Combined Data Transfer Response Time and Mapping Exploration in MPSoCs

Alexander Diewald, Simon Barner
Model-based Systems Engineering
fortiss GmbH
80805 Munich, Germany
lastname@fortiss.org

Selma Saidi
Institute of Embedded Systems
Hamburg University of Technology
21073 Hamburg, Germany
selma.saidi@tuhh.de

***Abstract*—Recent embedded applications such as Autonomous Driver Assistance Systems (ADAS) require large computational resources that increase the need for HW accelerators, e.g., in system-on-chip-based platforms. Synthesising optimal task/data mappings and schedules for such platforms becomes increasingly challenging, even more in safety-critical contexts. For designing real-time heterogeneous systems, response time computation and the resolution of task mapping problems are required as demanded in the WATERS 2019 challenge. Our contribution to address the challenge is to extend a design space exploration (DSE) formulation of mapping applications on MPSoCs architectures to consider DMA-based data (pre)-fetching. The approach is performed in two steps. First, we determine task mappings to a heterogeneous MPSoC platform using a multi-objective evolutionary algorithm (MOEA)-based DSE. In order to check the feasibility of an allocation, and to rate its quality, we use a SMT solver to construct schedules whose latency is close to the achievable minimum. Our task response time analysis considers the effects of memory access times and DMAs to supply the SMT scheduler with data fetching latencies. The MOEA-DSE, the SMT scheduler, and the response time calculation are integrated into the AutoFOCUS 3 tool that has been extended with an importer for the AMALTHEA model that specifies the challenge use case.**

***Index Terms*—Real-time systems, Design optimization, Model-driven development**

I. Introduction and Problem Statement

Over the last decades, embedded system applications and platforms became more complex, resulting in additional hardware (HW) accelerators in typical commercial off-the-shelf (COTS) processors such as system-on-chips. However, despite the high amount of available computation capabilities, data communication often constitutes the limiting factor for performance where the required data needs to be transferred timely between remote off-chip memories and MPSoCs on-chip memories. As a result, data transfers become an important aspect in safety-critical systems besides the proper partitioning of tasks. In the WATERS 2019 challenge, a shared-memory-based heterogeneous system is considered where the separation of data in the memory, and its access times are relevant for avoiding temporal overruns and for optimizing the system schedule. To optimize such a system, we use a DSE that combines MOEA with a SMT solver to investigate the effects of the needed synchronization between computations and data communications onto the system's performance. .

This research has received funding from the German Federal Ministry of Education and Research (BMBF; grant agreement Nº 01IS16025F) in the ARAMiS II project (www.aramis2.de).

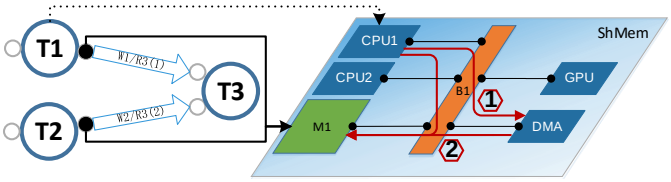


Fig. 1: Write operation of Task T1 for DMA-only communication: 1) Instruct DMA; 2) DMA write operation.

DSE has a long history in embedded system design, e.g., on the system-level [1], [2]. These techniques were connected to Model-Driven Engineering (MDE), e.g., in [3], where a cross-domain first-order constraint language is constructed to describe exploration problems. Similar approaches are found in [4], [5], where a constraint solver is combined with simulated annealing to produce optimized system designs, e.g., task-to-core allocations. On the device-level, Benini et al. [6] present a simulation-based design-space exploration that is coupled with a SystemC model.

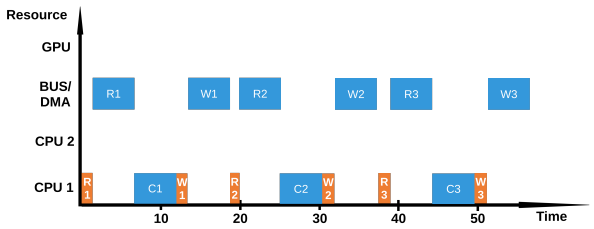
In our proposed approach, we consider for every task an *AER* model of execution that includes an *Aquisition* (read) phase required to copy data from an external memory location, typically a DRAM, to a local CPU/GPU memory location, an *Execution* phase where task kernels are executed on the CPU or GPU, and a *Restitution* phase where data is written back to main memory. In addition to the assumptions of the challenge, we suppose that all memory access are exclusively handled by DMA transfers (see Sec. II-C for details). The communication scheme illustrated in Fig. 1 is equivalent to read and write operations where a DMA is first instructed before it reserves the bus for the actual transfer.

