D2.5 Evaluation of performance-aware model transformations

Version 1.0

Documentation Information

Contract Number	871669
Project Webpage	https://www.ampere-euproject.eu/
Contractual Deadline	30.06.2023
Dissemination Level	Public (PU)
Nature	Report
Authors	Sara Royuela, Adrian Munera, Oriol Pascual (BSC)
Contributors	Tommaso Cucinotta, Gabriele Ara (SSSA)
Reviewer	Thomas Vergnaud, Olivier Constant (TRT)
Keywords	Optimisation, Performance, OpenMP Taskgraph

AMPERE project has received funding from the European Union's Horizon 2020 research and innovation programme under the agreement No 871669.

Change Log

Version	Description Change
V0.1	Initial version with the original deliverable description.
V0.2	First complete version.
V0.3	Reviewers comments and suggested changes.
V1.0	Reviewers' comments included.

Table of Contents

1 Executive Summary

This deliverable covers the work done during the fourth and last phase of the AMPERE project within WP2. The deliverable spans 9 months' work, as defined in the Grant Agreement amended in December 2021 [1] (from month 34 until month 42), and includes the work done in Task 2.5, *Performance-aware model transformation validation* to reach milestone 4 (MS4). Concretely, the deliverable covers the activities conducted within WP2 towards validating the multi-criteria optimisation model transformations addressing performance, developed in Task 2.3, *Performance-aware transformation techniques*, and presented in deliverable D2.3 [2].

At MS4, Task 2.5 has to provide an evaluation of the performance optimisation strategy applied to the AMPERE use cases. To that end, the task has devoted efforts to:

- augment the PCC and ODAS models provided by partners BOSCH and GTSI (formerly THALIT) respectively, with the augmentations described in D2.3 for expressing parallelism and heterogeneity, including the support for function specializations, and
- analyze the effects on performance of using OpenMP to parallelize AMALTHEA tasks with the AMPERE use cases.

Tasks 2.5 has been carried out successfully, and all objectives of MS4 have been reached and documented in this deliverable.

2 Introduction

WP2 aims to develop a meta parallel programming abstraction independent of the underlying processor architecture, capable of capturing all system functional and non-functional requirements, as well as incorporating the parallel semantics required to enable an efficient model transformation, optimized for performance, timing, resiliency, security, and energy-efficiency. Figure 1 depicts the AMPERE software tools workflow, with the different components and the communication between them. WP2 is enclosed in the yellowish box on the left.

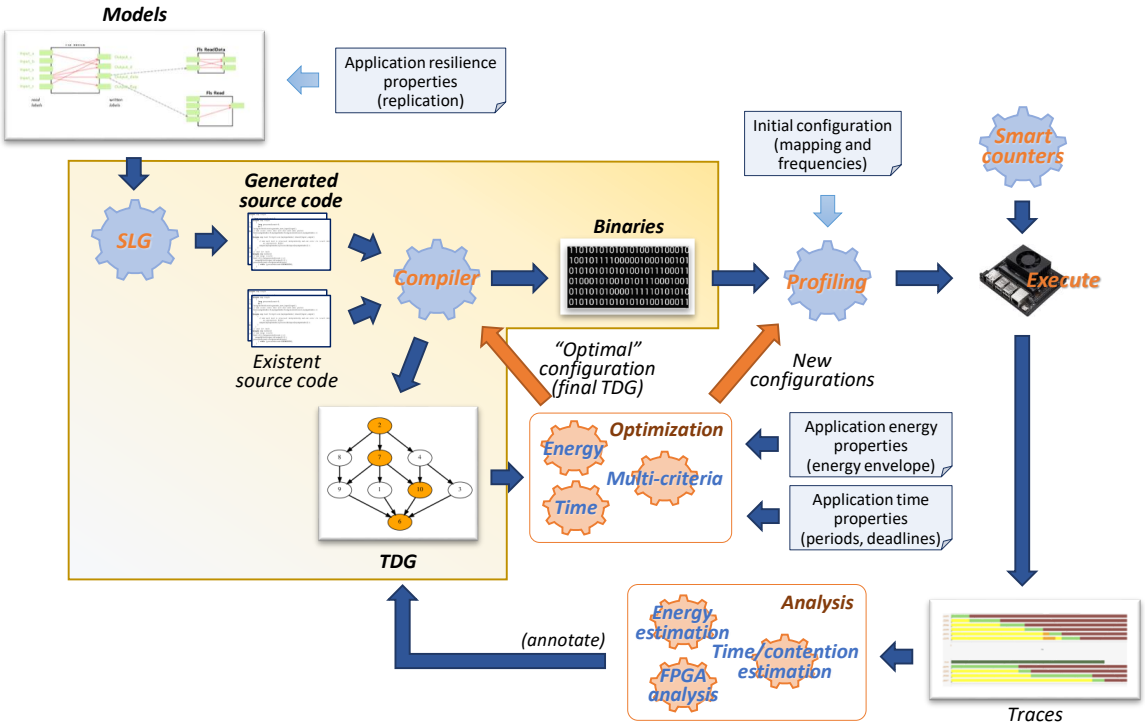


Figure 1: The AMPERE software ecosystem with WP2 work highlighted

This deliverable, including task T2.5, corresponds to the performance evaluation of the model augmentations and model-to-code transformations defined to express fine-grained parallelism within processes (or tasks) of an AMALTHEA model. In particular, the use of OpenMP to orchestrate the parallel execution of AMALTHEA runnables.

