Optimal Memory Allocation and Scheduling for DMA Data Transfers under the LET Paradigm

Paolo Pazzaglia*, Daniel Casini†‡, Alessandro Biondi†‡ and Marco Di Natale†‡
*Saarland University, Saarbrücken, Germany †TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy
‡Department of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy

***Abstract*—The Logical Execution Time (LET) paradigm is increasingly used to achieve predictable communications in modern multicore automotive applications. Direct Memory Access (DMA) engines can perform the data copies that are needed in a LET implementation on behalf of the cores with improved parallelism and reduced overheads. However, each DMA transfer operates on contiguous memory areas, and the performance is strongly dependent on the allocation in memory of the variables to be copied. This paper proposes a protocol to perform LET communications with a DMA and presents an optimal memory allocation scheme and scheduling using a mixed-integer linear programming formulation. Experimental results are reported to compare the performance of different communication approaches.**

I. Introduction

Multicore platforms introduce many challenges in the design and development of safe and time-predictable applications. Achieving predictable and deterministic inter-core communication is one of them. Tasks typically communicate by moving data across globally-shared memories, which can be concurrently accessed from all cores. The access to these memories is typically optimized for average-case performance and not for time predictability. As a consequence, contention on memory accesses may severely harm the application timeliness, introduce nondeterminism, and possibly lead to faults.

The Logical Execution Time (LET) paradigm [1] recently received renewed attention thanks to its capability of providing time-deterministic communications on multicore platforms, and found particularly relevant by automotive developers [2]. Under LET, communications among tasks (i.e., memory accesses) shall occur at predefined time instants (e.g., the beginning and the end of the tasks' periods), hence providing an opportunity to avoid memory contention by explicitly scheduling memory accesses [3].

A possible option to implement LET communications is to introduce tasks running at the highest priority in each core that copy data between a private core-local memory (e.g., a scratch-pad) and the global memory [3]. This may not be a suitable solution when dealing with emerging autonomous driving applications, which require moving huge amounts of sensor data (e.g., camera images, lidar data, etc.). In these cases, Direct Memory Access (DMA) engines provide a convenient solution to manage communications, while allowing tasks to run in parallel. To minimize the processor intervention in programming the DMA, data needs to be allocated in *contiguous* memory areas, thus giving rise to the need for an optimized memory allocation for local and global memories.

This paper first proposes a DMA-based protocol to handle LET communication in multicore applications. Its goal is to minimize the data acquisition latency of each task, while addressing all causality constraints. An optimal schedule of the communications, together with the corresponding memory allocation scheme, is obtained using a mixed-integer linear programming formulation. Experimental results are reported, to compare different communication strategies.

II. Related Work

The LET execution model has been initially proposed in the Giotto framework [1] to enhance the determinism of control software. More recently, Hamann et al. [2] proposed the LET paradigm to preserve the causality in execution order when porting an automotive application from a single-core to a multi-core setting. Biondi and Di Natale [3] described different variants of the LET paradigm and implemented LET communication with predictable shared-memory communication on an Aurix Tricore TC275 [4]. Pazzaglia et al. [5] provided a formulation for the functional partitioning of a real-time application using the LET paradigm. No previous work considered LET with DMA engines. Orthogonally, several authors addressed the problem of loading data in a core local memory with DMAs [6]–[8]. For example, Rouxel et al. [9] proposed a method to reduce the communication delays using a DMA under static scheduling, without using LET. Other works considered the problem of bounding the memory-space requirement and memory allocation algorithms for hard real-time systems [10, 11]. Puaut and Pais [12] and Whitham and Audsley [13] proposed methods to achieve a predictable allocation in scratchpad memories. Overall, to the best of our knowledge, no prior work jointly considered the usage of DMA engines to load data into scratchpad memories for communicating tasks using LET.

III. System Model

A. Platform and Application Model

The platform considered in this paper consists of a set $\mathcal{P} = \{P_1, \ldots, P_N\}$ of N identical cores. Each core $P_k \in \mathcal{P}$ has a private, dual-ported [6, 8] local memory (e.g., a scratchpad). The platform also includes a global memory shared by all cores. The set of all memories is denoted with $\mathcal{M} = \{M_1, \ldots, M_N, M_G\}$, where the first N are local memories, and M_G is global. A DMA engine performs memory transfers between local and global memories. This setting is representative of real commercial platforms used in automotive systems, e.g., the AURIX TC2xx and AURIX TC3xx by Infineon [4], or other high-end platforms when using cache lockdown.

The application consists of a set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ of periodic real-time tasks, under partitioned scheduling. Each task τ_i is statically assigned to one processor $\mathcal{P}(\tau_i) \in \mathcal{P}$, and the subset of tasks assigned to the k-th processor is denoted by Γ_k. Each task τ_i is characterized by a period T_i and a relative (implicit) deadline $D_i = T_i$. All tasks are synchronously released at the system startup $s_0 = 0$ and release a potentially infinite sequence of instances called *jobs*. A task is deemed *schedulable* when each of its jobs completes before the release of the next job. Hereafter, we work under the hypothesis that the entire task set Γ is schedulable. A job is said to be *ready* when it has been released and all the data it requires (i.e., from inter-task communications) has already been provided in its core-local memory.

The *data acquisition deadline* γ_i of a task τ_i is the latest possible (relative) time when any job of τ_i may become ready (the time when it is actually available for execution) to preserve schedulability of τ_i. The maximum time elapsed between the release of any job of τ_i and when it becomes ready is referred to as *data acquisition latency* and denoted with λ_i. If the system is schedulable, we can safely restrict our problem to a time interval $[0, H)$, where H is the hyperperiod. The set of release instants of τ_i in $[0, H)$ is defined as $\mathcal{T}_i = \{t_{i,0},$

$t_{i,1}, \ldots, t_{i,N_i-1}\}$, with $N_i = H/T_i$, $t_{i,0} = s_0$ and $t_{i,j+1} = t_{i,j} + T_i$. The set of the release instants of all tasks is $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$.

