Optimizing Inter-Core Communications under the LET Paradigm using DMA Engines

Paolo Pazzaglia *Member, IEEE*, Daniel Casini *Member, IEEE*, Alessandro Biondi *Member, IEEE* and Marco Di Natale *Senior Member, IEEE*

Abstract—Modern automotive applications are increasingly characterized by the need to transfer massive amounts of data in a predictable and deterministic way, possibly leveraging the Logical Execution Time (LET) paradigm. However, current proposals for LET communications are limited to core-commanded data transfers, which may result in large delays for data-intensive systems. To address this issue, we explore the use of Direct Memory Access (DMA) to handle LET communication with improved parallelism. Each DMA transfer operates on a contiguous memory area, thus calling for an optimized memory mapping to maximize performance. Modern DMA engines offer also advanced configurations, such as linked-lists of data transfers, which may provide more flexibility at the expenses of an increased (initial) programming overhead. Leveraging all such features of DMA engines, we propose a set of designs and protocols for LET communications with trade-offs between latency and space requirements. For each option we present the formulation to compute the optimal scheduling and memory allocation solution as a mixed-integer linear programming problem. Experimental results show the feasibility of the approach and a comparison of the solutions obtained using the proposed methods, showing a considerable improvement in terms of data acquisition latency when compared to LET communication without DMA.

Index Terms—Logical Execution Time, Direct Memory Access Engines, Multicore Automotive Systems, Real-Time Systems.

1 Introduction

Modern multicore automotive applications are characterized by the need for predictable and deterministic communication among computational activities (*tasks*), possibly allocated on different cores. To this end, the Logical Execution Time (LET) paradigm [1] has been increasingly adopted by the automotive industry [2]. The LET paradigm allows for time-deterministic communications among tasks on multicore platforms by restricting the copy of data at specific time instants, such as the beginning and the end of a task period. LET also emerged as an opportunity to precisely schedule in time memory accesses and avoid contention among tasks concurrently accessing shared memories (e.g., a DRAM) from different cores [3], [4].

In literature, implementations of inter-core LET communication that also mitigate inter-core memory contention leverage tasks running at the highest priority in each core, copying data between a private core-local memory (e.g., a scratch-pad) and the global memory, used as intermediate storage [5], [6]. In this paper, we refer to such communications as *local-global-local*. In prior work, these copies are core-commanded and may not be suitable when large amounts of data are transferred across cores due to their potentially large delays. This can be a relevant issue in many emerging automotive applications, such as autonomous driving systems, which involve data-intensive communications (e.g., camera images, lidar data, etc.). In these cases, Direct Memory Access (DMA) engines offer an interesting opportunity to free cores from the duty of copying data, fostering parallelism and reducing waiting times for tasks.

In practice, a careful design is required to conjugate the parallelism introduced by the DMA with the semantics of the LET paradigm. First of all, the causality dependencies of all DMA-based communications must always be guaranteed. Next, DMA engines require data to be allocated in contiguous memory areas to be moved in a single data transfer (i.e., without processor intervention), thus calling for an optimized memory allocation. On the other hand, many DMA engines found in commercial platforms [7], [8], [9] include more advanced features, such as the possibility of programming multiple consecutive DMA data transfers with a single processor intervention (*"linked-list" mode* [7]). In this case, data can be transferred between non-contiguous memory areas but requiring additional programming overheads and interrupt management. When using the linked-list mode, the DMA can raise the completion interrupt when each transfer is completed, or when the whole list is completed, giving rise to non-trivial trade-offs. Furthermore, by leveraging clever buffering mechanisms, the DMA engine can also be used to perform direct transfers between private core-local memories, referred to as *local-to-local* communications, without involving the (slower) global memory.

This paper. This paper proposes a set of new DMA-based protocols to handle inter-core LET communication in multicores, thus allowing to retain the benefits of LET while freeing the cores from the duty of copying data. We tackle both classic and linked-list-based DMA transfers, as well as supporting both local-global-local copies and direct copies between local memories. The paper presents a mixed-integer linear programming formulation to derive

- *P. Pazzaglia is with the Saarland University, Saarbrücken, Germany.*
- *D. Casini, A. Biondi, M. Di Natale are with the Real-Time Systems Laboratory (ReTiS Lab) and the Department of Excellence in Robotics and Artificial Intelligence, Scuola Superiore Sant'Anna.*
  E-mail: pazzaglia@cs.uni-saarland.de, {daniel.casini, alessandro.biondi, marco.dinatale}@santannapisa.it

Manuscript received XXX.

an optimal memory allocation scheme and schedule of the communications in different configurations. The goal of the optimization problem is to address all the causality constraints mandated by LET while minimizing the data acquisition latency of each task. An evaluation is performed to compare different communications strategies, showing that the usage of DMA engines can reduce the data acquisition latency compared to the original approach of [1].

This paper extends a conference publication [10] of the same authors by: **(i)** supporting two variants of DMA transfers programmed through a linked-list, on top of the classical configuration where each transfer between contiguous memory areas requires a processor intervention to program the DMA; **(ii)** supporting also local-to-local DMA transfers, while [10] targeted only local-global-local communications, thus aiming to a reduced number of transfers; **(iii)** providing a more general modelization of the system, as well as supporting the new contributions in the optimization problem, which is improved and enriched with additional constraints and comments not included in [10]; and **(iv)** reporting extended experimental results to explore the advantages provided by the new contributions of this work.

2 Related Work

The Logical Execution Time paradigm was first proposed in the seminal paper of Henzinger et al. [1], as part of the Giotto time-triggered language. While the original proposal dated 2001, LET only gained renewed attention in recent years [11], when the automotive industry faced the problem of restoring the causal order of execution when moving legacy applications from single to multicore platforms (see e.g. Hamann et al. [2]). Later, Biondi and Di Natale [5] proposed a scheme for practically implementing the LET paradigm in a multicore platform, focusing on the Aurix Tricore TC275 [12]. Igarashi et al. [13] proposed heuristic methods to schedule tasks communicating with LET to avoid communication contention, later extending the approach to clustered many-core platforms [14]. Pazzaglia et al. [6] presented a method for functional partitioning on a multicore platform of real-time applications communicating according to the LET paradigm. Gemlau et al. [15] introduced the concept of System-level LET, which aims at making LET suitable also for tasks communicating in distributed systems, where communication delays are considerably longer than those experienced between cores.