3 Modeling parallelism in the AMPERE use-cases

Use-case providers BOSCH and THALIT/GTSI have developed the base versions of the PCC and the ODAS use cases, respectively. These versions allow exploiting accelerators (FPGA and GPU) for some components, but they do not exploit parallelism within AMALTHEA tasks. To that end, the base models have been augmented with the annotations described in D1.5 [3] to express parallelism and heterogeneity based on specializations.

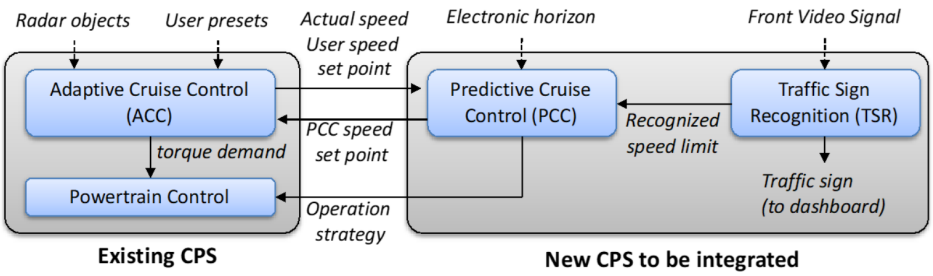
This chapter describes how parallelism and heterogeneity is exploited in AMPERE within each of the use cases of the project, and what are the reasons for the use of such configurations.

3.1 Predictive cruise control (PCC)

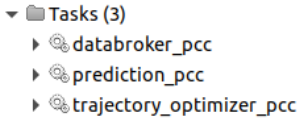
The PCC use case is formed by four different components, as shown in figure 3: the Powertrain control or Engine Control Management (ECM), the Adaptive Cruise Control (ACC), the Predictive Cruise Control (PCC) and the Traffic Sign Recognition (TSR). The components have different behaviors with respect to the number and the granularity of the runnables that compose them:

- ECM has several tasks with several runnable calls within each task, and all runnables show very thin granularity.
- TSR has a reduced number of tasks that run sequentially one after the other, and all tasks are composed of a unique runnable call, meaning that inter-runnable parallelism is not possible in this component.
- PCC has a very reduced number of tasks with a reduced number of runnables inside and a quite thin granularity in all of them.
- ACC has a reduced number of tasks, with some tasks running very limited number of runnables with thin granularity, but some others run a quite large amount of coarser-grained runnables.

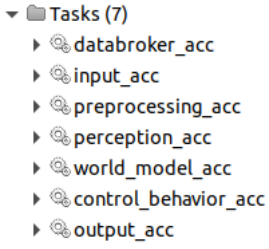
(a) ECM tasks.



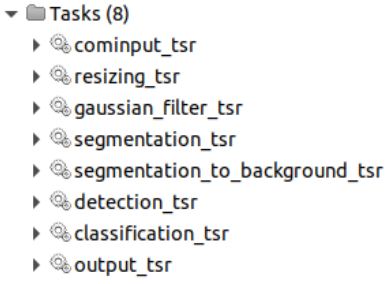
(b) Workflow overview.



(c) PCC tasks.



(d) ACC tasks.



(e) TSR tasks.

Figure 2: PCC use case and its modeling with AMALTHEA tasks.

Although OpenMP can be used within all components, analysis performed during the development of the OpenMP runtime for predictability have shown that, although our proposed runtime handles much thinner task granularities than the original runtime, it still has a limit when reaching granularities smaller than 10^{-3} seconds if performance is to be obtained out of the parallelization. This results were preliminary shown in D4.2 [4], and recently presented in an article accepted in the IEEE Transactions on Parallel and Distributed Systems (TPDS) [5]. Based on this know-how, we performed an exploratory analysis on the scalability of OpenMP using an x86 machine, applying OpenMP to all components (except for TSR, because of its sequential nature) and executing with 1 and 8 threads. The results, shown in Table 1, demonstrate that the ECM and the PCC components have too thin granularity to be exploited with OpenMP. ACC, instead, shows good scalability for the majority of the tasks.

Table 1: Exploratory analysis of the scalability of the PCC use-case, when using 1 and 8 threads, in the NVIDIA Jetson AGX (time in ms).