B. Communication Model

Tasks access data stored in memory slots called *labels*. Each label ℓ_l is characterized by: **(i)** a size σ_l (in bytes), **(ii)** a memory $M_k \in \mathcal{M}$ to which ℓ_l is assigned to, and **(iii)** an address $a_{l,k}$ in M_k at which label ℓ_l is contiguously mapped (i.e., ℓ_l spans from $a_{l,k}$ to $a_{l,k} + \sigma_l$). The set of all labels in M_k is denoted as $\mathcal{L}(M_k)$. The labels read by τ_i are in the set $\mathcal{L}^{\mathrm{R}}(\tau_i)$, while those that are written are in the set $\mathcal{L}^{\mathrm{W}}(\tau_i)$. Labels may be read by multiple tasks but can be written by one task only. *Shared* labels store data dynamically produced by a task and consumed by another task, and encode functional dependencies between tasks. When a label is shared among tasks running on the same core, a double buffer mechanism [2] can be used to guarantee deterministic communications. Conversely, when labels are shared among tasks running on different cores, a different approach is required. In this paper we focus on the latter case.

Given a producer task τ_p and a consumer task τ_c, such that $\mathcal{P}(\tau_p) \neq \mathcal{P}(\tau_c)$, $\mathcal{L}^{\mathrm{S}}(\tau_p, \tau_c)$ denotes the set of *inter-core* shared labels, written by τ_p and read by τ_c. To avoid memory access interference across cores, we require that an arbitrary task τ_i in P_k accesses only labels mapped in M_k, and that all inter-core shared labels are mapped in M_{G} and not accessed directly by tasks. Copies of each shared label are maintained in the local memories to be accessed by the corresponding tasks. In detail, for any label $\ell_l \in \mathcal{L}^{\mathrm{S}}(\tau_p, \tau_c)$ accessed by multiple cores, two labels $\ell_{l,p} \in \mathcal{L}^{\mathrm{W}}(\tau_p)$ and $\ell_{l,c} \in \mathcal{L}^{\mathrm{R}}(\tau_c)$ are provided in $M(\tau_p)$ and $M(\tau_c)$, respectively, where $M(\tau_i)$ denotes the local memory accessed by τ_i.

The DMA is in charge of copying data between local copies (in local memories) and shared labels (in global memory) implementing inter-core *communication*. A communication $c_z(t)$ at time t is denoted as **(i)** $W(\tau_p, \ell_l)$, if it represents a write from $\ell_{l,p}$ to the shared label ℓ_l or **(ii)** $R(\ell_l, \tau_c)$ if it represents a read from ℓ_l to $\ell_{l,c}$. The DMA copies involve different timing parameters due to data transfers and programming overheads, which are presented in Section V.

IV. The LET Semantics

In the original LET definition (Giotto) [1], at the release instant of every periodic instance of τ_i, its inputs are updated (*LET read*). The task uses the input data to compute new output values, which are made available (*LET write*) only at the end of the period. We will refer hereafter to LET writes and reads as *LET communications*. LET communications in Giotto are logically performed in *zero-time*. The resulting behavior is deterministic in both time and value, and causality is always enforced. In a practical implementation, it may be convenient to perform at the start of each period both the LET writes that were logically supposed to happen at the end of the previous period and the reads for the current one. Such communication will still be in agreement with the original LET semantics [1] as long as all writes are performed before the reads that are casually related. By introducing a partial order "$\prec$" between communications the previous statement is formally defined in Property 1.

Property 1 (LET communications to and from a task)**.** *A periodic task $\tau_i \in \Gamma$ communicates according to LET if $W(\tau_i, \ell_a) \prec R(\ell_b, \tau_i)$ holds for any release time $t_{i,x} \in \mathcal{T}_i$ and for every label $\ell_b \in \mathcal{L}^S(\tau_p, \tau_i)$ read by τ_i and produced by a task τ_p, and every label $\ell_a \in \mathcal{L}^S(\tau_i, \tau_c)$ written by τ_i and consumed by τ_c. All LET communications must complete before executing the x-th job of τ_i.*

In addition, causal dependencies between tasks communication with LET must be satisfied. At any point in time, LET writes by a producer task τ_p to a shared label $\ell_a \in \mathcal{L}^{\mathrm{S}}(\tau_p, \tau_c)$ need to complete before starting the corresponding LET reads for the corresponding consumer task τ_c. This is formally defined in Property 2.

Property 2 (LET inter-task communications)**.** $W(\tau_p, \ell_a) \prec R(\ell_a, \tau_c)$ *must hold for each pair of tasks $\tau_p, \tau_c \in \Gamma$ such that $\mathcal{L}^S(\tau_p, \tau_c) \neq \emptyset$, if $\exists t \in \mathcal{T}_i \cap \mathcal{T}_j$, $\forall \ell_a \in \mathcal{L}^S(\tau_p, \tau_c)$.*

When dealing with real platforms, the time needed to move data across different memory slots and to execute tasks cannot be neglected. Therefore, the hypothesis of zero-time communication made in Giotto does not hold. All communications and executions must then be properly scheduled to retain causality as assessed by the LET semantics. To solve this issue, a strict order of execution for LET communications is proposed in [1], satisfying both Property 1 and 2. At any time instant $t \in \mathcal{T}$ when one or more LET communications are required, this order is enforced with the following sequence:

1) First, each task instance released at t performs all its LET writes.
2) Then, each task instance released at t performs all its LET reads.
3) Finally, all task instances released at t are set as ready to execute.

In addition, communications issued at different time instants must not overlap. This requirement is formally stated in Property 3.

Property 3. *For each pair $t_1, t_2 \in \mathcal{T}$ with $t_1 < t_2$, all LET communications required at time t_1 are completed before t_2.*

When implemented on a real platform, LET communications are usually managed by the CPU. For instance, the authors in [3] use a dedicated task running with the highest priority to perform such communications. In this way, any LET communication delays the execution of any task, and the delay introduced by LET communications must be small enough to keep the system schedulable.

The implementation proposed in Giotto satisfies causality and timing determinism. However, it has two fundamental issues. First, any task τ_i released at time t is required to wait for *all* LET write and read operations of *all* task instances that completed at t and start at t, even if such communications have no causal dependencies with τ_i. This may introduce unnecessary and possibly harming delays to latency-sensitive tasks, especially if heavy communication is required. Second, since the CPU is in charge of performing the copies, high priority tasks need to wait for communications related to lower priority tasks. The next section presents an alternative approach that solves such issues by leveraging the parallelism introduced by DMA.