DMAs are often used in MPSoCs as HW support for transferring data. They offer asynchronous transfer operations that relieve the main processor from explicitly managing them. Otherwise, the processor would be blocked during this time, but idling most of it, waiting for the memory operations to complete. Instead, whenever a processor needs data, it issues a transfer request to the DMA that then manages the copy operation during which the processor can be allocated to other tasks to be executed. When the transfer is completed, the

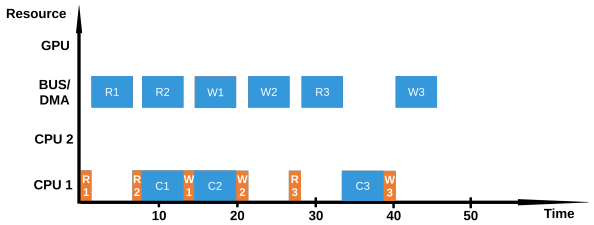
DMA notifies the main processor that can then resume the processing task that issued it. Computations and data transfers can therefore be performed in parallel in a pipelined execution. This technique is extensively used for data prefetching where before performing a task, the processor can issue a transfer request to the DMA to fetch data required by the next task to be processed. Prefetching techniques are used to hide the transfer latency and therefore optimize the overall performance of the system.

In our contribution, we focus on the two following effects:

- *Synchronization between computation and data transfers, especially the latencies introduced by communication.* When offloading a task to a GPU accelerator, additional data transfers can be compensated by shorter computation times to optimize system performance. To optimize the overall performance, we jointly explore task mappings and data transfers strategies that are strongly coupled via common communication resources.
- *Increasing system performance by using prefetching techniques for schedule construction.* Prefetching data using DMAs allows exploiting parallelism more efficiently since the main processor cores do not need to idle while getting data from a slow memory. Instead, the cores can access the data from a closer and faster memory to which it has been transferred asynchronously by a DMA engine. Fig. 2a and 2b illustrate the benefits of prefetching (reduction of overall latency).



(a) Schedule without prefetching.



(b) Schedule with prefetching.

Fig. 2: Temporal effect of data prefetching.

II. Approach

We address both tasks of the challenge by combining a MDE tool, its MOEA-based DSE algorithm [7], an SMT scheduling synthesis formalization [8], and a temporal analysis specialized on DMA-based shared memory platforms [9]. Based on an imported version of the provided AMALTHEA [10] challenge model[1], the DSE is used to map tasks to CPU cores or the GPU, and and data transfers to the DMA unit. For each candidate mapping, the SMT scheduler and the temporal analysis are invoked to estimate the resulting communication latencies.

A. System Modelling

Our approach builds on MDE to represent the DSE's input models and to encode the resulting solution models. We use the infrastructure as well as the modelling viewpoints and metamodels of the open source AutoFOCUS 3 (AF3)[2] tool [11] (e.g., logical and technical viewpoint including a hierarchical platform metamodel [12]).

We created an importer that translates the AMALTHEA *runnables*, *tasks*, *labels*, *label accesses*, as well as the HW architecture and timing information, e.g., periods, worst-case execution times (WCETs), to the corresponding AF3 artefacts.

AutoFOCUS 3 relies on logical component and task networks whose entities communicate by means of typed ports that are connected by channels/signals, to provide platform-independent and implementation-specific descriptions of a system's architecture, respectively. In contrast to AF3's more system-level oriented view, AMALTHEA puts more emphasis on the device-level and models the communication of its runnables and tasks by means of shared variables. This is considered when AMALTHEA runnables are converted into AF3 components, AMALTHEA tasks into AF3 tasks, and label (accesses) into ports and their types respectively. Data-dependencies between AMALTHEA tasks whose runnables write/read the same labels are encoded as AF3 logical channels and signals between AF3 components and tasks, respectively. In order to "mimic" global AMALTHEA variables, our importer converts them into separate AF3 components (tasks) that are connected to components (tasks) that read from or write to them.