Component	AMALTHEA task	1 thread	8 threads
ECM	task_200ms	0.023573	0.042087
	task_1000ms	0.023088	0.048195
	task_2ms	0.023169	0.049279
	task_50ms	0.023292	0.042001
	task_1ms	0.023173	0.055221
	task_5ms	0.023258	0.047233
	angle_sync	0.023610	0.034236
	task_20ms	0.024481	0.034234
	task_100ms	0.024352	0.063528
	task_10ms	0.024718	0.034559
PCC	prediction	0.035444	0.034830
	trajectory_optimizer	0.010112	0.026043
ACC	input	0.052142	0.067312
	output	0.019944	0.031140
	preprocessing	**11.754811**	**2.131537**
	perception	**13.582399**	**1.908297**
	world_model	**11.706874**	**1.748688**
	control_behavior	1.415694	0.367440

Given the previous results, the PCC use case provided by BOSCH has been modified to parallelize the ACC component by adding a *Parallelism* custom property. The results are shown in Section 4.2.1

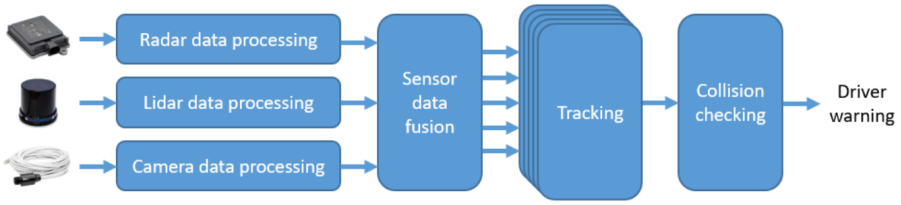
3.2 Obstacle detection and avoidance system (ODAS)

The ODAS use case is formed of 4 different phases, as depicted in figure 3 that occur sequentially one after the other, :

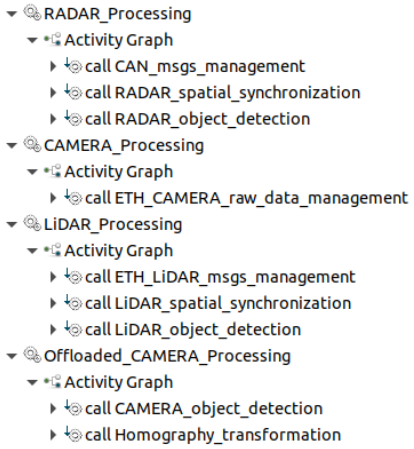
- Phase 1 - Data processing. In this phase, three Amalthea tasks run in parallel, one for each sensor, i.e., camera, lidar and radar. In the case of the camera, there are actually two tasks, one executing in the CPU (the one that gathers data) and one executing in the GPU (the one processing the data). The tasks are composed of one-to-three runnable calls, and these runnables cannot be parallelized as they represent sequential phases on the data collected.
- Phase 2 - Sensor data fusion. In this phase, one Amalthea task executes three runnables that synchronize the data received from the sensors, associates them and recognizes objects. This steps must run sequentially.
- Phase 3 - Tracking. This phase executes one Amalthea task with one Unscented Kalman Filter (UKF) for

each object detected in the previous step. These processes can run in parallel as they are independent among them.

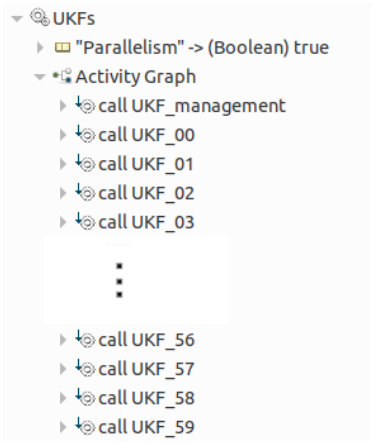
- Phase 4 - Collision checking. This phase contains one Amalthea task that runs one unique runnable.



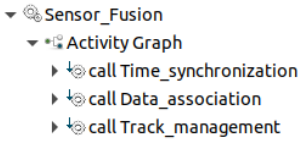
(a) Workflow overview.



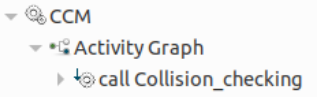
(b) Phase 1 tasks.



(c) Phase 3 tasks.



(d) Phase2 tasks.



(e) Phase 4 tasks.

Figure 3: PCC use case and its modeling with AMALTHEA tasks.

Given the analysis of the system, the only component that exposes inter-runnable parallelism is the tracking module, which can run the UKFs in parallel. The corresponding AMALTHEA task has been augmented with a *Parallelism* custom property. The results are shown in Section 4.2.1

4 Use cases evaluation

The performance capabilities of the AMPERE software (SW) ecosystem are evaluated on top of two different processor architectures: (1) the NVIDIA Jetson AGX Xavier [6], and (2) the Xilinx Zynq UltraScale+ MPSoC ZCU102 [7]. Parallelism and heterogeneity is accomplished in two different ways depending on the architecture. While the GPU is exploited through OpenMP, the FPGA is exploited with FRED. The components of the AMPERE ecosystem for CPU+GPU exploitation are defined in D2.3 [2], while the components for CPU+FPGA exploitation are defined in D4.3 [8]. Following the same structure, the evaluation of the two boards is also split in two different deliverables, i.e., D2.5 (this document) evaluates the performance of the CPU+GPU support in AMPERE, while D4.4 [9] evaluates the performance of the CPU+FPGA support.

4.1 Environmental setup

This section introduces the environmental setup used for the evaluation of the AMPERE software ecosystem with regard to performance.

