Multi-criteria analysis and optimisation in the AMPERE ecosystem

Sara Royuela, Adrian Munera, Eduardo Quinones
Barcelona Supercomputing Centre, Spain; email: {sara.royuela, adrian.munera, eduardo.quinones@bsc.es}@bsc.es

Tiago Carvalho, Luís Miguel Pinho, Mohammad Samadi
Instituto Superior de Engenharia do Porto, Portugal; email: {tdc, lmp}@isep.ipp.pt

Tommaso Cucinotta, Gabriele Ara, Francesco Paladino
Scuola Superiore Sant'Anna, Italy; email: {tommaso.cucinotta, gabriele.ara, francesco.paladino}@santannapisa.it

Sergio Mazzola, Thomas Benz
ETH Zürich, Switzerland; email: {smazzola, tbenz}@iis.ee.ethz.ch

Abstract

The AMPERE project is developing the next generation of high-performance and energy efficient Cyber-Physical Systems supporting multi-criteria optimization. This paper provides the general overview of the AMPERE approach to analyse and optimise parallel real-time applications in heterogeneous platforms. More specifically, it details how parallel OpenMP programs are generated from AMALTHEA models, and the multi-criteria optimisation methodology, considering the fault-tolerance, time and energy properties of the targeted applications.

Keywords: Real-time Systems, Multi-criteria optimisation, AMPERE

1 Introduction

The growing computational demands of complex Cyber-Physical Systems (CPS) is hastening the introduction of parallel and heterogeneous architectures in domains with tight functional and non-functional requirements with respect to resilience, time and energy budgets, among other aspects. However, the parallel programming models, e.g. OpenMP, used to exploit parallelism multi-cores and accelerator devices are not compatible with the current Model-Driven Engineering (MDE) approaches used to develop CPS.

AMPERE strives to close the gap between the MDE techniques used in safety-critical automotive and railway systems and the parallel programming models used in high-performance systems by developing a complete software stack and development environment to help system developers leverage low-energy, highly-parallel, and heterogeneous systems in their development process, while fulfilling the non-functional constraints inherited from the cyber-physical interactions of safety-critical automotive and railway systems.

The paper is structured as follows. Section 2 presents a general overview of the AMPERE ecosystem and the flow used to analyse and optimise parallel applications considering the fault-tolerance, time and energy properties. The two main parts of the flow are provided next. Section 3 explains how OpenMP programs are generated from models, whilst Section 4 describes the multi-criteria optimisation approach.

2 The AMPERE project in a nutshell

AMPERE [1] is building a system design ecosystem and computing software to help system developers to leverage low-energy and highly-parallel and heterogeneous computation while fulfilling non-functional constraints.

2.1 The AMPERE ecosystem

One of the main challenges of the AMPERE project is to enable Model Driven Engineering (MDE) of CPS, accounting for parallelism and heterogeneity in high-performance embedded systems. As such, MDE tools provide the front-end to the entire AMPERE ecosystem (Figure 1). These tools include Domain Specific Modelling languages (DSML), e.g., AMALTHEA [2], which are used to describe the system in a modular and composable manner. Models are further annotated by system designers with the functional and non-functional requirements that determine how the system shall be generated. These annotations are key for the automatic optimization of the system of systems with respect to energy, timing guarantees, resilience, and heterogeneity [3].

Once the system has been modelled in AMALTHEA, a Synthetic Load Generator (SLG) [4] generates the corresponding source code, including OpenMP [5] annotations to exploit parallelism and heterogeneity. The source code is passed to an OpenMP compiler, for compilation. At this point, extensions provided in the OpenMP compiler allow for producing not only the binaries themselves, but also structured information of the system that is later used during the optimisation process [6] to ensure that the final system fulfils all requirements modelled in the DSML. The fundamental data structure generated as part of the structured information is the Task Dependency Graph (TDG) [7].