Moreover, the mapping of runnables to tasks is transformed into an AF3 mapping table. In order to account for the AER scheme, the importer creates dedicated AF3 read and write tasks for each task present in the AMALTHEA model. By mapping these tasks directly to the CPU cores or to a DMA unit, different memory access patterns can be explored (e.g., DMA-based data-prefetching in parallel to the functional computation tasks).

Since the AMALTHEA HW metamodel allows to arbitrarily nest structural elements, it is not possible to define a generic approach that automatically maps instances to the hierarchical AP3 platform metamodel that supports a 4-tier architecture consisting of clusters, nodes (ECUs), tiles (processors), and cores [12]. Hence, we opted to use the manually crafted platform model shown in Fig. 3 that is composed of several execution units (red), busses (blue) and a memory unit (RAM, green). The execution units consist of four A57 cores (top-left), two Denver cores (bottom-left), two DMA "cores" (middle), as well as a GP10b GPU "core" The busses are annotated with bandwidth information that is derived from the AMALTHEA model. For the read and write tasks, two

[1]https://www.ecrts.org/forum/viewtopic.php?f=43&t=126&sid=8cc2f7c06f29e09b25476b714d191b3d

[2]https://af3.fortiss.org/

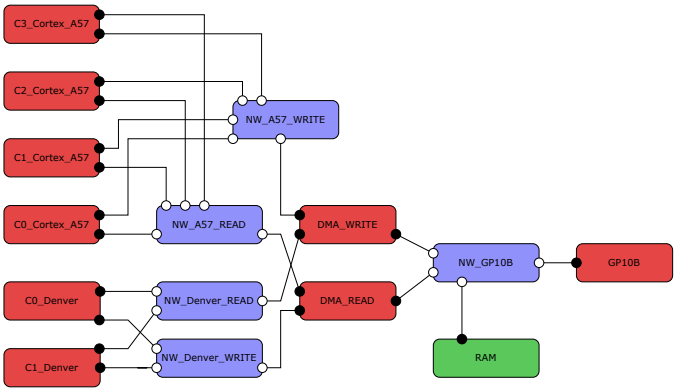


Fig. 3: Handcrafted HW platform model (exported from AF3).

distinct DMA units have been modelled. While this does not accurately model the Jetson TX 2 platform, this modelling approach is tailored to our DSE algorithm in order to enforce inter-core communication between read and write tasks.

B. DSE Interface and Overview

Our MOEA-based DSE [7] explores the allocation of tasks to (heterogeneous) execution units at varying granularity (CPUs, cores, ...) and considers constraints and objective functions to rate the quality of candidate solutions. Constraints and objectives are passed to the DSE as declarative DseML [3] expressions via an exploration metamodel [13]. AF3's DSE perspective enables i) to define optimization problems in terms of the aforementioned DseML using pattern-based editors, ii) to launch and monitor the DSE execution, and iii) to export the results of an exploration to the corresponding AF3 models.

Our DSE algorithm is configured for the challenge to first allocate tasks to execution units (cores), then to generate the required messages for inter-core communication, and finally to schedule the tasks and messages on their resources.

- **Allocation.** The MOEA algorithm assigns tasks to execution units under the consideration of direct constraints such as allocations to specific units or joint allocations of tasks to specific execution units.
- **Messages & Routing.** First, the required messages are calculated based on decisions taken in the task allocation step. Then, the corresponding shortest-path routes along the resources of the underlying platform are determined.
- **Scheduling.** Lastly, the aforementioned SMT scheduler calculates the starting times of tasks and messages, resulting into a strictly time-triggered schedule. It is called repeatedly to approximate the minimal achievable latency of the overall system by means of bisection.

All these steps are performed iteratively within an outer loop governed by the MOEA algorithm such that solutions consisting of the above encodings are optimized. This decomposition has been chosen to combine the flexibility and performance of MOEA algorithms for generic optimization problems with the suitability of SMT solvers in solving combinatorial constraint satisfaction problems such as scheduling. Moreover, a complete unified problem formulation in either of these algorithms would result in sub-optimal performance.

C. Assumptions and Limitations

In addition to the challenge specification, we make the following assumptions to reduce the use case's complexity.