V. A New Protocol for LET Communications with DMA

In this section, we propose a new protocol to perform LET communications. Our proposal allows for: **(i)** a limited interference on the task executions, thanks to the usage of a DMA to offload data transfers, and **(ii)** the possibility of finding a more flexible order of LET communications with respect to the Giotto proposal, guaranteeing an early release for latency-sensitive tasks. Indeed, when LET communications can be performed in parallel to task execution, it is possible to improve the responsiveness of latency-sensitive tasks.

In the proposed protocol, a single DMA engine is in charge of moving inter-core shared data from a source memory M_s to a destination memory M_d (with $M_s \neq M_d$). This choice allows avoiding contention in accessing global memory. Since, during execution, tasks access data from local memory only, such accesses are also free from contention as long as the local memory is dual-ported. For each core $P_k \in \mathcal{P}$, a LET task $\tau_{\mathrm{LET},k} \in \Gamma$ is in charge of dispatching LET communications by programming the DMA (a shared resource whose usage is regulated via inter-core synchronization as in [3]). For each data transfer, programming the DMA requires specifying: **(i)** the start address of the data to be copied in the source memory M_s, **(ii)** the

Algorithm 1 Constructing sets of LET communications

1: **function** COMPUTE_LETGROUP (t, τ_i)
2: $G^W(t,\tau_i) = \emptyset,\ G^R(t,\tau_i) = \emptyset$
3: **for** $\tau_j \in \Gamma$ **do**
4: **for** $v \in \mathbb{N}^{\geq 0}, v < H_i^*/T_i$ **do**
5: **if** $\eta_{i,j}(v)^W \cdot T_i == t$ **then**
6: **for** $\ell_l \in \mathcal{L}^{\mathrm{S}}(\tau_i,\tau_j)$ **do**
7: $G^W(t,\tau_i) = G^W(t,\tau_i) \cup W(\tau_i,\ell_l)$
8: **if** $\eta^R_{j,i}(v) \cdot T_i == t$ **then**
9: **for** $\ell_l \in \mathcal{L}^{\mathrm{S}}(\tau_j,\tau_i)$ **do**
10: $G^R(t,\tau_i) = G^R(t,\tau_i) \cup R(\ell_l,\tau_i)$
11: **return** $G^W(t,\tau_i),\ G^R(t,\tau_i)$

start address in the destination memory M_d, **(iii)** the size of the data transfer. By design of the DMA engine, each data transfer involves contiguous portions of memory, both in M_s and in M_d.

A. Grouping LET communications

Depending on the periods of the producer and consumer tasks, it is possible to safely skip *unnecessary* LET reads and writes [3]. For example, a producer task τ_p that is oversampled with respect to a consumer τ_c might skip some writes if that data is overwritten before it is consumed. Similarly, a consumer τ_c that is oversampled with respect to a producer τ_p may skip some of its reads if the data has not changed since its previous activation. Considering any pair of tasks $\tau_i, \tau_p \in \Gamma$ such that $\mathcal{L}^{\mathrm{S}}(\tau_p,\tau_i) \neq \emptyset$, the set of time instants where a LET write by τ_i is required is defined in [3] as $\{\eta^W_{p,i}(v)\cdot T_i \,|\, v \in \mathbb{N}^{\geq 0}\}$, with

$$\eta^W_{p,i}(v) = \begin{cases} \lfloor v \cdot T_i/T_p \rfloor & \text{if } T_p < T_i, \\ v & \text{otherwise.} \end{cases} \quad (1)$$

Similarly, for any consumer task $\tau_c \in \Gamma$ such that $\mathcal{L}^{\mathrm{S}}(\tau_i,\tau_c) \neq \emptyset$, the set of time instants when a LET read is needed is defined [3] as $\{\eta^R_{i,c}(v)\cdot T_i \,|\, v \in \mathbb{N}^{\geq 0}\}$, with

$$\eta^R_{i,c}(v) = \begin{cases} \lceil v \cdot T_i/T_c \rceil & \text{if } T_c > T_i, \\ v & \text{otherwise.} \end{cases} \quad (2)$$

The values defined by Eqs. (1) and (2) repeat every $\mathrm{LCM}(T_i,T_p)$ and $\mathrm{LCM}(T_i,T_c)$, respectively [3]. By considering all the tasks $\tau_j \in \Gamma \setminus \{\tau_i\}$ that have shared labels with τ_i, the LET writes and reads issued by task τ_i will then repeat periodically with period H_i^*:

$$H_i^* = \mathrm{LCM}\left(T_i, \left\{T_j \,|\, \mathcal{L}^{\mathrm{S}}(\tau_i,\tau_j) \neq \emptyset \ \vee\ \mathcal{L}^{\mathrm{S}}(\tau_j,\tau_i) \neq \emptyset\right\}\right). \quad (3)$$

Building upon this formulation, we extract the *necessary* LET communications of τ_i. Since H_i^* is an integer divisor of H, we only need to check the subset $\mathcal{T}_i^* \subseteq \mathcal{T}_i$ of the release instants of τ_i that require at least one LET communication in the interval $[0, H_i^*)$. The set of LET writes and reads required by τ_i at $t \in \mathcal{T}_i^*$ are defined as $G^W(t,\tau_i)$ and $G^R(t,\tau_i)$, respectively, and computed with Algorithm 1. The algorithm works as follows. Given $\tau_i \in \Gamma$ and $t \in \mathcal{T}_i^*$, for each task $\tau_j \in \Gamma$, it checks all the jobs with index v of τ_i in $[0, H_i^*)$ (line 4). Then, it checks whether t coincides with a release time in which a LET communication is needed (lines 5 and 8). If so, the corresponding LET writes and reads for each label shared between τ_i and τ_j are added to $G^W(t,\tau_i)$ and $G^R(t,\tau_i)$ (lines 7 and 10).