Several works adopted the LET paradigm for the case of chains of communicating tasks. For example, Martinez et al. [16] proposed an end-to-end analysis of tasks communicating according to LET, also proposing methods to select offsets to increase the predictability. The same authors proposed in [17] an extended analysis for communicating tasks using different communication models, such as explicit, implicit and LET. Becker et al. [18], [19] analyze the end-to-end delay of effect chains with LET, while Gunzel et al. [20] provided a timing analysis of asynchronized distributed chains. Kordon and Tang [21] proposed methods to bound the age latency of a real-time task set under LET.

In an orthogonal research direction, several authors addressed the problem of optimizing the loading of data in core-local memories with DMAs. Saidi et al. proposed methods to find an optimal clustering of transfers for applications that require moving data from an off-chip slow memory to local memories using buffering mechanisms [22], [23]. Other notable works proposed protocols [24], [25], [26] to preload core-local memories when adopting the PRedictable Execution Model (PREM) [27] characterized by disjoint read, execute, and write phases under different settings. Wasly and Pellizzoni derived protocols for static [28] and dynamic [24] scheduling in systems using PREM. Tabish et al. [29], [25] considered the case where the DMA is accessed via time-division multiplexing, whereas Casini et al. [26] proposed a co-scheduling mechanism optimized to favor latency-sensitive tasks. Rouxel et al. [30] presented an approach for reducing communication delays using a DMA and a static scheduling algorithm. Other papers addressed the problem of limiting the memory-space requirement and proposed memory allocation algorithms [31], [32]. Whitham et al. [33] and Puaut et al. [34] presented algorithms to obtain a predictable allocation in scratchpad memories.

Overall, to the best of our knowledge, no prior work jointly considered the usage of DMA engines to perform inter-core communications using LET, with the exception of the previous conference version of this paper [10].

3 System Model

3.1 Platform Model

The real-time embedded platform considered in this work consists of a set $\mathcal{P} = \{P_1, \ldots, P_N\}$ of N cores. Each core $P_k \in \mathcal{P}$ has a private local memory (e.g., a scratchpad or a cache using lockdown), which is dual-ported [24], [25], i.e., two different areas of the local memory can be simultaneously accessed without contention. Shared cache memories are either not present or turned off. 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 ones are local memories (with M_k being the memory assigned to P_k), and M_G is global. A DMA engine is included in the platform, performing data transfers between any pair of memories, and operating in parallel with respect to the core executions. This setting is representative of commercial platforms used in automotive systems, e.g., the AURIX TC2xx and AURIX TC3xx by Infineon [12], or other high-end platforms when using cache lockdown.

3.2 Application Model

The application consists of a set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ of periodic real-time tasks, under partitioned scheduling, i.e., each task τ_i is statically assigned to a processor. We introduce $\mathcal{P}(\tau_i)$ and $M(\tau_i)$ as functions that retrieve the core where τ_i executes and the associated memory, respectively. The subset of tasks assigned to processor P_k is denoted by Γ_k. Each task τ_i is synchronously released at the system startup $s_0 = 0$ and releases a potentially infinite sequence of instances called *jobs*, separated by a period T_i. Each instance is required to complete within D_i time units from its release, where $D_i = T_i$ is the relative (implicit) deadline. A task is deemed *schedulable* when each of its jobs completes before its deadline. Hereafter, we work under the hypothesis that Γ is always schedulable. Introducing H as the hyperperiod of the

task set, the set of release instants of τ_i in the interval $[0, H)$ is denoted with the symbol $\mathcal{T}_i = \{t_{i,0}, t_{i,1}, \ldots, t_{i,N_i-1}\}$, 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 $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$.

An arbitrary pair of tasks τ_p and τ_c is characterized by a *producer-consumer* relationship when the consumer task τ_c reads a *variable* d_x that is written by the producer task τ_p. We refer to this relationship as a *functional dependency* $f_q = (\tau_p, \tau_c, d_x)$. In this context, a variable is an abstract entity that contains data. The set $\mathcal{F}(\tau_p, \tau_c)$ denotes the set of functional dependencies between τ_p (producer) and τ_c (consumer). For ease of notation, given a functional dependency f_q and a variable d_x, we introduce the operator $\in$ to denote whether d_x is part of f_q: in this case, $d_x \in f_q$ holds.

Tasks access variables by reading and writing data stored in memory locations called *labels*. In the following we restrict our analysis to those labels that are assigned to variables in functional dependencies. Each label ℓ_l is characterized by a tuple $\ell_l = \{\sigma_l, M_k, a_{k,l}, d_x\}$, where **(i)** σ_l represents the size (in bytes), **(ii)** $M_k \in \mathcal{M}$ represents the memory which ℓ_l is assigned to, **(iii)** $a_{k,l}$ represents the address in M_k at which label ℓ_l is contiguously mapped (i.e., ℓ_l spans from $a_{k,l}$ to $a_{k,l} + \sigma_l$), and **(iv)** d_x represents the variable mapped in that label. To avoid memory interference across cores, we require that any task τ_i in P_k accesses only labels mapped in the corresponding local memory M_k. The set of all labels in M_k is denoted as $\mathcal{L}(M_k)$.

In the following, we consider the case where a shared variable d_x may be read by multiple tasks but can be written by one task only. This assumption reflects common programming practices in automotive systems, which avoid the need of introducing locking protocols to maintain consistency when multiple writers access the same variable [2]. It can be relaxed at the expense of storing additional memory buffers and a more complicate notation. Also, we consider the case where consumer and producer tasks may be mapped in different cores. This is also a common occurrence in many real-world applications.

3.3 Communications and Data Acquisition Deadline

To guarantee the temporal consistency of the shared variables across all functional dependencies, the *Logical Execution Time* paradigm is enforced in the system. Here we present the basic features that are required to understand the system structure.