Firstly, we suppose that all memory accesses are performed by DMA requests, favouring solutions that benefit from data prefetching. Here, we neglect interference effects on the latency of the DMA data fetching. We also assume that the bandwidth of the DMA accesses equals those of the GPU since both would access the system RAM over the memory fabric. Nevertheless, the DMAs connection to the fabric is established over the system fabric (i.e., peripheral bus) whose interferences are not considered. Secondly, we do not consider the different task periods specified in the challenge model, but only a reduced single-rate version in order to compensate scalability issues of our current SMT scheduler implementation in the multi-rate case. Thirdly, our scheduler does not support task preemption. However, since the better part of long-running tasks could not be preempted anyways since they are offloaded to the GPU accelerator, we argue that the impact of this assumption onto the resulting overall latency is limited.

III. Design Space Exploration

The dependencies between the involved input and output artefacts drive our MOEA-based DSE, fostering configurability w.r.t. the engineering task at hand and reusability of exploration features. This architecture of the exploration framework allows optimizing non-convex and multi-objective problems where variable couplings are encoded by dependencies.

a) Decision Variables: In our approach, there exist two major decision variables: the allocation of tasks to cores and the starting time of tasks. In order to express allocations, we define a selection vector $\mathbf{a}_i$ whose rows correspond to the execution unit to which a task τ_i shall be allocated. This vector has a one entry in the row corresponding to the allocation target, otherwise a zero. Furthermore, we optimize the starting time $\mathrm{t}_{\mathrm{start},i}$ of each task in the SMT scheduler.

b) Objective function: It minimizes the overall latency per core e_i (here: ARM A57, Denver, GPU, DMA) to emulate a race-to-idle approach for minimizing the energy consumption by reducing wakeups and enabling lower power states:

$$\min_{\forall e_j \in \mathcal{E}} \quad \max_{\forall \tau_i \in \mathcal{T}^{e_j}} (t^{e_j}_{\mathrm{end,i}}) \tag{1}$$

where the set $\mathcal{E}$ is the set of cores, while $\mathcal{T}^{e_i}$ is the set of tasks composed of the read, execute, write tasks allocated to the execution unit e_i ($\mathcal{T}$ is the set of all tasks).

A. Spatial Constraints

For the task mapping problem, we define a matrix $\mathbf{A}$ constructed from DseML allocation constraints that represents the allowed allocations of tasks to cores such that

$$\mathbf{A}_{\mathrm{alloc}} = \begin{bmatrix} 1 & 1 & \dots & 0 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}, \tag{2}$$

where tasks correspond to rows and tiles to columns. In the challenge, the entries for CPU-only tasks are one for all A57 and Denver cores, and zero otherwise. The same holds for the respective GPU-only and DMA read and write entries. Only the *Lane_Detection*, *Location*, and *SFM* tasks have entries for the GPU "core" and the A57 and Denver cores. Consequently,

$$\mathrm{u}_i^T \mathbf{A}_{\mathrm{alloc}} \mathbf{a}_i = 1, \quad \forall \tau_i \in \mathcal{T} \tag{3}$$

describes the allowed allocations that are already considered in the mapping operators of the DSE that construct solution candidates. The unit vector u_i selects valid allocations from the matrix $\mathbf{A}_{\mathrm{alloc}}$ and $\mathbf{a}_i$ is the allocation vector of a task τ_i to an execution unit with index i.

Complementing the allocation constraints, a feasible solution contains only messages whose corresponding signals fulfil the reachability constraint

$$\underset{r \in \mathcal{R}}{\exists} r(\mathrm{f}_{\mathrm{send},a}(m_i)^T \mathbf{e}, u_j^T \mathrm{f}_{\mathrm{recv},a}(m_i)\mathbf{e}), \quad \forall m_i \in \mathcal{M}, \forall j \in 1, \ldots, \mathrm{R} \tag{4}$$

where $\mathcal{M}$ is the set of exchanged signals, and R is the number of receivers of a signal, and $\mathbf{e}$ is the vector of execution units. The functions $\mathrm{f}_{\mathrm{send},a}(\ldots)$ and $\mathrm{f}_{\mathrm{recv},a}(\ldots)$ extract the sender allocation vector and the receiver allocation matrix, where each column corresponds to a receiver task, respectively. Thus, the function $r(\ldots)$ takes a sending and receiving resource of a message m_i as input and returns the routes connecting them. Hence, equation 4 states that for each message, given an allocation table, a route connecting the sender with its receivers must exist. The constraint is implicitly considered by the DSE when calculating routes using a shortest path algorithm. More spatial constraints (e.g., safety-related separation constraints) can be considered by the DSE, but the focus of this work is on the integration of the SMT scheduler as well as the shared-memory system type with DMA units.