The set of all the LET writes and reads at time t that may involve tasks in Γ_k are $\mathcal{C}^W(t,M_k) = \bigcup_{\tau_i\in\Gamma_k} G^W(t,\tau_i)$ and $\mathcal{C}^R(t,M_k) = \bigcup_{\tau_i\in\Gamma_k} G^R(t,\tau_i)$, respectively. Finally, if $\mathcal{T}^* = \bigcup_{\tau_i\in\Gamma} \mathcal{T}_i^*$, the set of all the LET communications at time $t \in \mathcal{T}^*$ is defined as $\mathcal{C}(t) = \bigcup_{\tau_i\in\Gamma} G^R(t,\tau_i) \cup G^W(t,\tau_i)$. Since all tasks are synchronously released at time s_0, the set of communications at each time $t \in \mathcal{T}^*$ is a subset of the set at time s_0 [3], i.e., $\mathcal{C}(t) \subseteq \mathcal{C}(s_0),\ \forall t \in \mathcal{T}^*$.

In our proposal, we group such LET communications in *DMA transfers*. Each DMA transfer includes (a subset of) communications occurring at time t that share the same source and destination memory, and that will be performed all together. Since transferring data using the DMA requires copying a set of *contiguous* labels from the source M_s to the destination memory M_d, a contiguous mapping of the corresponding labels in M_s and M_d is required. In our model, one of those two memories is always M_{G}, while the other is local.

A DMA transfer is formally defined as a tuple $d_g(t) = \{\mathcal{C}_g(t), \mathcal{L}_g(t), M_s, M_d, a_{g,s}, a_{g,d}\}$, with $t \in \mathcal{T}^*$. Here, $\mathcal{C}_g(t)$ represents a set of *ordered* communications taken either from $\mathcal{C}^W(t,M_k)$ if $M_s = M_k$ and $M_d = M_{\mathrm{G}}$, or $\mathcal{C}^R(t,M_k)$ if $M_d = M_k$ and $M_s = M_{\mathrm{G}}$, while $\mathcal{L}_g(t)$ represents the corresponding set of labels involved in the data transfer, such that $\mathcal{L}_g(t)$ and their copies are *all contiguously allocated* both in M_s and M_d, and with the same order. The overall amount of data moved by the DMA transfer $d_g(t)$ is $\sum_{\ell_l\in\mathcal{L}_g(t)} \sigma_{\ell_l}$. Finally, $a_{g,s}$ and $a_{g,d}$ represent the start addresses at which labels in $\mathcal{L}_g(t)$ are contiguously allocated in the source memory M_s and the destination memory M_d, respectively.

The index g of $d_g(t)$ represents the order of execution of the DMA transfer. To preserve the LET semantics, the index values must be carefully assigned so that both Properties 1 and 2 are always satisfied $\forall t \in \mathcal{T}^*$. The set of all the DMA transfers at t due to tasks from all cores is denoted by $\mathcal{D}(t) = \bigcup_g d_g(t)$, such that $\bigcup_g \mathcal{C}_g(t) = \mathcal{C}(t)$ and $\bigcap_g \mathcal{C}_g(t) = \emptyset$. The whole set of DMA transfers is $\mathcal{D} = \bigcup_{t\geq 0} \mathcal{D}(t)$.

B. LET Communication Protocol

For each $t \in \mathcal{T}^*$, let $d_g(t) \in \mathcal{D}(t)$ be the data transfer with the highest priority (i.e., order of execution), and $\tau_{\mathrm{LET},k}$ be the LET task associated with the corresponding memory M_k, which is the source or the destination local memory in $d_g(t)$. The proposed protocol behaves according to the following rules.

- **R1** A task τ_i released at time t, is *ready* for execution when all the LET communications in $G^W(t,\tau_i)$ and $G^R(t,\tau_i)$ are completed.
- **R2** $\tau_{\mathrm{LET},k}$ programs the DMA for $d_g(t)$ and suspends. Upon completion, an interrupt is raised to notify the termination of the data transfer and to awaken one of the LET tasks to handle the next transfer $d_{g+1}(t)$ (possibly running in a core $P_p \neq P_k$).
- **R3** When a DMA completion interrupt arrives, all the tasks for which the data dependencies are satisfied by the completed g-th DMA transfer are marked as ready.

We assume that *at worst* o_{DP} time units are required to program a regular DMA transfer. The interrupt service routine (ISR) notifying the DMA completion requires up to o_{ISR} time units.

Fig. 1 shows an example of a schedule for LET communications using a DMA with the protocol proposed in this work and with the original order of the LET communications as specified in Giotto [1]. Inset (a) reports the memory layouts and the parameters characterizing a DMA data transfer for a platform with two cores. Inset (b) illustrates the communication schedule when using the proposed protocol with an optimized re-ordering of the communications, while inset (c) considers the Giotto approach [1], where tasks become ready after *all* writes and reads are performed. Note that the schedule of Fig. 1(b) provides a considerably smaller data acquisition latency for task τ_2, which may be essential to guarantee its schedulability.

The proposed protocol requires computing offline the set $\mathcal{D}$ of all the DMA transfers, presented in Section V-A, and the knowledge of the addresses in the global and local memories of each label. Moving multiple labels with a single transfer reduces the overhead, as it requires less processor intervention. This is beneficial for satisfying data acquisition deadlines, while it complicates the label allocation problem. Furthermore, a single DMA transfer may involve data related to different tasks: this may cause additional delays to latency-sensitive tasks, since tasks become ready as an effect of the DMA