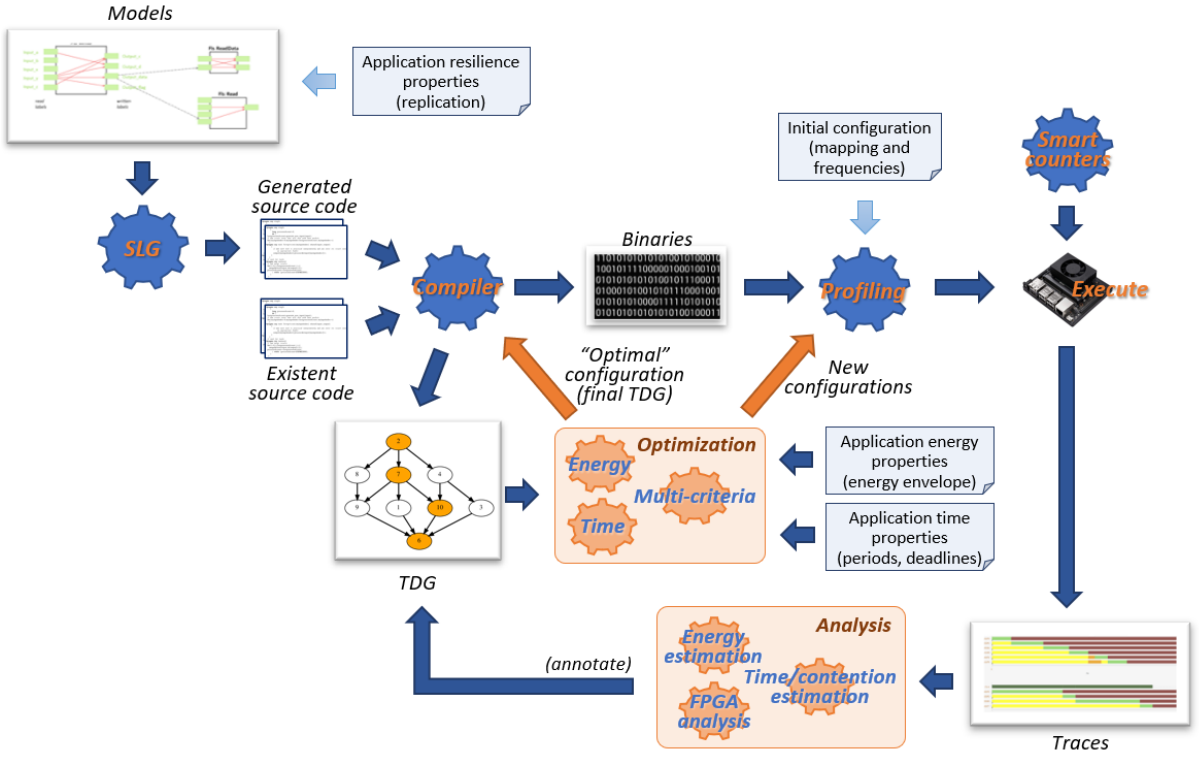


Figure 1: AMPERE software ecosystem flow.

The TDG provides the parallel structure of the different components of the system based on the dependencies described between the runnables of the AMALTHEA tasks, as outlined in the DSML. The TDG further contains meta-information that is used for the optimization (non-functional properties) annotated in the DSML towards which the system should be optimized. As part of the compilation process, the generated binary is profiled, and the information inserted in the TDG. As such, the TDG provides the necessary abstraction for determining the modelled requirements and dependencies of every task in the system, as well as wide information about the behaviour of each task as obtained through profiling.

2.2 The AMPERE analysis and optimisation flow

The analyses and optimization phases consist of multiple components operating in parallel: timing analysis and optimization, energy optimization, and scheduling. Heterogeneity and resilience techniques are implemented at the model and compiler levels, through the definition of function specializations (components which have multiple implementations, potentially for accelerators) and replication, respectively. This information is also included in the TDG and taken into consideration by the analyses.

Once the optimization phase has completed, it is either finished, i.e., all functional and non-functional requirements are guaranteed to be upheld, or another round of optimization is required. To that end, the AMPERE optimization relies on an optimization feedback loop that includes additional profiling information (red arrows in the figure).

The encoded information in the TDG allows for later use by the earlier components in the AMPERE pipeline, such that information in the model could either be updated, or warnings emitted to the MDE framework, and made available to the end user. At the end of the optimization pipeline, the TDG information can also be used to inject runtime hooks and configuration headers based on the optimization outcome into the generated source code. This enables actuation and monitoring of the non-functional requirements at runtime.