- **Hardware architecture**. The NVIDIA Jetson AGX Xavier platform was introduced in D5.1 [10]. As a summary, the board counts with an octal-core NVIDIA Carmel ARMv8.2 CPU @2.26GH with 32GB 256-bit LPDDR4x and a 512-core Volta GPU.
- **Software**. The board counts with the software presented in D6.5 [11], which includes the agx2 5.10.104-tegra OS, CUDA 11.4, ROS2 version Foxy and the LLVM 17.0 extended as described in D2.3 [2].
- **Use-cases configurations**. The use cases evaluated are those proposed in the AMPERE project and described in D1.5 [3]:
  - The Predictive Cruise Control (PCC) use case is an heterogeneous and adaptable system that presents runnables exclusively for CPU execution, runnables that have specializations for CPU and FPGA, and runnables with specializations for CPU and GPU. In the Jetson, two configurations are tested: (1) a complete CPU version, and (2) a version that has CPU runnables (the exclusive CPU ones and those with CPU/FPGA specializations) and GPU runnables (those with CPU/GPU specializations).
  - The Obstacle Detection and Avoidance System (ODAS) use case, instead, exposes either GPU runnables, within the camera pipeline for object recognition, or CPU runnables (all the rest). Consequently, there is only one configuration regarding the target platform.

4.2 Performance speedup

The performance speedup of the system is measured by executing the different configurations of the use-cases with 1, 2, 4, 6 and 8 threads. The results presented next are extracted using the $-O2$ optimization flag during compilation, and exploit the Taskgraph framework based on the TDG representation presented in D2.2 [12] and refined in D2.3 [2]. The figure are computed as the average of 20 executions of the applications running for 20 seconds.

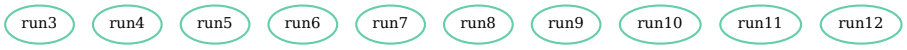
4.2.1 PCC

Table 2 shows the characterization of the tasks in each component of the PCC use case, including the total number of tasks, the number of tasks that show inter-runnable parallelism, and the granularity of the runnables.

The only component showing enough granularity to exploit inter-runnable parallelism is the ACC. The TDGs that represent the tasks of this component are shown in figure 4.

Table 2: Characterization of the tasks in the PCC use case.

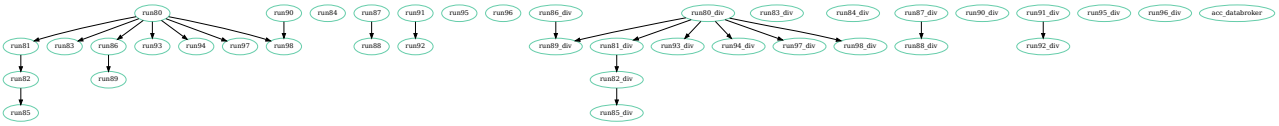
	AMALTHEA tasks	Tasks w. inter-runnable parallelism	Granularity
ACC	6	6	10^4
ECM	22	22	10^1
PCC	3	2	10^1
TSR	8	1	10^1-10^2



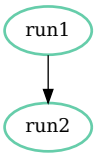
(a) Preprocessing.

(b) Perception.

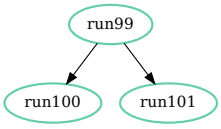
(c) World model.



(d) Control behavior.



(e) Input.



(f) Output.

Figure 4: TDGs of the AMALTHEA tasks from the ACC component of the PCC use case.

4.2.1.1 CPU execution

The performance speedup of the PCC use case when running all components in the CPU is presented in figure 5. All components reach around 1.5x speedup for all threads, showing slight variations when increasing the numbers of threads due to the overhead of handling parallelism. This seems to be the limit for exploiting fine grain parallelism in this machine due to the competition with other threads handled by ROS that are in charge of executing all the AMALTHEA tasks of the system. The two tasks that suffer more the overhead of increasing the number of threads are *perception* and *world model*. These tasks present a high number of runnables in parallel task, achieving their best speedup with 4 threads. The *input* and *output* tasks are not show because the do not obtain speedup neither slowdown.

To understand the possibilities of the platform to obtain performance out of the ACC component, we run experiments with this component isolated. The performance speedup obtained in the ACC AMALTHEA tasks with respect to running the whole use case concurrently are shown in figure 6. As expected, the *perception* runnable obtains up to 3x speedup with 8 threads, which proves (1) the capabilities of OpenMP to exploit fine grained parallelism in highly-parallel automotive components, and (2) confirms the fact the the PCC use-case already exploits a great amount of parallelism at the AMALTHEA task level, hence handled by the OS, and there are not enough resources in the tested processor architecture to allow OpenMP fully exploit the parallel nature of some AMALTHEA tasks.

The results show how OpenMP is able to efficiently parallelize AMALTHEA tasks that expose fine-grained parallelism, and evince the need for highly parallel architectures to support the advanced automotive systems represented with the PCC use case.