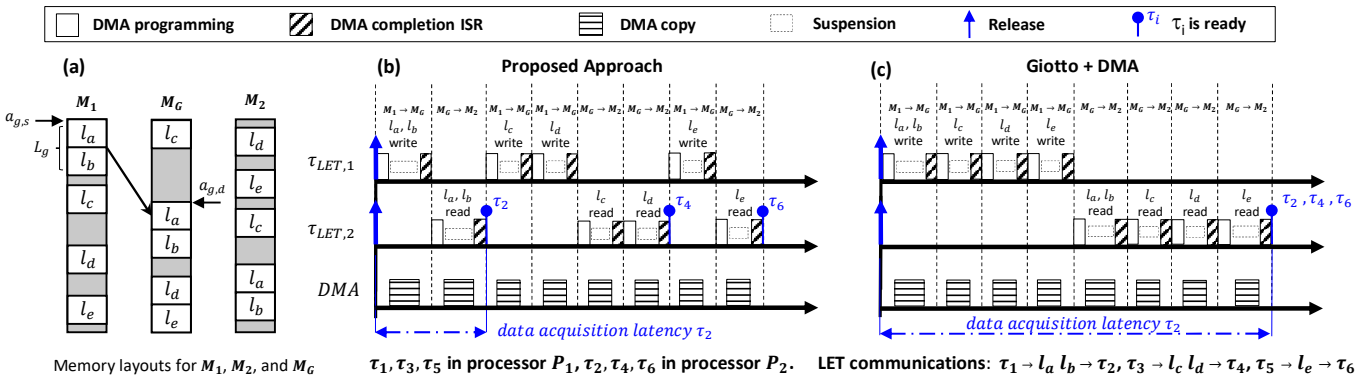

Figure 1. Scheduling of LET communications using a DMA with the proposed approach (inset (b)) and with the original Giotto approach [1] (inset (c)).

completion interrupt (rule R3). Hence, the definition of a feasible set of data transfers requires the satisfaction of multiple constraints, which are managed by the optimization problem in Section VI.

C. Schedulability Analysis

Although it is not the main focus of this paper, we briefly discuss how to leverage state-of-the-art results for analyzing the system schedulability. For each core $P_k \in \mathcal{P}$, this involves the schedulability of: **(i)** the LET task $\tau_{\text{LET},k}$ and, **(ii)** all the other tasks running on P_k. Since $\tau_{\text{LET},k}$ runs at the highest priority, it can be delayed only by LET tasks of different cores contending for the DMA, and by the ISR associated with the DMA completion interrupt. Its schedulability is ensured by the optimization problem of Section VI and is required for Property 3 to hold. Other non-LET tasks running on P_k can be analyzed with state-of-the-art response-time analysis techniques for periodic tasks with a release jitter given by the data acquisition latency. Additionally, the LET task behaves as a generalized multiframe task [3], where each job exhibits a *segmented* self-suspending behavior [14] (rules R2 and R3). When computing the high-priority interference, it is possible to model each execution segment of $\tau_{\text{LET},k}$ as an independent sporadic task [14].

VI. Optimization Problem

This section presents an MILP formulation to derive an optimal memory allocation and an optimal schedule of DMA transfers under the proposed LET protocol. The problem input is a task set statically mapped on a multicore platform. Labels shared by tasks on different cores are mapped in global memory, while the copies are mapped in the local memories of the communicating tasks. We are interested in finding the optimal mapping for the synchronous release instant s_0, when communications involve all labels and tasks, while ensuring also that the mapping is feasible for each other set $\mathcal{C}(t)$, $\forall t \in \mathcal{T}^*$, i.e., that all communications mapped in the same DMA data transfer have all the corresponding labels contiguously mapped. The goals of the MILP formulation are the following. **(I)** Determine which LET communications are scheduled within each DMA transfer, and enforce a contiguous allocation for labels involved in the same transfer (Constraints 1-6). **(II)** Determine the order of the data transfers in accordance with LET (Constraints 7, 8 and 10). **(III)** Ensure that the data acquisition deadlines are not exceeded (Constraint 9).

A. MILP Variables

The main variables of the formulation are ($\mathbb{B}$ is the Boolean set):

- *Adjacency of labels:* $\text{AD}_{k,a,b} \in \mathbb{B}$ is set to 1 if the address of label ℓ_b is immediately below ℓ_a in M_k; otherwise it is set to 0.
- *Communication in DMA transfer:* $\text{CG}_{z,g} \in \mathbb{B}$ is set to 1 if the communication $c_z(s_0)$ (being either $R(\ell_l, \tau_i)$ or $W(\tau_i, \ell_l)$) is mapped in the g-th DMA transfer; otherwise it is set to 0.
- *Last LET Read of a Task:* $\text{RG}_{i,g} \in \mathbb{B}$ is set to 1 if the last LET read of τ_i occurring at s_0 is in the g-th DMA transfer; otherwise it is set to 0.

The following are auxiliary variables.

- *Position of label:* $\text{PL}_{k,a} \in \mathbb{R}$ is equal to the relative position of the label ℓ_a mapped in M_k. $\text{PL}_{k,a}$ is defined such that $\forall M_k \in \mathcal{M}$: $\sum_{\ell_a \in \mathcal{L}(M_k)} \text{PL}_{k,a} = \sum_{i=1}^{|\mathcal{L}(M_k)|} i$.
- *ID of Communication:* $\text{CGI}_z \in \mathbb{R}$ is equal to the index g of the DMA transfer $d_g(s_0)$ where the communication c_z is mapped, i.e., $\forall c_z(s_0) \in \mathcal{C}(s_0)$, $\text{CGI}_z = \sum_g g \cdot \text{CG}_{z,g}$.
- *ID of last LET read:* $\text{RGI}_i \in \mathbb{R}$ is equal to the index g of the DMA transfer $d_g(s_0)$ where the last LET read of τ_i at s_0 is mapped, i.e., $\forall \tau_i \in \Gamma$, $\text{RGI}_i = \sum_g g \cdot \text{RG}_{i,g}$.

The latter variables represent integer values, but are relaxed as reals to improve the performance of the optimization engine. In reality, the variables CGI_z and RGI_i can only take integer values from the equality constraints on them. Similarly the values of $\text{PL}_{k,a}$ are constrained to integers only by Constraint 5.

B. Constraints on Mapping Labels and Communications

Each communication $c_z(t) \in \mathcal{C}(t)$ must be mapped to exactly one DMA data transfer. Since at $\mathcal{C}(s_0) \supseteq \mathcal{C}(t)$, $\forall t \in \mathcal{T}^*$, it is sufficient to perform the following check (Constraint 1).

Constraint 1. $\forall c_z(s_0) \in \mathcal{C}(s_0),\ \sum_g CG_{z,g} = 1$

Similarly, the variable representing the last read of task τ_i must be mapped to exactly one DMA data transfer (Constraint 2).

Constraint 2. $\forall \tau_i \in \Gamma,\ \sum_g RG_{i,g} = 1$

Constraint 3 crucially enforces the definition of RGI_i, stating that the DMA data transfer index of the last LET read of a task at s_0 is computed as the maximum data transfer index of all its reads.