B. Temporal Constraints

The SMT scheduler's constraints are based on [8]. Here, we focus on discussing the periodicity, causality, and non-overlapping constraints for tasks, and provide a more compact formulation for reasoning about prefetching techniques and AER characteristics. Periodicity is also evaluated by the outer MOEA-DSE optimization loop to propagate potential constraint violations to the candidate construction logic for the next iteration. The periodicity constraint is defined by

$$0 \leq t_{\mathrm{end},i} \leq \mathrm{p}_i, \quad \forall \tau_i \in \mathcal{T} \tag{5a}$$

$$0 \leq t_{\mathrm{start},i} \leq \mathrm{p}_i, \quad \forall \tau_i \in \mathcal{T} \tag{5b}$$

such that each task τ_i must be started and finished within the bounds established by the period constraints. Additionally,

$$t_{\mathrm{end},i} = t_{\mathrm{start},i} + C_{\mathrm{dur},i}, \quad \forall \tau_i \in \mathcal{T} \tag{6}$$

must hold. The constraints 5b and 6 are implicitly handled and we consider a task's WCET as its duration $C_{\mathrm{dur},i}$. These constraints can be easily extended to the multi-rate case if we consider task instances instead of tasks and assign an instance index, e.g., j to them. Furthermore, the bounds of the equations 5a and 5b have to be aligned by the j-th periods such that the constraints become $j\mathrm{p}_j \leq \ldots \leq (j+1)\mathrm{p}_j, \ \forall j \in 0, \ldots, J-1$.

Data dependencies between scheduled tasks constrain only the SMT scheduler and can be expressed by

$$t_{\mathrm{end},j} \leq t_{\mathrm{start},i}, \quad \forall j \in \mathcal{T}_{\mathrm{pred},i} \forall \tau_i \in \mathcal{T} \tag{7}$$

where $\mathcal{T}_{pred,i}$ are the predecessor tasks of a task τ_i, $t_{\mathrm{end},j}$ are the ending times of the predecessor tasks, and $t_{\mathrm{start},i}$ the starting times of the regarded task. Complementary, the non-overlapping constraint

$$t_{\mathrm{end},j} \leq t_{\mathrm{start},i}, \quad \forall \tau_i \in \{\mathcal{T} | \mathrm{u}_i^T \mathbf{A}_{\mathrm{alloc}} \mathbf{a}_i = 1 \wedge t_{\mathrm{start},j} \leq t_{\mathrm{start},i}\} \tag{8}$$

must hold to avoid temporal resource conflicts on execution units. Data transfers are represented by the acquire and restitution tasks defined in the next section.

C. Schedule Optimization and Prefetching

For the schedule optimization, we are optimizing the starting time of tasks to minimize the overall system schedule duration. Since the SMT scheduler is embedded in a MOEA algorithm that forms the main optimization loop, its runtime is critical for the performance of the algorithm as a whole. Thus, we utilize the fact that SMT-based solvers are very fast at detecting infeasibilities and finding solution for constraint satisfaction problems, whereas additional optimization is costly. Therefore, we construct an additional constraint on the duration of the system schedule that we pass to the SMT solver (Z3). Initially, an upper bound is selected that is equivalent to the overall schedule length if all tasks are scheduled onto a single core. Starting from this bound, a binary search is performed that decreases or increases the duration bound based on the respective feasible or unfeasible state reported by the solver (schedule latency bisection). In case of a timeout (5 s), an already found schedule is taken as the solution, or it is considered infeasible by the MOEA optimization loop otherwise. The search is stopped if the step size is smaller than 1% of the initially calculated upper bound.

Now that the SMT scheduler optimizes the starting times of all tasks, including all acquisition, execution, and restitution tasks, prefetching mechanisms can be considered. Considering the *AER* execution scheme, the relations

$$t_{\mathrm{end},i}^{\mathrm{acq}} \leq t_{\mathrm{start},i}^{\mathrm{exe}}, \quad \forall \tau_i \in \mathcal{T}^{\mathrm{exe}}, \tag{9}$$

$$t_{\mathrm{end},i}^{\mathrm{exe}} \leq t_{\mathrm{start},i}^{\mathrm{res}}, \quad \forall \tau_i \in \mathcal{T}^{\mathrm{exe}} \tag{10}$$

must hold, where $t^{acq}, t^{exe}, t^{res}$ correspond to the acquire, execute, and restitution tasks and $\mathcal{T}^{exe} \subset \mathcal{T}$ the set of execution tasks. Combining this with equation 7, we can derive that

$$t_{\mathrm{end},j}^{\mathrm{res}} \leq t_{\mathrm{start},i}^{\mathrm{acq}}, \quad \forall j \in \mathcal{T}_{\mathrm{pred},i} \forall \tau_i \in \mathcal{T} \tag{11}$$

must hold. The times t^{acq} and t^{res} are the times required to read data from and write data to the memory.