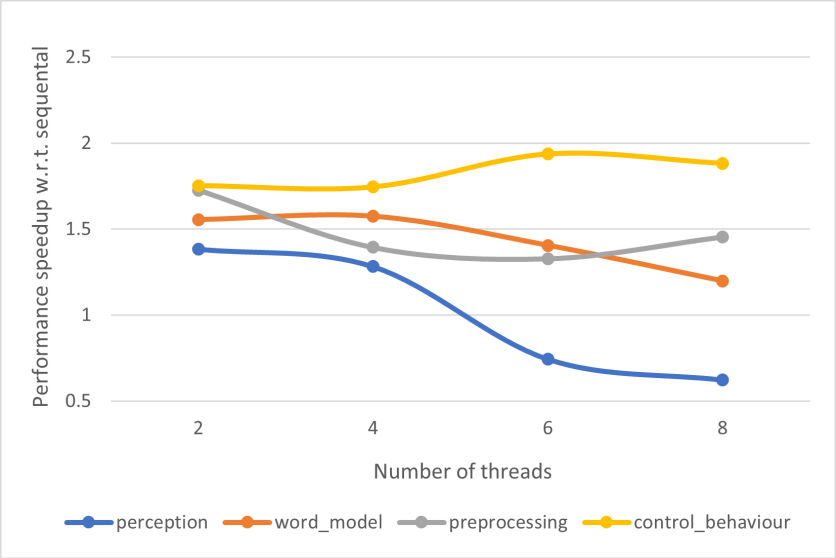


Figure 5: Performance speedup of the PCC use-case when increasing the number of threads.

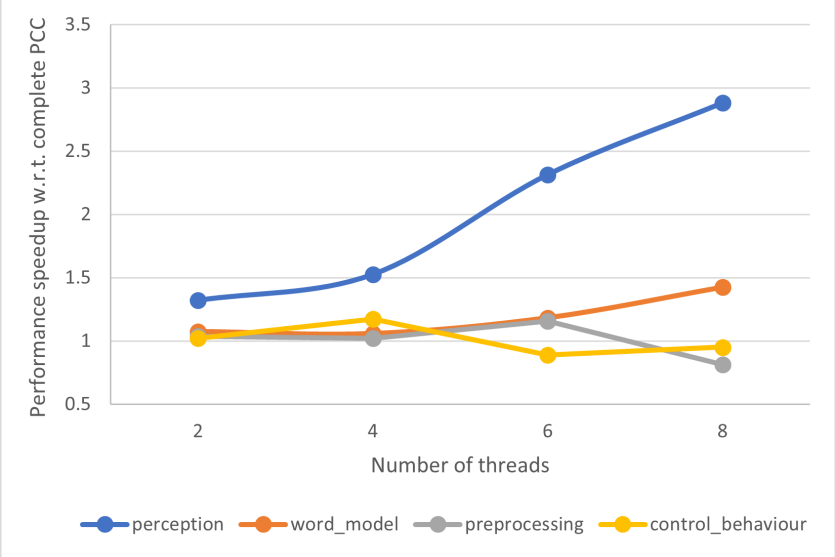


Figure 6: Performance speedup of the ACC component when running in isolation with respect to run the whole PCC use-case.

4.2.1.2 CPU + GPU execution

The PCC use case presents GPU specializations for the TSR component, where all tasks can run in the device. Overall, in this evaluation we are exploiting inter-runnable parallelism in the ACC component, and heterogeneous execution in the TSR component. This should leave more resources in the CPU for the competition between OpenMP threads and ROS threads.

The impact of using the GPU to execute the TSR component is shown as the impact on the component moved to the GPU, i.e., TSR, and the impact on the component run in the CPU, i.e., ACC. The impact on the ECM and PCC components is not shown because the instrumentation of this tiny-grained components would add too much overhead and the results would not be representative of the actual behavior of the use case.

- The TSR chart in figure 7 shows that most of the tasks (i.e. *resizing*, *classification*, *detection* and *segmentation_tsr*) have worse performance in the GPU than in the CPU, and this is because the granularity of the tasks is very thin ($10\mu s$), and the overhead of offloading surpasses the benefits of the accelera-

tor. In the case of the *gaussian_filter* task, there is a considerable gain when increasing the number of threads because in the all-CPU version, these configurations were suffering important penalties due to the overhead.

- The ACC chart in figure 8 shows instead performance speedup with respect to the all-CPU version in the *perception* task, as more CPU resources are free and the execution does not suffer from high overhead when increasing the number of threads.