Constraint 3. $\forall \tau_i \in \Gamma,\ RGI_i = \max_{c_z(s_0) \in G^R(s_0,\tau_i)} CGI_z$

Constraint 4 states that, for each memory $M_k \in \mathcal{M}$, each label is allocated immediately after only one label, and immediately before another label. We provide dummy labels at the beginning and at the end of the memory space to ensure the consistency of the constraint.

Constraint 4. $\forall M_k \in \mathcal{M}, \forall \ell_a \in \mathcal{L}(M_k)$:
$\sum_{\ell_b \in \mathcal{L}(M_k) \setminus \ell_a} AD_{k,a,b} = 1$ *and* $\sum_{\ell_b \in \mathcal{L}(M_k) \setminus \ell_a} AD_{k,b,a} = 1$

To obtain an unique position address for each label, the following constraint is also added.

Constraint 5. $\forall M_k \in \mathcal{M}, \forall \ell_a, \ell_b \in \mathcal{L}(M_k), \ell_a \neq \ell_b$:
$PL_{k,a} + 1 - (1 - AD_{k,a,b}) \cdot M \leq PL_{k,b} \leq PL_{k,a} + 1 + (1 - AD_{k,a,b}) \cdot M$,
where M is a large positive constant value that represents infinity.

Constraint 5 uses a *big-M* formulation: if ℓ_b is mapped immediately after ℓ_a in M_k ($\text{AD}_{k,a,b} = 1$), then the position index of ℓ_b is equal to the one of ℓ_a plus 1. If $\text{AD}_{k,a,b} = 0$ the constraint has no effect.

Next, for each DMA transfer, all the labels involved must be contiguous in the same order in both the source and destination memories. This is equivalent to check that, if any two communications $c_i(t)$ and $c_j(t)$ are in the same DMA data transfer $d_g(t)$, then it exists at least one label ℓ_c copied during $d_g(t)$ that is adjacent to at least one of the two labels ℓ_a and ℓ_b involved in $c_i(t)$ and $c_j(t)$, respectively, in both source and destination memories, where it can even be $\ell_c = \ell_a$ or $\ell_c = \ell_b$. This is formally enforced by Constraint 6.

Constraint 6. *For each pair $c_i(t), c_j(t) \in \mathcal{C}^W(t, M_x)$ (respectively, $c_i(t), c_j(t) \in \mathcal{C}^R(t, M_x)$), with $i \neq j$ and $t \in \mathcal{T}^*$, such that $c_i(t)$ involves ℓ_a and $c_j(t)$ involves ℓ_b, $\forall g \in \{1, 2, \ldots\}$ it holds that*

$$(CG_{i,g} \wedge CG_{j,g}) \leq \sum_{c_z(t) \in Z(t)} \left(LG^z_{a,c,g} + LG^z_{b,c,g}\right)$$

where ℓ_c is the label involved in the $z-th$ communication $c_z(t)$, with $Z(t) = \mathcal{C}^W(t, M_x)$ (respectively, $Z(t) = \mathcal{C}^R(t, M_x)$), and $LG^z_{\star,c,g} = (AD_{G,\star,c} \wedge AD_{x,\star,c} \wedge CG_{z,g})$.

Proof. If $c_i(t)$ and $c_j(t)$, which move data from local memory M_x to global memory M_G or viceversa, are in the same DMA transfer of index g, then the LHS of the inequality assumes value 1. From the definition of DMA transfers, this requires that either **(i)** labels ℓ_a and ℓ_b are mapped in adjacent memory slots, or **(ii)** at least one between ℓ_a and ℓ_b is adjacent to another label ℓ_c that is part the g-th DMA transfer. In both cases, the constraint enforces that there must exist a communication $c_z(t)$ moving data in the very same direction of $c_i(t)$ and $c_j(t)$, which involves a label ℓ_c. In this case, for the constraint to hold, the RHS must also be set to at least 1. Note that $\text{LG}^z_{a,c,g} = 1$ if and only if label ℓ_c is mapped below ℓ_a in both M_x and M_G ($\text{AD}_{G,a,c} = \text{AD}_{x,a,c} = 1$) and $c_z(t)$ is in the DMA transfer of index g ($\text{CG}_{z,g} = 1$). This is consistent with the above points (i), when $\ell_c = \ell_b$, and (ii). A dual reasoning can be made if $\text{LG}^z_{b,c,g} = 1$.

Conversely, if at least one between $c_i(t)$ and $c_j(t)$ is not in the DMA transfer of index g, then the LHS of the inequality equals 0 and the constraint has no effect. □

C. Constraints on LET Properties and Data Acquisition Deadlines

Property 1 (all LET writes of a task must complete before starting its LET reads) is enforced through Constraint 7, by imposing an order between the indexes of the corresponding DMA data transfers.

Constraint 7 (Property 1)**.** *$\forall \tau_i \in \Gamma$, for each pair $c_w(s_0) \in G^W(s_0, \tau_i)$ and $c_r(s_0) \in G^R(s_0, \tau_i)$: $CGI_w < CGI_r$.*

Again, checking the LET properties for all communications at s_0 is sufficient to provide LET guarantees at all other communication instants. Constraint 8 enforces Property 2, i.e., for each shared label, the corresponding LET write must be completed before the LET read.

Constraint 8 (Property 2)**.** *For each pair $c_w(s_0), c_r(s_0) \in \mathcal{C}(s_0)$ with $c_w(s_0) = W(\tau_i, \ell_l)$ and $c_r(s_0) = R(\ell_l, \tau_j)$: $CGI_w < CGI_r$.*

The communication latency λ_i is computed by accumulating the delays of all the communications occurring until the last LET read of τ_i. The temporal constraints on data acquisition deadlines at s_0 are then guaranteed by Constraint 9.

Constraint 9. *$\forall \tau_i \in \Gamma$, $\lambda_i \leq \gamma_i, \forall \overline{g} \in \{1, 2, \ldots\}$ where*

$$\lambda_i \geq (RGI_i{+}1)\lambda_O + \omega_c \left(\sum_{g=1}^{\overline{g}} \sum_{c_z(s_0) \in \mathcal{C}(s_0)} \sigma_l CG_{z,g} \right) - (1 - RG_{i,\overline{g}})M,$$

where ℓ_l is the label involved in the communication $c_z(s_0)$.