To prevent access conflicts between tasks communicating under the LET semantics, lock-free implementations are used, and multiple labels are used as local copies for each shared variable d_x. For each variable d_x in a functional dependency $f_q = (\tau_p, \tau_c, d_x)$, a *pointer* $\rho_p^x(t)$ (resp., $\rho_c^x(t)$) is assigned to τ_p (resp., τ_c). Each pointer refers to the label (memory address) that the task must access at time t to write (resp., to read) the variable d_x. Pointers are updated over time to guarantee that a producer and a consumer task do not access the same local copy of the variable, while satisfying the LET semantics.

Communications are the concrete means used to guarantee the temporal consistency of the functional dependencies across the local labels. In this paper, a communication with index z occurring at time t, denoted with $c_z(t)$, moves the value of a variable d_x between its associated labels, and updates (e.g., swaps) the pointers associated with the (reader or writer) task. Depending on where the producer and consumer tasks are allocated, such communications can be either *intra-core* or *inter-core*. How to properly optimize such communications while enforcing the LET paradigm will be discussed in the next sections.

A job of τ_i released at time $t \in \mathcal{T}_i$ is *ready* when all the data it requires, according to its functional dependencies, are available in the labels pointed by its set of pointers. When the job is ready, it may start executing. 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 *while preserving the schedulability of* Γ. The maximum time elapsed between the (periodic) release of any job of τ_i and when it becomes ready is referred to as *data acquisition latency* and denoted with λ_i: thus, $\lambda_i \leq \gamma_i$ must be enforced to ensure schedulability.

4 The LET Semantics

In the original LET definition (Giotto) [1], every periodic instance of τ_i updates its input values at its release instant (*LET read*). The job then uses those values to compute new output values, which are made available to a consumer (*LET write*) only at the end of the period. We refer to LET writes and reads in general 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.

When dealing with real platforms, the time needed to move data and to execute tasks cannot be neglected. Thus, the zero-time communication assumption does not hold. All communications and executions must then be properly scheduled to retain causality as assessed by LET. In the following, we extract the key properties of LET, which will serve as guidelines to propose a label mapping and protocol for DMA-based communications under the LET paradigm.

4.1 Extracting the Core Properties of LET

LET communications in the application proposed in this paper are formally denoted as follows:

- **LET write** $W(\tau_p, d_x, t)$: task τ_p makes available to a consumer τ_c the instance of d_x produced during its job completed at t, and updates the pointer $\rho_p^x(t)$ to reserve a label for its next job; and
- **LET read** $R(d_x, \tau_c, t)$: task τ_c acquires the value of d_x available at t, and stores it in a local label pointed by $\rho_c^x(t)$.

In the following, when referring to a generic LET communication occurring at t, we will also use the abstract notation $c_z(t)$, where $c_z(t)$ can refer to either a LET write $W(\tau_p, d_x, t)$ or a LET read $R(d_x, \tau_c, t)$. Similarly to the functional dependencies, we introduce the operator $\in$ to denote whether τ_i is involved in $c_z(t)$.

In practical implementations, it is convenient to perform at the beginning of each period the LET writes that were 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 causally related. Introducing a partial order "$\prec$" between

communications (i.e., $a \prec b$ means that communication a must be completed before starting communication b), the previous statement is formally defined in Property 1.

Property 1 (LET communications to and from the same task)**.** *$W(\tau_i, d_a, t_{i,x}) \prec R(d_b, \tau_i, t_{i,x})$ must hold for any release time $t_{i,x} \in \mathcal{T}_i$ and for every variable $d_b \in \mathcal{F}(\tau_p, \tau_i)$ read by τ_i (and produced by any task τ_p), and every variable $d_a \in \mathcal{F}(\tau_i, \tau_c)$ written by τ_i and consumed by any τ_c. All LET communications involving τ_i must complete before executing the x-th job of τ_i.*

In addition, causal dependencies between tasks communicating using LET must be satisfied. At any point in time, LET writes of a producer task τ_p for a data $d_a \in \mathcal{F}(\tau_p, \tau_c)$ need to complete before starting the LET reads of τ_c. This is formally defined in Property 2.

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

As final requirement, under the realistic hypothesis that communications require a non-zero amount of time to be performed, LET communications issued at different time instants must not overlap, as 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 starting those assigned to time t_2.*

4.2 Identifying Necessary LET communications

Depending on the periods of the producer and consumer tasks, it is possible to safely skip those LET reads and writes that are *unnecessary* [5]. For example, a producer task τ_p with an undersampled consumer τ_c might skip some writes if that data will be overwritten before it is consumed by τ_c. Similarly, a consumer τ_c that is oversampling a producer τ_p may skip some of its reads if the value of the shared variable has not changed since its previous activation.

Considering a pair $\tau_p, \tau_i \in \Gamma$ such that $\mathcal{F}(\tau_p, \tau_i) \neq \emptyset$, the set of time instants where a LET write by τ_p is required is defined in [5] as $\{\eta^W_{p,i}(v) \cdot T_p \,|\, v \in \mathbb{N}\}$, 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, considering a pair $\tau_i, \tau_c \in \Gamma$ such that $\mathcal{F}(\tau_i, \tau_c) \neq \emptyset$, the set of time instants when a LET read is needed by τ_c is defined [5] as $\{\eta^R_{i,c}(v) \cdot T_c \,|\, v \in \mathbb{N}\}$, 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 LCM(T_i, T_p) and LCM(T_i, T_c), respectively [5]. 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:

$$H^*_i = \text{LCM}\left(T_i, \{T_j \,|\, \mathcal{F}(\tau_i, \tau_j) \neq \emptyset \vee \mathcal{F}(\tau_j, \tau_i) \neq \emptyset\}\right). \quad (3)$$

Building upon this observation, 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 $[0, H^*_i)$.

