burgio and Marko Bertogna, 2017)
- [3] CPU-GPU Response Time and Mapping Analysis for High-Performance Automotive Systems (Robert Höttger, Junhyung Ki, The Bao Bui, Burkhard Igel and Olaf Spinczyk, 2019)
- [4] Model-based Task Chain Latency and Blocking Analysis for Automotive Software (Robert Höttger, The Bao Bui, Junhyung Ki, Burkhard Igel and Olaf Spinczyk, Not yet published)

2.10 Contact

Name: Junhyung Ki

Personal Email: kijoonh91@gmail.com

Student Email: junhyung.ki001@stud.fh-dortmund.de

[LinkedIn](#)