Proof. Assume that the number of DMA transfers, occurring at s_0 and performed before activating τ_i, is $\overline{g}$. If this is not the case, the last big-M term in the RHS makes the constraint inactive. When the constraint is active, the number of DMA transfers at s_0 until the release of τ_i is given by $\text{RGI}_i + 1$. Each DMA transfer generates a (worst-case) overhead given by the programming and interrupt costs, i.e., $\lambda_O = o_{\text{DP}} + o_{\text{ISR}}$: this corresponds to the first term in the RHS. The second term is the sum of sizes of the labels involved in the first $\overline{g}$ DMA transfers, multiplied by the cost of each copy ω_c. □

Finally, Property 3 is enforced by Constraint 10, which states that all the DMA data transfers for each $t_1 \in \mathcal{T}^*$ must end before the next activation instant $t_2 \in \mathcal{T}^*$, with $t_2 > t_1$.

Constraint 10 (Property 3)**.** *$\forall t_1, t_2 \in \mathcal{T}^*$, such that $t_1 < t_2$,*
$(\max_{\tau_i \in \Gamma}(RGIT_{t_1,i}) + 1)\lambda_O + \omega_c \sum_{c_z(t_1) \in \mathcal{C}(t_1)} \sigma_l \leq t_2 - t_1$, where ℓ_l is the label involved in the z-th communication at t_1.

This constraint follows similarly to the previous one. Here, $\text{RGIT}_{t,i}$ is an auxiliary variable that denotes the index of the last LET read of task τ_i among the communications occurring at time t.

In this formulation, checking the communication delays at s_0 as presented in Constraint 9 is a sufficient condition to guarantee schedulability at all other instants $t \in \mathcal{T}^*$, $t > s_0$. This is formally stated in the following theorem.

Theorem 1. *A mapping that satisfies Constraints 1-10 guarantees that, for each $t \in \mathcal{T}^*$, the data communication delay experienced by τ_i cannot be larger than the one experienced by τ_i at s_0.*

Proof. By contradiction, let assume that the theorem does not hold for a given $t \in \mathcal{T}^*$. Then, at t, either **(i)** more LET communications than at s_0 are required or **(ii)** more DMA transfers are issued due to communications involving labels that are more fragmented (because communications involving intermediate labels are skipped). Condition (i) cannot occur since $\mathcal{C}(s_0) \supseteq \mathcal{C}(t)$, $\forall t \in \mathcal{T}^*$. On the other hand, condition (ii) is also impossible since Constraint 6 guarantees that for all communications $\forall t \in \mathcal{T}^*$, the labels mapped in the same DMA transfer are contiguous. Hence the theorem follows. □

Objective Function: While the problem is in essence a feasibility problem, two objective functions are proposed to encode the intuitive goal of minimizing the communications overheads: minimizing the number of DMA data transfers, or the maximum ratio between the communication delay and the period of a task, i.e.:

$$\textbf{minimize} \max_{\tau_i \in \Gamma} \left(\text{RGI}_i\right) \quad (4) \qquad \textbf{minimize} \max_{\tau_i \in \Gamma} \left(\lambda_i / T_i\right). \quad (5)$$

VII. Experimental Results

The proposed approach has been evaluated with a realistic case study representative of an autonomous driving application presented by Bosch for the WATERS 2019 Industrial Challenge [15]. The parameters of tasks and labels and the data dependencies are provided with the case study, while the task mapping is based on the challenge solution of [16]. We performed experiments considering a DMA programming time $o_{\text{DP}} = 3.36\mu s$, using the results of measurements from [8], and the delay due to each DMA completion interrupt equal to $o_{\text{ISR}} = 10\mu s$. Since the data acquisition deadlines of the tasks were not provided by the Challenge, they have been set according to the following sensitivity analysis procedure. First, we computed the worst-case response time (WCRT) R_i of each task $\tau_i \in \Gamma$ (as discussed in Section V), and the slack $S_i = D_i - R_i$. Then, we set $\gamma_i = \alpha \cdot S_i$, and we checked schedulability by computing the WCRTs using γ_i as a bound on the jitter, with $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$.

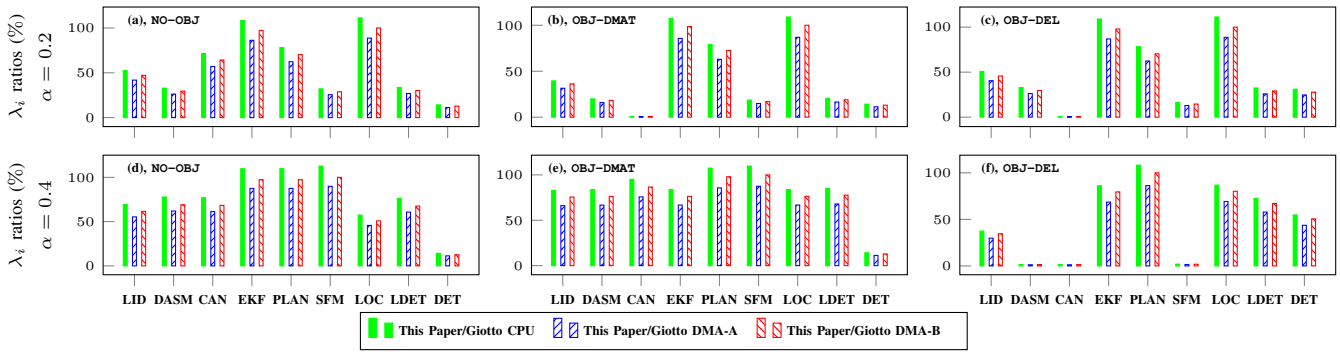

Figure 2. Ratios between data acquisition delays λ_i obtained under configurations for the tasks of the WATERS 2019 Challenge [15].