Algorithm 1 Constructing sets of LET communications

```
1:  function COMPUTE_LETGROUP (t, τ_i)
2:      G^W(t, τ_i) = ∅, G^R(t, τ_i) = ∅
3:      for τ_j ∈ Γ do
4:          for v ∈ ℕ, v < H*_i / T_i do
5:              if η^W_{i,j}(v)·T_i == t then
6:                  for d_x ∈ f_q = (τ_i, τ_j, d_x) do
7:                      G^W(t, τ_i) = G^W(t, τ_i) ∪ W(τ_i, d_x, t)
8:              if η^R_{j,i}(v) · T_i == t then
9:                  for d_x ∈ f_q = (τ_j, τ_i, d_x) do
10:                     G^R(t, τ_i) = G^R(t, τ_i) ∪ R(d_x, τ_i, t)
11:     return G^W(t, τ_i), G^R(t, τ_i)
```

The set of necessary LET writes and LET 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 are 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).

Finally, by introducing $\mathcal{T}^* = \bigcup_{\tau_i \in \Gamma} \mathcal{T}^*_i$, 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 [5], i.e., $\mathcal{C}(t) \subseteq \mathcal{C}(s_0)$, $\forall t \in \mathcal{T}^*$.

4.3 Implementing LET: the Giotto proposal

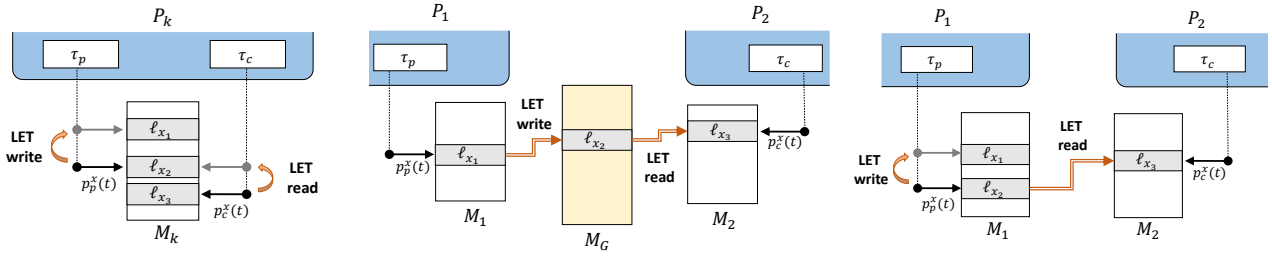
When considering a practical implementation, the authors of Giotto [1] proposed a strict order of execution for LET communications, that satisfies both Properties 1 and 2 (and under the hypothesis that the communication overhead is limited such that Property 3 also holds). At any time instant $t \in \mathcal{T}$ 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.

This implementation trivially satisfies the requirements of Properties 1 and 2, and it has been adopted in other works that consider the LET paradigm. 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 complete at t and start at t, even if such communications have no causal dependencies with τ_i. This may introduce unnecessary and harmful delays to latency-sensitive tasks, especially if heavy communication is required. Furthermore, high-priority tasks may need to wait for communications related to lower priority tasks, resulting in a priority inversion. The next section presents an alternative approach that solves such issues by leveraging the parallelism introduced by DMA engines.

5 LET Communications with DMA

This section proposes a set of possible designs and protocols to perform DMA-based LET communications. The



(a) Intra-core (b) Local-global-local inter-core (c) Local-to-local inter-core

Figure 1: Different label mapping configurations for LET communications, with the associated pointer and copy mechanism

design involves the mapping of shared variables in local or global memories, as well as the usage of different features of DMA engines to arrange data transfers. Our proposals allow for the following benefits: **(i)** a limited interference on the task executions, thanks to the usage of a DMA to offload data transfers; **(ii)** the possibility of finding a more flexible order of LET communications with respect to the Giotto proposal, which can be exploited to guarantee an early release for latency-sensitive tasks; **(iii)** the possibility of leveraging local-to-local transfers besides classical local-global-local ones to reduce delays in LET communications.

5.1 Mapping Shared Variables to Labels

First, we need to consider the actual mapping of variables in labels to separate the data manipulated by the tasks during their execution from the value logically defined by the LET abstraction. We distinguish three different cases.

5.1.1 *Intra-core communications*

When two tasks τ_p and τ_c in a functional dependency $f_l = (\tau_p, \tau_c, d_x)$ are mapped into the same core P_k, they are involved in an *intra-core* communication. Here, this kind of communications is managed using triple buffering [17].

In detail, for each variable d_x, three labels ℓ_{x_1}, ℓ_{x_2}, and ℓ_{x_3} are allocated in the local memory M_k, assigned to d_x. If more than one consumer exist in the same core, a similar mechanism can be applied by reserving a total of $2 + n_{cons}$ labels instead, where n_{cons} is the number of consumer tasks. LET reads and writes are implemented by updating the value of the pointers $\rho_p^x(t)$ and $\rho_c^x(t)$ assigned to τ_p and τ_c, respectively. Such LET communications are *not* managed by the DMA engine. An example is shown in Figure 1(a). At the beginning, $\rho_p^x(t)$ and $\rho_c^x(t)$ point to ℓ_{x_2} and ℓ_{x_3}, respectively. When the following instance of τ_p is released, $\rho_p^x(t)$ is switched to ℓ_{x_1}, which was unused, while ℓ_{x_2} retains the value produced by the previous instance of τ_p, which will become available for the next instance of τ_c, effectively performing a LET write. When the next job of τ_c is released, $\rho_c^x(t)$ is switched to point to ℓ_{x_2}, performing a LET read. $\rho_c^x(t)$ is moved only if there is a pending read. The process then continues, cyclically rotating the three labels.

5.1.2 *Inter-core local-global-local communications*

When tasks τ_p and τ_c in a functional dependency f_l execute on two different cores $\mathcal{P}(\tau_p) \neq \mathcal{P}(\tau_c)$, a possible label mapping consists in mimicking the local-global-local communications presented in [5]. Three labels are assigned to d_x: ℓ_{x_1} mapped in $M(\tau_p)$, ℓ_{x_3} in $M(\tau_c)$, and ℓ_{x_2} in the global memory M_G (Figure 1(b)).