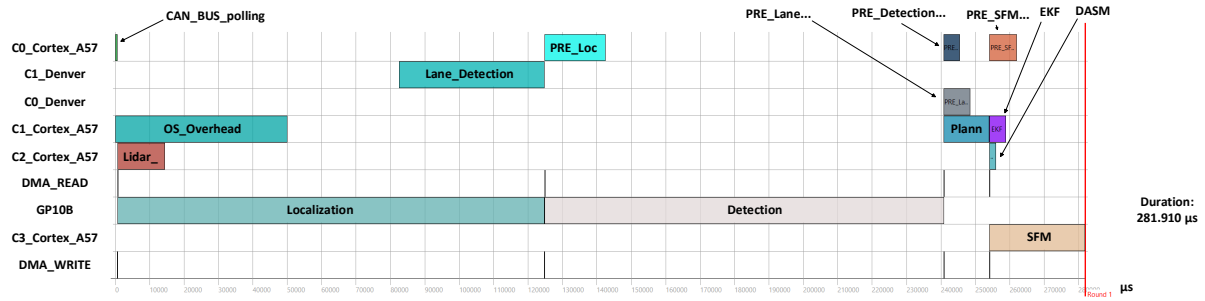


Fig. 4: Solution: Gantt chart of task schedule with DMA-supported data pre-fetching (AF3 screenshot with additional labels).

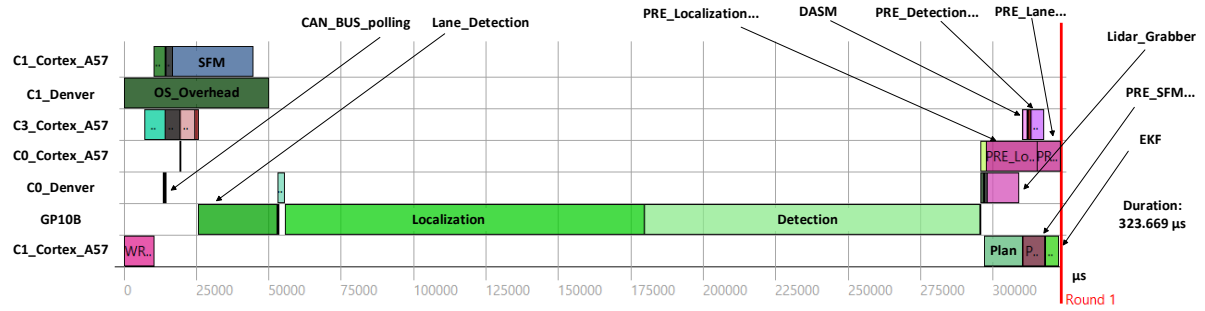


Fig. 5: Solution: Gantt chart of task schedule without data pre-fetching (AF3 screenshot with additional labels).

Using the notation of arrival times, i.e., the time a task could be started independently of resource usage, we can define

$$t^{\text{res}}_{\text{end},j} = t^{\text{acq}}_{\text{arr},i}, \quad \forall j \in \mathcal{T}_{\text{pred},i} \forall \tau_i \in \mathcal{T} \tag{12}$$

and thus derive

$$t_{\text{arr},j} = t_{\text{start},i} + \epsilon, \quad \forall \tau_i \in \mathcal{T}, \tag{13}$$

where ϵ is the parameter that is effectively minimized by the solver. Since it is present for any task, the solver tries to maximize parallelism purely under consideration of constraints 7 and 8. In general, there would be a trade-off with the neglected communication-induced latencies.

D. DMA-based Latencies

We start with a simplified version of the latency estimation of DMA transfers defined in [14]. It depends on i) the initialization overhead that is fix for every DMA request and that defines the time required by the DMA controller to initiate a transfer, ii) the size of the transferred data and operation mode of the DMA such as the burst mode that is particularly useful for large data transfers, and iii) the target memory location where longer distances negatively affect the latency.