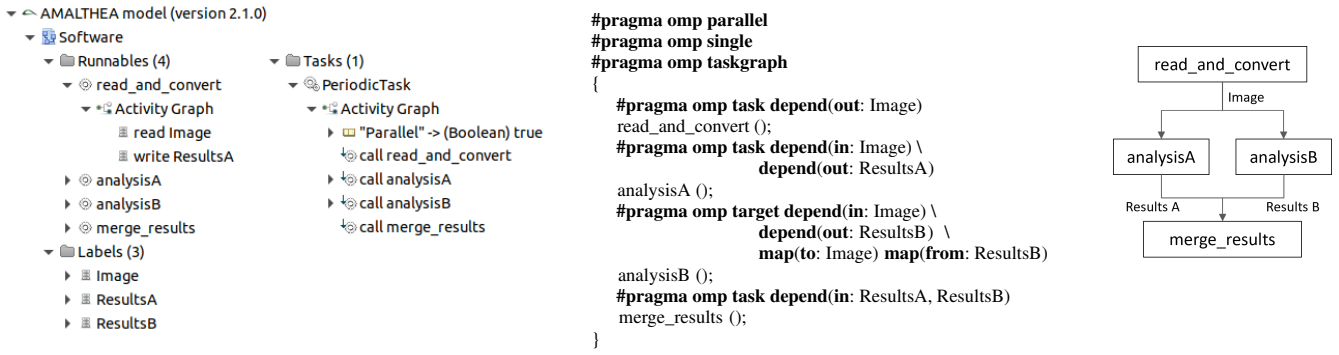
3 Model-to-code transformations

AMPERE defines model-to-code transformations targeting performance and fault tolerancein. These transformations are performed in two steps. First, AMALTHEA models are transformed into parallel code through the APP4MC SLG. Then, this code is analyzed and further transformed into a TDG to efficiently exploit the parallelism of the underlying processor architecture.

3.1 Performance

The synthesis tool included in the APP4MC framework processes AMALTHEA models by transforming runnables and tasks into C functions, and labels into global variables. In the frame of the AMPERE project, the SLG has been extended [8] to exploit concurrency among tasks no only with Pthreads for Linux systems, but also with ROS2 [9] primitives for ROS2 middleware communication. Moreover, extensions based on the OpenMP tasking model have been included to exploit parallelism within runnables from the same task.

Figure 2 illustrates the model-to-code transformation implemented in APP4MC in AMPERE, including a sample model in Figure 2a, the corresponding OpenMP code generated by the extended SLG in Figure 2b, and the TDG representing the OpenMP code in Figure 2c.



(a) AMALTHEA model **(b) OpenMP code** **(c) TDG**

Figure 2: Example of AMALTHEA to OpenMP transformation

A new custom property, named *Parallel*, triggers inter-runnable parallelism within AMALTHEA tasks (see igure 2a). For such tasks, the SLG wraps the associated code in a `parallel` directive followed by a `single` (see Figure 2b), to first spawn parallelism and second allow only one thread executing the inner region. Next, a taskgraph directive allows for optimizations towards predictability and performance [10]. Then, each runnable call is annotated with a tasking directive that depends on the processor defined for the runnable, i.e., a `task` directive for host runnables (AMPERE considers multi-core architectures) and a `target` directive for accelerator runnables (i.e., a GPU device). Finally, accesses to labels are used to define dependency clauses, transforming each read into an `in` dependency and each write into an `out` dependency. In the case of accelerated tasks, label accesses are also used to define the `map` clauses, which describe data movements between the host and the accelerator.

The code generated by the SLG is later compiled using an extended version of the LLVM compilation framework [11]. During this process, an extended version of the OpenMP tasking model [10] is used to replace the regions of OpenMP code that exploit the tasking model with a TDG (see Figure 2c). The TDG avoids the need for running the user code in order to instantiate and execute tasks, reducing time spent in context switching, and also enables optimizations at the runtime level that reduce contention due to accesses to shared resources (e.g., task queues) and overhead due to unnecessary computations (e.g., dependency ressolution). Besides enhancing the performance of the parallel orchestration, the TDG enables timing analysis techniques (see Section 4.1) for predictable execution. The TDG is described in a JSON format that is used as the interface between the different tools included in the multi-criteria optimization phase. Figure 3 illustrates a portion of the JSON corresponding to the example in Figure 2.

```
1: {
    "ins": [ ],
    "outs": [2,3]
},
2: {
    "ins": [1],
    "outs": [4]
},
```

```
3: {
    "ins": [1] ,
    "outs": [4]
},
4: {
    "ins": [2,3],
    "outs": [ ]
}
```

```
1: read_and_convert
2: analysisA
3: analysisB
4: merge_results
```

Figure 3: JSON format for describing TDG in Figure 2c.