In this case, both LET writes and reads are implemented as physical copies of data between labels, from $M(\tau_p)$ to M_G and from M_G to $M(\tau_c)$, respectively. The pointers $\rho_p^x(t)$ and $\rho_c^x(t)$ always point to ℓ_{x_1} and ℓ_{x_3}, respectively, and they are never switched. If d_x has multiple consumers, the mapping strategy still requires only one local copy in $M(\tau_p)$, one in the global memory M_G, and one each for every consumer task in their respective memories.

This scheme allows for a limited usage of memory space in local memories but requires a global memory M_G to allocate a copy of the label. Also, two physical communications are needed, accessing the (usually slower) global memory. Consequently, this option may be more suitable in platforms where local memories are small. Also intra-core communications can be managed using this design.

5.1.3 *Inter-core local-to-local communications*

When tasks τ_p and τ_c in a functional dependency f_l execute on two different cores, we can choose a different mapping that does not involve communicating by means of an intermediate copy in the global memory.

Also in this case, the communication of a variable d_x can be managed by using three labels ℓ_{x_1}, ℓ_{x_2}, and ℓ_{x_3}. The first two are assigned to $M(\tau_p)$, while label ℓ_{x_3} is allocated to $M(\tau_c)$ (Figure 1(c)). Labels ℓ_{x_1}, ℓ_{x_2} are handled with a pointer switching mechanism for τ_p such that, at each job release of τ_p, $\rho_p^x(t)$ switches between ℓ_{x_1} and ℓ_{x_2}. On the other hand, the pointer $\rho_c^x(t)$ of τ_c always refers to label ℓ_{x_3}. In this case, the producer task τ_p performs a LET write by swapping the pointer and storing the data produced in the previous job of τ_p in the label not currently pointed by $\rho_p^x(t)$. Conversely, the LET read requires a physical copy of data from the label not currently pointed by $\rho_p^x(t)$ (i.e., corresponding to the last available output of τ_p) to ℓ_{x_3}.

By trading one data copy with a pointer swap, this strategy helps in reducing the total communication overhead, but requires more space in the local memory of τ_p to store the buffer label. This mechanism can be generalized for the case where the variable d_x has multiple consumers: two local copies will be again required in $M(\tau_p)$, plus one label for each consumer task in their respective memories.

Alternatively, two labels could be reserved for each consumer task, and one for the producer. In this case, the LET write would be a physical copy to one of the labels at $M(\tau_c)$, while the LET read would consist of a pointer swap for the consumer task. However, when considering

one producer but possibly multiple consumers per variable, as in this paper, this translates in a higher total number of labels in local memories. For the sake of space, this case will not be explicitly treated in this paper. Nonetheless, our approach can be easily adapted to this case.

For the case of local-to-local strategy, depending on the state of $\rho_p^x(t)$ at the time of the LET read for a consumer task τ_c, the actual copy mandated by the LET read for a variable d_x involves two different pairs of labels ($\{\ell_{x_1}, \ell_{x_3}\}$ and $\{\ell_{x_2}, \ell_{x_3}\}$, respectively). Since τ_p will swap its pointer at each activation, if τ_c has an odd number of activations in the interval $[0, H)$, then after the hyperperiod the pattern of the label copies associated to LET reads of τ_c will be reversed. In this case, it is necessary that the set $\mathcal{T}^*$ of activations is properly computed across the interval $[0, 2 \cdot H)$ to account for all the different label pairings in the analysis.

5.2 DMA-Based LET Communication

In our application, a DMA engine is in charge of moving shared data from a source memory M_s to any other different destination memory M_d. Memories are dual-ported and each task may access data from its local memory only, allowing the DMA engine to perform LET communications on a core in parallel with the execution of tasks of the same core, provided that they are not accessing the same labels at the same time. Conversely, the *pointer swapping* mechanism is managed at the task level, and we assume it has a negligible overhead.

A DMA transfer is formally defined as a tuple $d_g(t) = \{\mathcal{C}_g(t), M_s, M_d, \mathcal{L}_g(t), a_{s,g}, a_{d,g}\}$, with $t \in \mathcal{T}^*$. Here, $\mathcal{C}_g(t)$ represents a set of *ordered* LET communications (write or read) involving one ore more tasks, i.e., $\mathcal{C}_g(t) = \{c_{g_1}(t), c_{g_2}(t), \dots\}$; M_s and M_d represent the source and destination memory of the transfer; $\mathcal{L}_g(t)$ represents the corresponding set of labels involved in the LET communications of $\mathcal{C}_g(t)$, in both memories; finally, $a_{s,g}$ and $a_{d,g}$ represent the start addresses at which labels in $\mathcal{L}_g(t)$ are contiguously allocated in M_s and M_d, respectively. The overall amount of data moved by the DMA transfer $d_g(t)$ is $\sum_{\ell_l \in \mathcal{L}_g(t)} \sigma_{\ell_l}$. The set of all the DMA transfers at t due to tasks from all cores is denoted by $\mathcal{D}(t) = \bigcup_g d_g(t)$.

To be suitable for a DMA transfer, $\mathcal{L}_g(t)$ and their copies must be all contiguously allocated both in M_s and M_d, and with the same order. Additionally, the index g of $d_g(t)$ represents the *order of execution* of that DMA transfer, with respect to the other transfers of $\mathcal{D}(t)$. To preserve the LET semantics, the index values must be carefully assigned so that both Properties 1 and 2 are always satisfied $\forall t \in \mathcal{T}^*$. All such aspects, including satisfying Property 3, must be verified at the design stage and will be reflected in the constraints of the optimization problem presented in Section 6.

5.2.1 Programming the DMA

To manage the DMA-based LET communication, a LET task $\tau_{\text{LET}} \in \Gamma$ with highest priority is added in one of the cores. The task τ_{LET} has the duty of dispatching those LET communications that require a physical copy by programming the DMA. Having only one LET task reduces the design complexity and allows to free all but one core from communication-related processing. This comes at the expense of a more significant interference on the core running the LET task (which could be, e.g., the one with the lowest utilization). Without loss of generality, we assume hereafter that τ_{LET} is mapped to P_N. Nonetheless, the following approach is adaptable to the case one LET task per core, by properly adapting the notation.