Table I
OBSERVED RUNNING TIMES AND NUMBER OF DMA TRANSFERS

Obj. Function	MILP running time		# DMA Transfers	
	$\alpha = 0.2$	$\alpha = 0.4$	$\alpha = 0.2$	$\alpha = 0.4$
NO−OBJ	8 sec	8 sec	16	16
OBJ−DMAT	1 hour	1 hour	12	12
OBJ−DEL	8 sec	12 sec	16	16

We compared four different approaches: **(i)** the one proposed in this paper, **(ii)** the state-of-the-art Giotto approach [1], with LET copies performed by the CPU (labeled Giotto−CPU), **(iii)** the Giotto approach enhanced with the usage of a DMA but without the communication re-ordering proposed in this paper and a separate DMA transfer (i.e., no knowledge of the memory layouts) for each LET copy (Giotto−DMA−A), and **(iv)** the Giotto approach with DMA and using the memory layout found by the optimization problem in Section VI for the case at point (i) (Giotto−DMA−B). We also considered three different cases for the objective function: (a) no objective function (NO−OBJ), (b) the minimization of the number of DMA transfers (Eq. (4), OBJ−DMAT), and (c) the minimization of the λ_i/T_i ratio (Eq. (5), OBJ−DEL). The experiments have been performed on a machine with 128GB of memory, 2x Intel Xeon(R) CPU E5-2640 v4 @ 2.40GHz, with 40 cores. The MILP has been solved with IBM CPLEX, setting a timeout of 1 hour.

Six representative configurations are reported in Fig. 2. On the X-axis there are the 9 tasks of the case study (LID, DASM, CAN, EKF, PLAN, SFM, LOC, LDET and DET) and on the Y-axis the ratio between the data acquisition latency λ_i obtained with our method and with the other three alternative approaches.

The three cases on top correspond to $\alpha = 0.2$, the bottom row to $\alpha = 0.4$. Fig. 2 (a) and (d) show the result obtained without an objective function. A solution is obtained very quickly ($8s$: see Table I) and already shows significant improvements with respect to the latency of DASM, CAN and SFM tasks. Fig. 2 (b) and (e) show the results obtained when minimizing the number of DMA transfers. The solution for $\alpha = 0.2$ has values for λ_i even lower than in the previous case, but this is not true for $\alpha = 0.4$. In this case, the solver is free to choose any solution with a minimal number of DMA transfers as long the constraints on γ_i are respected, which are less restrictive than those set for $\alpha = 0.2$. The number of DMA transfers found is 12 in both cases. This may not be the minimal number, as the solver stopped with a feasible solution after the timeout (Table I).

Finally, Fig. 2 (c) and (f) show the results when the ratio λ_i/T_i is minimized. The charts for $\alpha = 0.2$ and $\alpha = 0.4$ do not differ significantly and show very short delays for tasks with a small period (e.g., DASM, CAN, and SFM) and significant improvements also for the other tasks. Similar results have been obtained for $\alpha = 0.3$ and $\alpha = 0.5$, while for $\alpha = 0.1$ the solver was not able to find a feasible solution. Overall, the proposed approach offers much shorter latencies λ_i by optimizing the individual communication delay of each task τ_i, reaching improvements up to 98% on prior approaches.

VIII. CONCLUSIONS

This paper presented a new protocol to perform LET communications while leveraging the parallelism offered by a DMA engine. The problem of finding an optimal memory allocation and scheduling of the LET communications due to different tasks has been addressed, formulating it as a MILP. Experimental results reported improvements up to 98% with respect to the Giotto approach with CPU-driven data transfers on a realistic use-case.

REFERENCES

[1] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *International Workshop on Embedded Software*. Springer, 2001, pp. 166–184.

[2] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication Centric Design in Complex Automotive Embedded Systems," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, 2017.

[3] A. Biondi and M. Di Natale, "Achieving Predictable Multicore Execution of Automotive Applications Using the LET Paradigm," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018.

[4] Infineon, "AURIX™ 32-bit microcontrollers for automotive and industrial applications Highly integrated and performance optimized."

[5] P. Pazzaglia, A. Biondi, and M. Di Natale, "Optimizing the functional deployment on multicore platforms with logical execution time," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 207–219.

[6] S. Wasly and R. Pellizzoni, "Hiding memory latency using fixed priority scheduling," in *19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[7] D. Casini, P. Pazzaglia, A. Biondi, M. Di Natale, and G. Buttazzo, "Predictable memory-cpu co-scheduling with support for latency-sensitive tasks," in *57th Design Automation Conference (DAC)*, 2020.

[8] R. Tabish, R. Mancuso, S. Wasly, R. Pellizzoni, and M. Caccamo, "A real-time scratchpad-centric os with predictable inter/intra-core communication for multi-core embedded systems," *Real-Time Systems*, 2019.

[9] B. Rouxel, S. Skalistis, S. Derrien, and I. Puaut, "Hiding communication delays in contention-free execution for spm-based multi-core architectures," in *31st Euromicro Conference on Real-Time Systems*, 2019.

[10] A. Marchand, P. Balbastre, I. Ripoll, M. Masmano, and A. Crespo, "Memory resource management for real-time systems," in *19th Euromicro Conference on Real-Time Systems*, July 2007, pp. 201–210.

[11] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Memory feasibility analysis of parallel tasks running on scratchpad-based architectures," in *39th Real-Time Systems Symposium (RTSS)*, 2018.

[12] I. Puaut and C. Pais, "Scratchpad memories vs locked caches in hard real-time systems: a quantitative comparison," in *2007 Design, Automation Test in Europe Conference Exhibition*, 2007, pp. 1–6.

[13] J. Whitham and N. Audsley, "Implementing time-predictable load and store operations," in *Proceedings of the Seventh ACM International Conference on Embedded Software*, 2009, pp. 265–274.

[14] J.-J. Chen *et al.*, "Many suspensions, many problems: a review of self-suspending tasks in real-time systems," *Real-Time Systems*, Sep 2018.

[15] A. Hamann, D. Dasari, F. Wurst, I. Sañudo, N. Capodieci, P. Burgio, and M. Bertogna. WATERS Industrial Challenge 2019.

[16] D. Casini, P. Pazzaglia, A. Biondi, G. Buttazzo, and M. Di Natale, "Addressing analysis and partitioning issues for the Waters 2019 challenge," in *10th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2019)*, 2019.