3.2 Fault-tolerance

Fault-tolerance is addressed in AMPERE through software replication. The requirements for fault-tolerance are defined at the model level according to the Automotive Safety Integrity Level (ASIL), in the automotive use case, or the Safety Integrity Level (SIL), in the railway use case, of each component. Hence, runnables with an ASIL B or SIL 4 are defined with triple replication, while runnables with QM or SIL 0 are not replicated.

To express replication, the OpenMP `task` directive has been extended with the `replicated` clause [12]. This extension allows defining the number of replicas, the function used to check the results and the type of replication, with three different options: (a) *spatial*, which forces each replica and the original task to be executed in a different processor, allowing them to run in parallel, (b) *temporal*, which forces each replica and the original task to be executed in mutual exclusion among them, so they have to be sequentialized, and (c) *spatial_temporal*, which includes both cases.

The SLG has been extended to annotate tasks with a `replicated` clause when the ASIL B and SIL 4 levels are assigned to a runnable. The LLVM compiler has also been extended so it generates $n + 1$ tasks (where n is the replication level, i.e., 3 in AMPERE) when an annotated task is found. One of the tasks consumes the original data, while the rest consume copies of the data modified within the task to avoid race conditions. A synchronization task is inserted after the creation of the tasks, including as input dependencies all the tasks in the replication set, and inheriting as output dependencies those of the original task. Afterwards, a task performing the consolidation function is generated. This behavior is shown in Figure 4, where the application presented in Figure 2 defines $analysisB$ with ASIL B.

The replication mechanism has been optimized to support *MooN* optimizations, where M is the number of replicas that need to finish successfully out of a total of N replicas. A compilation flag is added to LLVM to enable this optimization and reduce the overhead of the replication.

4 Multi-criteria optimization

AMPERE defines a multi-criteria optimization phase, in which all components are co-operating to ensure that the

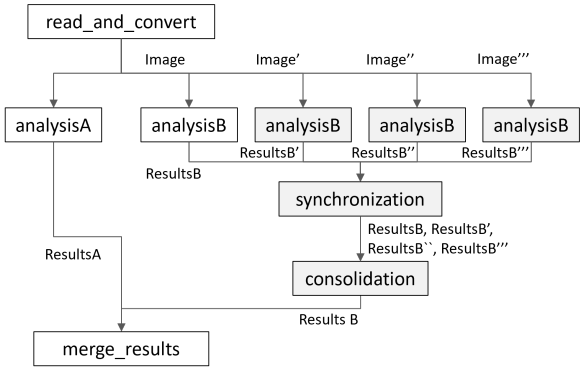


Figure 4: TDG when replicating $analysisB$.

system fulfils all requirements modelled in the DSML. During the analysis stage, two concurrent analysis are performed: a timing analysis (section 4.1) and a energy consumption analysis (section 4.2). During this stage, the analysis is performed for each TDG in the configuration file. The main purpose of the analysis stage is to annotate the TDGs with metrics that will be used in each target optimization phase. This stage does not change the configuration file, but it annotates the TDG files with new metrics. After the analysis, an optimization algorithm (section 4.3), is responsible to perform the multi-criteria optimization while each target optimization cannot find the most suitable configuration.

4.1 Timing analysis

The timing analysis phase of the optimization flow focuses on determining execution time metrics and the association of performance counters information, using measurement-based approaches, across all potential configurations and variants. The analysis is both for tasks as well as for the TDG as a whole.

For the tasks' execution time, the phase calculates the worst-case (WCET) and average execution times, which are then used in the optimization phase. It also determines fine grain information from performance counters (e.g. cache accesses, misses, etc.), which can be used for more detailed profiling of the applications.