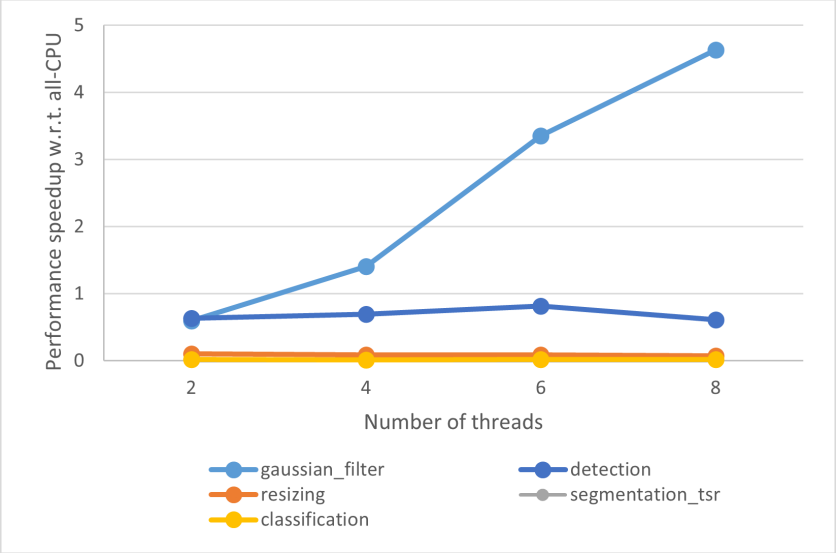


Figure 7: Performance speedup of the TSR component running in the GPU when increasing the number of threads.

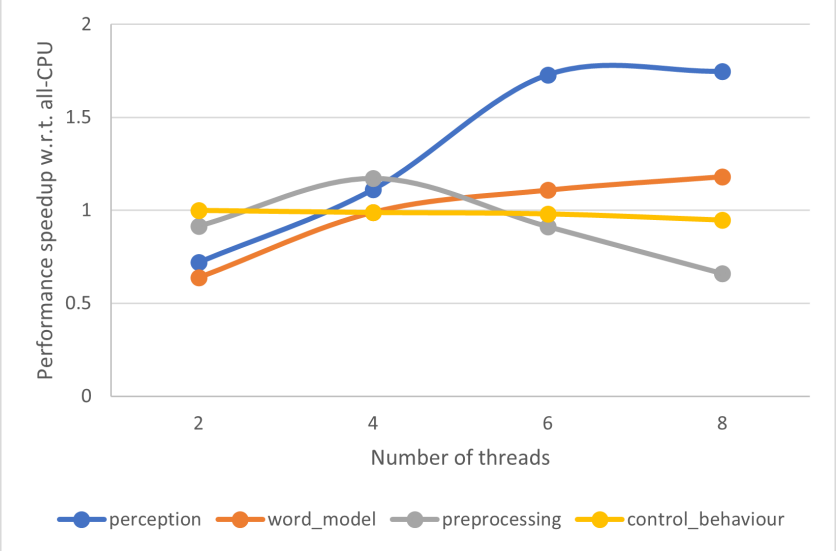
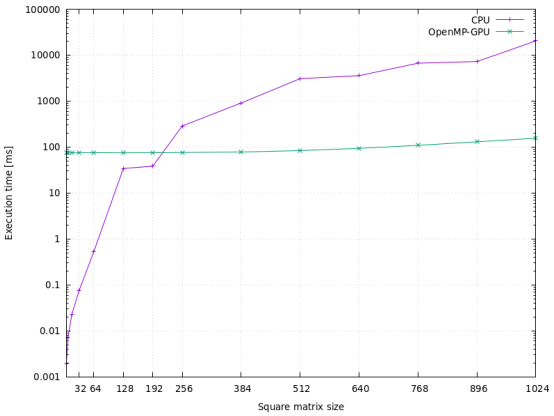


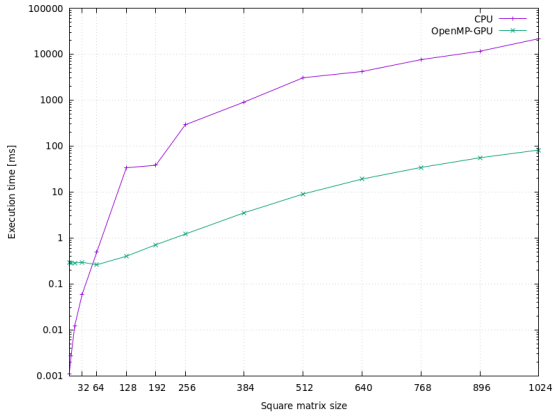
Figure 8: Performance speedup of the ACC component with TSR component running in the GPU when increasing the number of threads.

For illustration purposes, we tested the overhead of offloading work with OpenMP to the GPU on the NVIDIA Jetson Xavier AGX board, with the `target` directive using a simple matrix multiplication kernel that multiplies two matrices and stores the result in a third matrix, to later check if the result is an identity matrix. A sequential implementation is tested in the CPU and a GPU accelerated version is tested in the GPU. The time duration of the two implementations for the first execution, and for the subsequent executions, are shown in figure 9a and figure 9b, respectively. In the first execution, the GPU only starts showing better performance when the

matrix is bigger than 256x256 double elements, which corresponds to a computation time in the order of $10^4 \mu s$ (notice that this is the order of magnitude of the ACC component that we parallelized). After that, instead, matrices of 64x64 double elements already get benefit, which are in the order of $10^2 \mu s$ (which generally is still bigger than the order of magnitude in the TSR component). These results endorse the design decisions about parallelization and the results for GPU execution obtained with the PCC use case.



(a) Execution time vs matrix size on first kernel execution.



(b) Execution time vs matrix size on subsequent kernel executions.