Multiple LET communications can be grouped in a single *DMA transfer*. Each single DMA transfer involves contiguous portions of memory, both in M_s and in M_d. Thus, only communications that share the same source and destination memories can be grouped in a single DMA transfer. To program a single data transfer using the DMA, τ_{LET} requires specifying: **(i)** the start address of the label to be copied in the source memory M_s, **(ii)** the start address in the destination memory M_d, **(iii)** the size of the data transfer.

5.2.2 Advanced DMA Transfers Configurations

Inspired by realistic hardware designs [7], in this paper we consider three possible DMA behaviors: $\mathcal{B} \in \{\texttt{SIMPLE}, \texttt{LL-EOT}, \texttt{LL-EOL}\}$, where `SIMPLE` represent the standard mode, `LL-EOT` stands for *linked-list end-of-transfer*, and `LL-EOL` for *linked-list end-of-list*.

- `SIMPLE` is the mode presented above: the DMA must be programmed every time it needs to perform a data transfer; at the end of the transfer, an interrupt is triggered.
- In `LL-EOT` and `LL-EOL` modes, the DMA can be programmed to perform *multiple* (i.e., a linked-list of) DMA transfers. The two linked-list behaviors differ by the fact that `LL-EOL` triggers an interrupt *only* when the entire list of transfers is completed, while `LL-EOT` triggers an interrupt at the end of each individual DMA transfer.

While the `SIMPLE` behavior considers the traditional case where each transfer $d_g(t)$, consisting of contiguously allocated labels, is programmed independently, `LL-EOT` and `LL-EOL` model more flexible DMA behaviors. In a linked list, two DMA transfers can then occur back to back (i.e., without processor intervention between them) even if they have non-contiguous labels, and between different memories. For consistency in the remainder of the paper, we introduce the notation for a linked-list occurring at a time t as $\text{dl}_v(t) = \{d_{v_1}(t), d_{v_2}(t), \dots\}$, which contains an ordered list of DMA data transfers that are programmed at the same time. When $\mathcal{B} = \texttt{SIMPLE}$ the transfer list will correspond to a single DMA transfer, i.e., $\text{dl}_v(t) = \{d_v(t)\}$.

5.2.3 Timing Characterization of DMA

We assume that a fixed amount of time $o_{\text{IN}} + o_{\text{DT}}$ is required from τ_{LET} to program a DMA transfer in `SIMPLE` mode, where o_{IN} is the initialization overhead of the DMA programming and o_{DT} is the overhead of uploading the command for a single DMA transfer, with $o_{\text{DT}} \leq o_{\text{IN}}$ [35]. This overhead is independent of the size of the (contiguous) labels to be transferred. After the programming phase completes, the transfer phase will have a fixed time cost ω_C per byte copied. An interrupt service routine (ISR) notifies the DMA completion: we assume that processing such ISR requires up to o_{ISR} time units.

The time spent by τ_{LET} to program a linked-list with multiple DMA transfers will be in general longer than programming one single transfer, but not longer than programming

each transfer separately [35]. We represent the overhead of programming a list $\mathrm{dl}_v(t)$ with the sum $o_{\mathrm{IN}} + o_{\mathrm{DT}} \cdot (|\mathrm{dl}_v(t)|)$. This formulation allows for the equivalence between a data transfer in `SIMPLE` mode and a linked-list having only one transfer. The cost per byte copied and the ISR overhead will have the same formulation as for the `SIMPLE` mode, i.e., ω_C per byte copied and o_{ISR} for processing the ISR.

5.3 Communication Protocol

In this section, we present a protocol to enforce the LET Properties at the level of the DMA transfers, under different configurations. For each $t \in \mathcal{T}^*$, let $\mathrm{dl}_v(t)$ be the transfer list programmed at 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 If $\mathcal{B} =$ `SIMPLE`, τ_{LET} programs the DMA for $\mathrm{dl}_v(t) = d_v(t)$ and suspends. Upon completion, an interrupt is raised to notify the termination of the data transfer to the corresponding task(s) whose data was involved in the transfer. The LET task is then awakened to handle the next transfer $d_{v+1}(t)$.

R3 If $\mathcal{B} =$ `LL-EOT`, τ_{LET} programs the DMA with a linked-list $\mathrm{dl}_v(t)$ with possibly multiple consecutive transfers. When a transfer in the list is finished, an interrupt is raised to notify the termination of the data transfer to the corresponding task(s) whose data was involved in the transfer. If the list reached the end, the LET task is awakened to handle the next transfer list $\mathrm{dl}_{v+1}(t)$.

R4 If $\mathcal{B} =$ `LL-EOL`, τ_{LET} programs the DMA with a linked-list $\mathrm{dl}_v(t)$ with possibly multiple consecutive transfers. When *all* transfers in the list are finished, an interrupt is raised to notify the termination of the data transfers to the corresponding task(s) whose data was involved in the transfers. If the list reached the end, the LET task is awakened to handle the next transfer list $\mathrm{dl}_{v+1}(t)$.

R5 When a DMA completion interrupt arrives, all the tasks for which the data dependencies are satisfied by the completed DMA transfers are marked as ready.

The rules in the proposed protocol guarantee that a job can become ready as soon as the interrupt of a DMA transfer notifies that its last LET communication has been performed. After that, the job can start executing (if it is the pending job with the highest priority), while other DMA transfers are still possibly occurring in parallel.

5.4 Observations about the Design Options

Finding the optimal grouping of LET communications in DMA transfers, as well as the order of such DMA transfers and the grouping in linked-lists, requires solving an optimization problem at design time. Moving multiple labels with a single transfer, in general, reduces the overhead, as it requires less processor intervention. This is beneficial for guaranteeing that the data acquisition deadlines are satisfied, while it complicates the label allocation problem. Furthermore, a single DMA transfer or list may involve data related to different tasks: this may cause additional delays, since tasks become ready only at the DMA completion ISR.