Let $L^+_{i,dma}$ denote the maximum time required for a DMA request i to perform a transfer. It is defined by

$$L^+_{\text{DMA},i}(G_i) = I + L^+_{\text{mem},i}(G_i), \tag{14}$$

where I is the DMA initialization overhead, G_i is the granularity of transfers (i.e., number of packets composing the transfer) and $L^+_{\text{mem},i}$ is an upper bound on the latency of the target memory location. In this work, we define

$$L^+_{\text{mem},i}(G_i) = \sum_{res_{\text{c},i} \in r(\mathbf{a}_i^T\mathbf{e}, res_{\text{mem}})} \frac{1}{res^{\text{BW}}_{\text{c},i}} + L^+_{\text{intf}}(G_i, \mathbf{a}_i^T\mathbf{e}), \tag{15}$$

where $res_{\text{c},i}$ are the communication resources traversed by a route to the memory hosting the required data. $res^{\text{BW}}_{\text{c},i}$ is the bandwidth of these communication resources. Additionally, $L^+_{\text{intf}}(G_i, \mathbf{a}_i^T\mathbf{e})$ defines the maximum latency that a memory access i may experience. Thus,

$$t^{\text{acq/res}}_{\text{end},j} = t^{\text{acq/res}}_{\text{start},i} + \frac{size(t_i^{\text{acq/res}})}{G_i} \cdot L^+_{\text{DMA},i}(G_i), \quad \forall t_i^{\text{acq/res}} \in \mathcal{T}^{\text{acq/res}} \tag{16}$$

defines the duration of every acquisition and restitution task that uses a DMA unit to fetch its data.

IV. Experimental Results

Applying the DSE to the imported and slightly modified challenge model, the exploration algorithm produced valid and optimized schedules in three hours. Although the running time of the algorithm is fairly long, the iterative nature and internal operation allows retrieving valid solution already after the first iteration. Here, the SMT scheduler is the dominating factor in the optimization. Reducing the timeout of the SMT call, or lowering the granularity of the outer binary search (termination criterion) at the expense of accuracy of the identified schedule promises to reduce the needed time. For the MOEA algorithm, we have chosen a population size of 100 (α) and an offspring size of 20 individuals (μ) using elitism (fewer parent than offspring individuals) and a crossover rate of 0.3.

Fig. 4 and 5 show optimized schedules synthesized by the DSE, with DMA-based prefetching and without, respectively. As expected, the schedule without prefetching has a longer duration. For the *Localization*, *Lane_Detection*, and *SFM* tasks both a CPU and GPU implementation are available. *Detection* is fixed to the GPU. Allocating the long-running *Localization*

task to the GPU seams reasonable, as its at takes thrice the time execute on a CPU. The two solutions differ in the allocation of *Lane_Detection*. The DMA-based schedule exploits parallelism, which, however, would not lead to a shorter duration in the no-DMA case. The mapping of the *SFM* task is disadvantageous in both cases as it would have a shorter duration if mapped to the GPU. This might be resolved by additional MOEA iterations, a modified stopping criterion for the latency bisection, and/or an increased SMT solver timeout.

The shared off-chip memory appears to form the bottleneck of the system's platform. However, the data transfer times are almost negligible compared the task execution times. This results from the assumption that DMAs can access the memory with the same bandwidth as the GPU. Furthermore, we do not consider any interferences from cores as all tasks are defined to use the DMA for data transfers such that no conflicts can occur. We assume that DMA initialization latency is negligible, on the one hand because of the large amount of transferred data, and on the other hand because we neglect multi-rate scheduling for which an initialization would be required for each of the task instances. Overall, even if the data transfer latencies would be increased by, e.g., DMA initialization costs or additional interferences, the effect would be barely noticeable due to the dominance of computation times and DMA-based data fetching parallelization. Also, data prefetching is performed only once per task such that a large local memory would be needed. This could be resolved by splitting up tasks that require large data sets.

V. Conclusion

Our contributions are many-fold: We present an importer to automatically convert device-level AMALTHEA models into system-level AutoFOCUS 3 models. We also extend a MOEA-DSE framework that integrates with the AutoFOCUS 3 MDE tool to handle such models. In order to jointly optimize task allocations and the corresponding time-triggered schedules, we integrate a Z3-based SMT scheduler into the MOEA-DSE. Lastly, the DSE framework considers both task and DMA scheduling to optimize the response time of real-time tasks.