Figure 9: Overhead of offloading a matrix multiplication kernel to the GPU w.r.t. CPU execution when varying the size of the matrix.

4.2.1.3 CPU + FPGA execution

The AMPERE toolchain has been applied to the PCC use-case, resulting in a number of different configurations for the generated models. One of these considered the possibility to accelerate some of the runnables in the use-case, offloading the computations to a hardware IP deployed onto an FPGA slot using FRED [13]. The optimization process is described in detail in deliverable D3.4 [14], along with the obtained optimization results. Furthermore, an experimental evaluation of the methodology has been carried out, deploying the optimized system configuration and placement on the Xilinx UltraScale+ ZCU102 board. The obtained experimental results are described in detail in deliverable D4.4 [9], showing results coming from optimizing and deploying the use-case models, as well as a number of randomly generated real-time DAG scenarios. The experimentation was carried out with scenarios optimized both for minimum power consumption, and for maximizing the minimum relative slack, namely maximizing the tolerance of the configuration w.r.t. possible glitches in worst-case execution time estimations and measurements.

From these experimentation, we can conclude that our AMPERE end-to-end methodology succeeds in optimizing the design and run-time configuration of parallel real-time software components for highly heterogeneous platforms, including FPGA acceleration. The methodoloy is capable of providing optimized configurations of the scenarios, so to deploy minimum-power and/or robust configurations that intelligently take advantage of FPGA accelerators as needed, given the non-functional timing, power and resilience requirements in place.

4.2.2 ODAS

The AMALTHEA task where parallelism is exploited through OpenMP is the one included in the *Tracking* or *UKF* component. The TDG of this task is shown in figure 10, showing 60 runnables or OpenMP tasks, namely *UKF_XX*, that can run in parallel.

The performance speedup of the ODAS use case, running the *camera data processing* pipeline in the GPU as a requirement, and the rest of the components in the CPU is presented in figure 11. In this case, the speedup

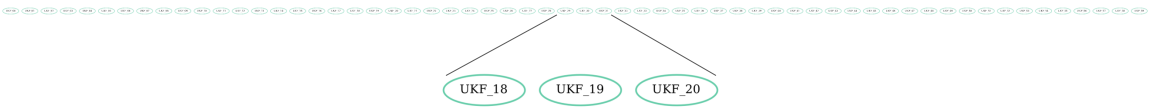


Figure 10: TDG of the AMALTHEA task implementing the 60 UKFs of the ODAS use case (with zoom-in in a region with three of them).

reaches a top value of 2.6x when using 4 threads, and remains between 1.5x and 2x for the rest of the configurations. This happened also with the PCC use case; ROS threads are probably competing with the OpenMP threads to get resources.

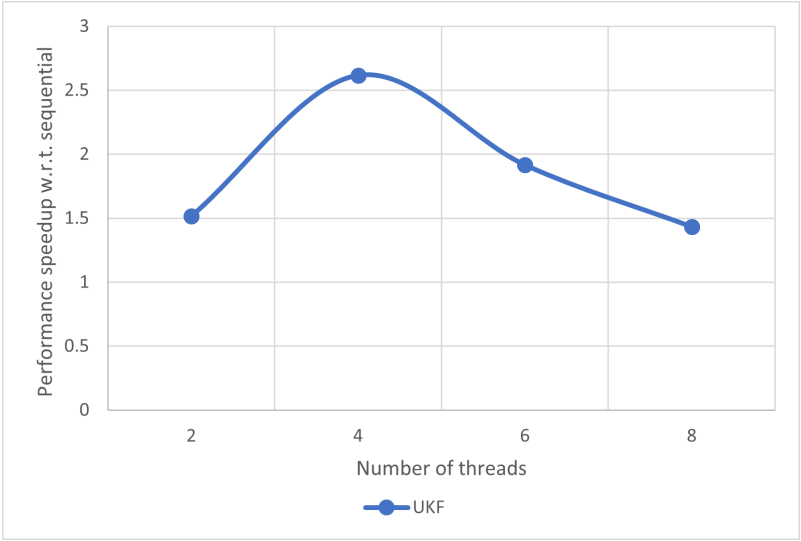


Figure 11: Performance speedup of the ODAS use-case when increasing the number of threads.

To understand the possibilities of OpenMP in the Jetson processors, we run the experiments with the *UKF* component isolated. The results, shown in Figure 12, exhibit the same behavior as for the PCC case, in which the architecture does not allow to exploit all the parallelism exposed by the application. Still, OpenMP shows benefit in parallelizing fine-grained tasks up to the maximum parallelism of 8 threads for the UKF.