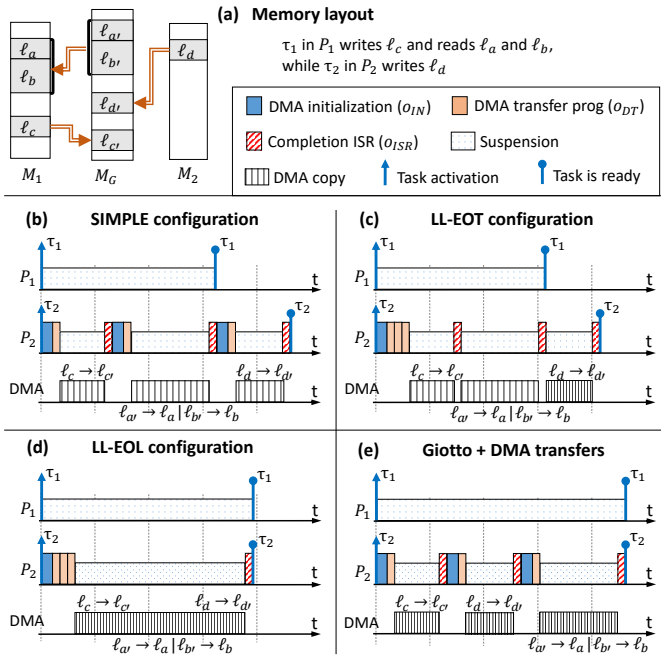


Figure 2: Example schedule of LET communications using a DMA engine with the three proposed approaches (insets (b)-(d)) and with the original Giotto approach [1] (inset (e)).

In conclusion, none of the linked-list approaches dominates the other. For instance, `LL-EOL` causes less overhead related to the processing of the DMA completion ISR, but it gives less control over when a task can start running. Indeed, a task may need to wait for potentially un-related communications of other tasks to be copied in the same DMA data transfer, since the interrupt is raised only at the completion of the list. This may translate in higher data acquisition latencies. Conversely, `LL-EOT` causes higher interference due to the DMA ISR, but provides more flexibility on the start of each task, since the processor is notified about the completion of each individual transfer.

Figure 2 shows an example schedule for LET communications of two tasks τ_1 in P_1 and τ_2 in P_2, using a DMA with the local-global-local mapping. At the beginning of the schedule, both tasks are activated. Task τ_1 requires a LET read of ℓ_a and ℓ_b and a LET write of ℓ_c, while τ_2 requires a LET write of ℓ_d. Inset (a) reports the memory layouts: ℓ_a and ℓ_b are mapped contiguously both in M_1 and in M_G, thus can be grouped in a single DMA transfer. Inset (b) illustrates the schedule obtained using the `SIMPLE` configuration: thanks to the proposed protocol, an optimized reordering of the communications is possible, where the LET reads of τ_1 are performed before the LET write of τ_2, which is useful if τ_1 is latency-sensitive. Inset (c) considers the `LL-EOT` configuration, where the three DMA transfers are mapped in one linked-list. While the LET write of τ_1 ends later than in case (b), both tasks are marked as ready earlier than in the `SIMPLE` case, due to an overall lower programming overhead. In inset (d), a single list under the `LL-EOL` configuration guarantees the earliest ready instant for τ_2, but also delays τ_1 at the same time. Finally, inset (e) shows the Giotto approach [1], when performing communications with the DMA; in this case, all LET writes are required to

be executed before the LET read of τ_1. The latency for both tasks is the greatest among all other cases.

5.5 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. Tasks running on P_k can be analyzed with response-time analysis techniques for periodic tasks, with a release jitter given by the data acquisition latency. For the core P_N, the schedulability involves also checking the LET task τ_{LET}. Since τ_{LET} runs at highest priority, under the considered model, it can be delayed only by the ISR associated with the DMA completion interrupt. Its schedulability is ensured by the optimization problem of Section 6 and is required for Property 3 to hold. Additionally, the LET task behaves as a generalized multiframe task [5], where each job exhibits a *segmented* self-suspending behavior [36]. When computing the high-priority interference on P_N, it is possible to model each execution segment of τ_{LET} as an independent sporadic task [36].

6 Optimization Problem

This section presents a mixed-integer linear programming (MILP) formulation to derive an optimal schedule of DMA-based LET transfers under the proposed protocol, with the relative ordering of labels in memories. The formulation handles local-global-local and local-to-local communication strategies, combined with any of the three DMA behaviors `SIMPLE`, `LL-EOT` and `LL-EOL` introduced before.

For each task in Γ, we assume that the strategy for inter-core communications has been chosen and that for each shared variable its corresponding set of labels has been assigned to the memories, as presented in Section 5.1. The label addresses are left as variables to be optimized.

6.1 MILP Variables

First, we introduce the main variables of the formulation. Here, $\mathbb{B}$ is the Boolean set and $\mathbb{R}$ is the set of real numbers. For brevity, all variables having index t are implicitly defined for all $t \in \mathcal{T}^*$.

- *Adjacency of labels:* $\text{A}_{k,a,b} \in \mathbb{B}$ is set to 1 if the address of ℓ_b is immediately below ℓ_a in M_k; otherwise it is set to 0.
- *LET communication in DMA transfer:* $\text{CD}_{t,z,g} \in \mathbb{B}$ is set to 1 if the LET communication $c_z(t) \in \mathcal{C}(t)$ is mapped in the g-th DMA transfer $d_g(t)$; otherwise it is set to 0.
- *Last communication of task in DMA transfer:* $\text{LCD}_{t,i,g} \in \mathbb{B}$ is set to 1 if the last LET communication of $\tau_i \in \Gamma$ occurring at t is in the g-th DMA transfer; otherwise it is set to 0.
- *Data acquisition latency of task:* $\text{LAT}_{t,i} \in \mathbb{R}$ contains the data acquisition latency experienced by $\tau_i \in \Gamma$ at t.

The following are the most prominent auxiliary variables.