In future work, we will improve the scalability of SMT-based multi-rate scheduling and integrate into our implementation. Furthermore, we aim at combing the DSE framework with more elaborated system-level timing analysis tools like [15] in order to directly incorporate more complex response time analyses into the exploration's schedule synthesis phase. Finally, we want to improve the integration of device-level with system-level modelling to reduce the semantic gap between these levels, e.g., by combining point-to-point communication with shared-memory-based communication.

References

[1] Blickle, T., Teich, J., and Thiele, L., "System-level synthesis using evolutionary algorithms," *Des. Autom. Embed. Syst.*, vol. 3, no. 1, pp. 23–58, Jan. 1998.

[2] Thaden, E. M., "Semi-automatic optimization of hardware architectures in embedded systems," PhD thesis, Universität Oldenburg, 2013.

[3] Eder, J., Zverlov, S., Voss, S., Khalil, M., and Ipatiov, A., "Bringing DSE to life: Exploring the design space of an industrial automotive use case," in *2017 ACM/IEEE 20th Int. Conf. Model Driven Eng. Lang. Syst. (MODELS)*, Sep. 2017, pp. 270–280.

[4] Hilbrich, R. and Behrisch, M., "Improving the efficiency of dislocality constraints for an automated software deployment in safety-critical systems," in *Comb. Proc. Workshops German Softw. Eng. Conf.*, 2018, pp. 90–95.

[5] Zimmermann, A., Maschotta, R., Wichmann, A., and Hilbrich, R., "Optimization of systems with nested design space," in *Int. Syst. Conf. (SysCon)*, IEEE, 2018.

[6] Benini, L., Bertozzi, D., Bogliolo, A., Menichelli, F., and Olivieri, M., "MPARM: Exploring the multi-processor SoC design space with SystemC," *J. VLSI Signal Processing systems for signal, image and video technology*, vol. 41, no. 2, pp. 169–182, Sep. 2005.

[7] Migge, J., Balbastre, P., Barner, S., Chauvel, F., Craciunas, S. S., Diewald, A., Durrieu, G., Haugen, Ø., Seyed, A. A. J., Pagetti, C., Oliver, R. S., and Vasilevskiy, A., "Algorithms and tools," in *Distributed Real-Time Architecture for Mixed-Criticality Systems*, Ahmadian, H., Obermaisser, R., and Perez, J., Eds. CRC Press, Aug. 21, 2018, ch. 5, p. 98, ISBN: 978-0-8153-6064-3.

[8] Voss, S. and Schätz, B., "Deployment and scheduling synthesis for mixed-critical shared-memory applications," in *2013 20th IEEE Int. Conf. Workshops Eng. Comput. Based Syst. (ECBS)*, Apr. 2013, pp. 100–109.

[9] Saidi, S. and Syring, A., "Exploiting locality for the performance analysis of shared memory systems in mpsocs," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2018, pp. 350–360.

[10] Eclipse, *APP4MC*, https://www.eclipse.org/app4mc/.

[11] Aravantinos, V., Voss, S., Teufl, S., Hölzl, F., and Schätz, B., "AutoFOCUS 3: Tooling concepts for seamless, model-based development of embedded systems," *Joint Proc. ACES-MB 2015–Model-based Archit. Cyber-phys. Embed. Syst.*, p. 19, 2015.

[12] Barner, S., Diewald, A., Migge, J., Syed, A., Fohler, G., Faugère, M., and Pérez, D. G., "DREAMS toolchain: Model-driven engineering of mixed-criticality systems," in *2017 ACM/IEEE 20th Int. Conf. Model Driven Eng. Lang. Syst. (MODELS)*, Sep. 2017, pp. 259–269.

[13] Diewald, A., Voss, S., and Barner, S., "A lightweight design space exploration and optimization language," in *Proc. 19th Int. Workshop Softw. Compil. Embed. Syst. (SCOPES)*, ACM, 2016, pp. 190–193.

[14] Saidi, S., Tendulkar, P., Lepley, T., and Maler, O., "Optimizing explicit data transfers for data parallel applications on the cell architecture," *TACO*, vol. 8, no. 4, 37:1–37:20, 2012.

[15] Diemer, J., Axer, P., and Ernst, R., "Compositional Performance Analysis in Python with pyCPA," in *Proc. Int. Workshop Anal. Tools Methodol. Embed. Real-time Syst.*, 2012.