Metrics related to a TDG use both existing information on the TDG and the new metrics calculated per task (the WCET of each task). The set of metrics outputted for the multi-criteria optimization flow includes volume, critical path length, potential maximum parallelism and average and worst-case makespan.

The volume corresponds to the aggregated WCET of all tasks within the TDG, whereas the critical path length signifies the overall cost of the path in the TDG that includes the longest route from the source task to the sink task.

The potential maximum parallelism serves as a metric that denotes the theoretical maximum parallelism achievable within a TDG, without considering any costs. It determines the highest number of tasks that could theoretically operate in parallel from all potential non-dependent siblings, even if they may not be executed simultaneously in practice.

The makespan represents the actual execution time of a TDG, spanning from the initiation of execution to its completion. This metric is pertinent when considering a specific task-to-thread mapping, as it provides the execution time of the TDG by taking into account the WCET of all tasks, given a certain number of available threads. The critical path length of a TDG can be regarded as the minimum duration the makespan of the TDG can take.

The timing analysis can be done also to reevaluate the TDGs, taking into account the additional information the optimization phase adds to the graph. It is also possible to use the analysis phase to optimize only for the time dimension, exploring different mapping algorithms, with an heuristic-based mapping approach [13].

4.2 Energy analysis

The aim of the energy analysis phase in the AMPERE multi-criteria optimization flow is to annotate the TDG with energy consumption information for each given task. Such energy consumption information relies on measurements from the profiling step embedded in the flow, as in figure 1. The energy annotations are then employed in the optimization step (section 4.3) to find the desired trade-off between energy consumption and execution time at the TDG level.

The AMPERE framework targets modern, heterogeneous, highly parallel systems. Measuring the energy consumption of a given workload running on these platforms is usually challenging for several reasons. Energy measurements require expensive external equipment, which impacts the scalability and flexibility of the system. Some platforms come equipped with built-in current sensors which can be used to estimate power consumption. However, analog sensors are slow with respect to the GHz regime of the digital hardware, and they can only provide coarse-grained measurements of the device sub-systems (e.g., entire CPU or GPU).

AMPERE's energy analysis is based on an approach to power modelling driven by the hardware performance monitoring counters (PMCs) of the target platform [14]. PMCs allow us to independently model the platform's power consumption at an arbitrary degree of granularity. Deriving directly from the digital domain, PMC-based power models expose high responsiveness, and match the workload execution in a reliable way, nevertheless impacting the optimization flow runtime with an extremely low overhead.

Thanks to a platform profile obtained in a one-time characterization step preceding the multi-criteria optimization, we calibrate our PMC-based power models to the desired target platform. Subsequently, we model each individual sub-system of the platform, at each one of its possible operating DVFS frequencies, with a set of representative PMCs coming from the platform characterization. The required PMCs are then sampled, in a low-overhead and non-invasive way, during the profiling in figure 1. Such measurements are employed in the energy analysis phase to estimate the average power and energy consumption of each task in the TDG.

Reliably supporting different devices and frequencies, the energy analysis step annotates each task's energy consumption

for each one of the task's functional specializations, and for each operating frequency of the platform.

AMPERE's energy analysis is fully automatized and flexible. Its data-driven nature requires minimal manual intervention: the best PMCs to model each sub-system of the target platform, at each frequency, are selected based on their correlation to their sub-system's power consumption. Additionally, requiring minimal architectural knowledge of the underlying hardware, the approach is easily extensible to additional devices.

4.3 Optimization approach

Starting from the annotated TDGs produced by the timing/energy analysis phases, the AMPERE workflow optimizer component [15] finds the system's optimal configuration. Considering the multi-criteria requirements of AMPERE, the developed tool supports two main optimization objectives.

The first objective is to find the system configuration that minimizes the target application's average energy consumption while preserving the system designer's timing constraints and, alternatively, maximizing the timing robustness of the application without exceeding its energy consumption budget.

The optimizer relies on a precise structural mathematical model of the target application, automatically derived from the TDG, that takes into account the resiliency requirements and the effects of selecting one or multiple heterogeneous hardware components of the target platform to perform (part of) the computations to find the optimal system configuration. The optimization is performed by applying mixed-integer quadratic constraint programming (MIQCP) and selecting one (or both) of the above objectives.