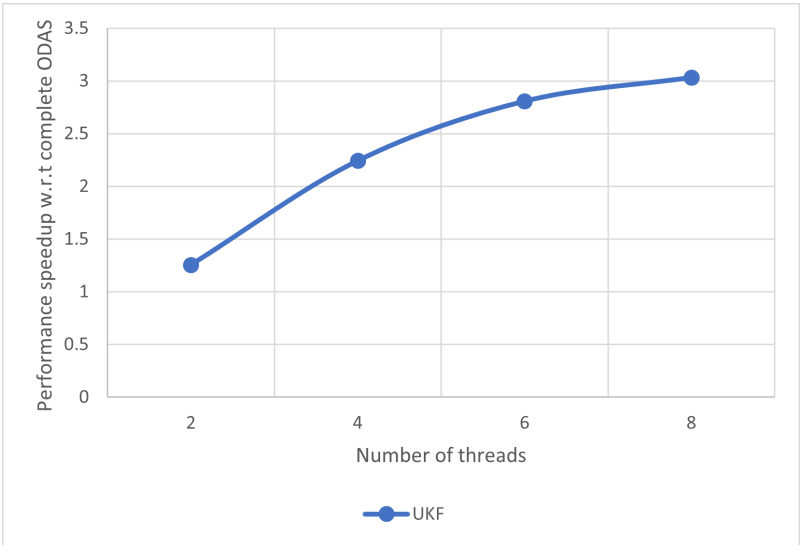


Figure 12: Performance speedup of the UKF component when running in isolation with respect to run the whole ODAS use-case.

5 Conclusions

This deliverable presents the final evaluation of the AMPERE ecosystem on top of the PCC and the ODAS use cases using the NVIDIA Jetson AGX Xavier platform. The results show how OpenMP is able to expose inter-runnable parallelism within AMALTHEA tasks, although it shows limitations in parallelizing very fine-grained tasks in automotive components such as the ECM from the PCC use case. Similarly, a computation to be offloaded to the GPU must have sufficient computational workload so the overhead of offloading does not surpass the benefits of the accelerator. These facts are to be understood by system designers, to decide where to expose parallelism and which type of device is better. However, the multi-criteria optimizer described in D3.4 is capable of considering all these factors when deciding what to offload and what to leave as running on the CPU.

Further tests simulate a system with more resources by temporarily removing from the PCC and the ODAS use cases those AMALTHEA tasks that do not exploit parallelism. The results show that OpenMP can obtain a performance speedup beyond the one shown when running the complete use cases. This is so because ROS threads implementing AMALTHEA tasks already occupy resources and they have to compete with OpenMP threads to execute their jobs. Based on this fact, we can conclude that the computing needs of the use cases that are being deployed in advanced automotive and railway systems push to the maximum the computing capabilities of current parallel embedded architectures, and the use of different levels of parallelism can help in reducing the worst case execution times.

The objectives of *Task 2.5, Performance-aware model transformation validation*, which produces this deliverable, have been met at MS4 by providing a successful evaluation of the performance optimization strategy applied to the AMPERE use cases.

6 Acronyms and Abbreviations

ACC	Adaptive Cruise Control
CPU	Central Processing Unit
D	Deliverable
ECM	Engine Control Management
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
IEEE	Institute of Electrical and Electronics Engineers
MDE	Model-Driven Engineering
MPSoC	Multiprocessor System-On-Chip
MS	Milestone
NFR	Non-Functional Requirement
ODAS	Obstacle Detection Avoidance System
OpenMP	Open Multi-Processing
OS	Operating System
PCC	Predictive Cruise Control
ROS	Robot Operating System
SLG	Synthetic Load Generator
T	Task
TPDS	Transactions on Parallel and Distributed Systems
TDG	Task Dependency Graph
TSR	Traffic Sign Recognition
UKF	Unscented Kalman Filter
WP	Work Package

7 References

[1] European Commission and AMPERE beneficiaries, "Grant Agreement Description of Action," 2021.

[2] AMPERE, "Deliverable D2.3, Programming model extensions and the multi-criteria performance-aware component," September 2022.

[3] ——, "Deliverable D1.5, Meta model-driven abstraction extensions and Use case enhancements," September 2022.

[4] ——, "Deliverable D4.2, Independent run-time energy support, and predictability, segregation and resilience mechanisms," March 2021.

[5] C. Yu, S. Royuela, and E. Quiñones, "Taskgraph: A Low Contention OpenMP Tasking Framework," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[6] NVIDIA, "Jetson AGX Xavier," 2023. [Online]. Available: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-agx-xavier/

[7] "Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit," https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html, 2023.

[8] AMPERE, "Deliverable D4.3, Integrated run-time energy support, and predictability, segregation and resilience mechanisms," September 2022.

[9] ——, "Deliverable D4.4, Evaluation of run-times," June 2023.

[10] ——, "Deliverable D5.1, Reference parallel heterogeneous hardware selection," June 2020.

[11] ——, "Deliverable D6.5, Final release of the AMPERE ecosystem," June 2023.

[12] ——, "Deliverable D2.2, First release of the meta parallel programming abstraction and the single-criterion performance-aware," March 2021.

[13] M. Pagani, A. Balsini, A. Biondi, M. Marinoni, and G. Buttazzo, "A linux-based support for developing real-time applications on heterogeneous platforms with dynamic FPGA reconfiguration," in *Proceedings of the 30th IEEE International System-on-Chip Conference (SOCC 2017)*, September 2017, pp. 5–8.

[14] AMPERE, "Deliverable D3.4, Evaluation of multi-criteria optimizations," June 2023.