While the MIQCP formulation provides the requested optimality guarantees to generate the "optimal" configuration (final TDG, in fig. 1), it is also very complex, which may hinder its applicability to large-scale problems comprising more complex applications. For this reason, the optimizer also implements some simpler heuristic solvers that can be used to find sub-optimal configurations that may be fed back into the AMPERE loop or used as a starting point by the optimal MIQCP-based solver.

The final output of the optimization loop is the configuration of the target system in its entirety, including its multiple heterogeneous components (e.g., FPGA/GPU accelerators), and the placement of the individual runnables of each AMALTHEA task comprising the target application, including the choice between software and hardware implementation for tasks that can be hardware-accelerated, either for reaching the optimum (e.g., lowest average energy consumption) or because it is necessary to satisfy the application resiliency requirements.

5 Conclusions

This paper presented the approach used in the AMPERE project, to analyse and optimize the configuration of parallel real-time applications, in heterogeneous platforms, considering non-functional properties such as fault-tolerance, energy-efficiency and response time. The paper provides the project's multi-criteria optimization flow, which targets OpenMP parallel applications, presenting how each of the properties is considered in AMPERE ecosystem, and how the different dimensions are integrated in a single ecosystem.

6 Acknowledgments

This research has been co-funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 871669, in the context of the AMPERE project.

References

[1] E. Quiñones, S. Royuela, C. Scordino, P. Gai, L. M. Pinho, and L. Nogueira et al., "The ampere project: : A model-driven development framework for highly parallel and energy-efficient computation supporting multi-criteria optimization," in *2020 IEEE 23rd Intl. Symposium on Real-Time Distributed Computing (ISORC)*, pp. 201–206, 2020.

[2] Eclipse, "APP4MC." `https://www.eclipse.org/app4mc`, 2023.

[3] AMPERE, "D1.3. first release of the meta model-driven abstraction release," 2020.

[4] AMPERE, "D2.2. first release of the meta parallel programming abstraction and the single-criterion performance-aware." `https://ampere-euproject.eu/results/deliverables`, 2021. Accessed: 06-03-2023.

[5] OpenMP Arquitecture Review Board (ARB), "OpenMP Application Program Interface v5.2." `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf`, 2021.

[6] AMPERE, "D3.3. energy optimisation framework, predictable execution models and analysis, and software resilient techniques," 2022.

[7] AMPERE, "D2.3. programming model extensions and the multi-criteria performance-aware component." `https://ampere-euproject.eu/results/deliverables`, 2023.

[8] AMPERE, "Extended APP4MC SLG." `https://gitlab.bsc.es/ampere-sw/wp2/amalthea`, 2023.

[9] Open Source Robotics Foundation (OSRF), "ROS2." `https://github.com/ros2`, 2023.

[10] C. Yu, S. Royuela, and E. Quiñones, "Taskgraph: A low contention openmp tasking framework," *arXiv preprint arXiv:2212.04771*, 2022.

[11] AMPERE, "Extended LLVM 16.0." `https://gitlab.bsc.es/ampere-sw/wp2/llvm`, 2023.

[12] A. Munera, S. Royuela, and E. Quiñones, "Fault-tolerant applications through openmp," in *Proceedings of the 10th International BSC Severo Ochoa Doctoral Symposium*, 2023.

[13] M. S. Gharajeh, S. Royuela, L. M. Pinho, T. Carvalho, and E. Quiñones, "Heuristic-based task-to-thread mapping in multi-core processors," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–4, IEEE, 2022.

[14] S. Mazzola, T. Benz, B. Forsberg, and L. Benini, "A data-driven approach to lightweight dvfs-aware counter-based power modeling for heterogeneous platforms," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 22nd International Conference, SAMOS 2022, Samos, Greece, July 3–7, 2022, Proceedings*, pp. 346–361, Springer, 2022.

[15] T. Cucinotta, A. Amory, G. Ara, F. Paladino, and M. D. Natale, "Multi-criteria optimization of real-time dags on heterogeneous platforms under p-edf," *ACM Trans. Embed. Comput. Syst.*, apr 2023. Just Accepted.