- *DMA transfer not empty:* $\text{DF}_{t,g} \in \mathbb{B}$ is set to 1 if at least one communication is mapped in the DMA transfer $d_g(t)$; otherwise it is set to 0.
- *Position of label:* $\text{P}_{k,a} \in \mathbb{R}$ is equal to the relative position of the label ℓ_a mapped in M_k. $\text{P}_{k,a}$ is defined such that $\forall M_k \in \mathcal{M}$: $\sum_{\ell_a \in \mathcal{L}(M_k)} \text{P}_{k,a} = \sum_{i=1}^{|\mathcal{L}(M_k)|} i$.
- *Index of DMA transfer for LET communication:* $\text{CDI}_{t,z} \in \mathbb{R}$ is equal to the index g of the DMA transfer $d_g(t)$ where the communication $c_z(t)$ is mapped, i.e., $\forall c_z(t) \in \mathcal{C}(t)$, $\text{CDI}_{t,z} = \sum_g g \cdot \text{CD}_{t,z,g}$.
- *Index of DMA transfer for last LET communication:* $\text{LCDI}_{t,i} \in \mathbb{R}$ is equal to the index g of the DMA transfer $d_g(t)$ where the last LET communication of τ_i at t is mapped, i.e., $\forall \tau_i \in \Gamma$ such that $\exists\, c_z(t) \in \mathcal{C}(t)$ associated to τ_i, $\text{LCDI}_{t,i} = \sum_g g \cdot \text{LCD}_{t,i,g}$.

The latter variables in $\mathbb{R}$ represent integer values, but relaxed as reals to improve the performance of the optimization engine. Indeed, $\text{CDI}_{t,z}$ and $\text{LCDI}_{t,i}$ can only take integer values from the equality constraints on them. Similarly $\text{P}_{k,a}$ is constrained to integer values by Constraint 6.

Finally, we introduce two additional variables to be used only for the case $\mathcal{B} = \texttt{LL-EOL}$ or $\mathcal{B} = \texttt{LL-EOT}$.

- *DMA transfer in linked-list:* $\text{DL}_{t,g,v} \in \mathbb{B}$ is set to 1 if the DMA transfer $d_g(t)$ is mapped in the v-th linked-list $\text{dl}_v(t)$; otherwise it is set to 0.
- *LET communication in linked-list:* $\text{CL}_{t,z,v} \in \mathbb{B}$ is set to 1 if the LET communication $c_z(t)$ is mapped in the v-th linked-list; otherwise it is set to 0.

6.2 Main MILP Constraints

At a given activation instant $t \in \mathcal{T}^*$, the number of possible DMA transfers (as well as the number of linked-lists) is upper-bounded by the number of communications required at t, i.e., $|\mathcal{C}(t)|$. When using the notation $\sum_g$ and $\sum_v$ without explicit ranges, referring to DMA transfers and linked-lists, respectively, we then imply using a range from 1 to $|\mathcal{C}(t)|$.

In the constraints we use the notation $A \wedge B := X$ to denote an auxiliary Boolean variable X that contains the logical AND between the Boolean values A and B. This corresponds in MILP formulation to the set of the following inequalities: $X \leq A$, $X \leq B$ and $X \geq A + B - 1$. Furthermore, the notation $\max_s A_s := B$ corresponds to the set of inequalities: $\forall s$, $B \geq A_s$ and $B \leq A_s + (1 - X_s) \cdot M$, where X_s are Boolean auxiliary variables such that $\sum_s X_s = 1$, and M is a sufficiently-large positive constant value to represent infinity (commonly denoted as *big-M*).

We start by introducing the main constraints of the formulation, which are used in all DMA behaviors. A proof is presented for those constraints that are not intuitive.

6.2.1 Mapping Labels and Communications

Each communication $c_z(t) \in \mathcal{C}(t)$ must be mapped to exactly one DMA data transfer. This is checked in Constraint 1.

Constraint 1. *$\forall t \in \mathcal{T}^*$, $\forall c_z(t) \in \mathcal{C}(t)$, then: $\sum_g CD_{t,z,g} = 1$.*

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

Constraint 2. *$\forall t \in \mathcal{T}^*$, $\forall \tau_i \in \Gamma$, s.t. $\exists c_z(t) \in \mathcal{C}(t)$ with $\tau_i \in c_z(t)$, it holds that: $\sum_g LCD_{t,i,g} = 1$*

Constraint 3 enforces the definition of $\text{LCDI}_{t,i}$, stating that the index of the DMA data transfer performing the last LET communication of τ_i is computed as the maximum data transfer index of all communications of τ_i occurring at t.

Constraint 3. *$\forall t \in \mathcal{T}^*$ and $\forall \tau_i \in \Gamma$, it holds that: $LCDI_{t,i} = \max_{c_z(t) \in (G^R(t,\tau_i) \cup G^W(t,\tau_i))} CDI_{t,z}$*

Since the indexes of DMA transfers reflect their priority order, we drive the solver to fill the ones with lower indexes first, possibly leaving empty those with higher indexes. This is obtained using Constraint 4, which also enforces the definition of $\text{DF}_{t,g} \in \mathbb{B}$.

Constraint 4. *$\forall t \in \mathcal{T}^*$ and $\forall g = \{1, 2, \dots, |\mathcal{C}(t)|\}$ then:* ***(i)*** *$DF_{t,g} \leq \sum_z CD_{t,z,g}$;* ***(ii)*** *$DF_{t,g} \geq CD_{t,z,g}$, $\forall c_z(t) \in \mathcal{C}(t)$; and* ***(iii)*** *$DF_{t,g} \geq DF_{t,g+1}$.*

Proof. In inequality **(i)**, $\text{DF}_{t,g} \in \mathbb{B}$ is constrained to be 0 if $\forall z$, $\text{CD}_{t,z,g} = 0$, i.e., when no communications are in $d_g(t)$; the inequality has no effect if at least one of $\text{CD}_{t,z,g}$ is equal to 1. In a complementary way, the inequality **(ii)** forces $\text{DF}_{t,g}$ to be 1 if at least one of $\text{CD}_{t,z,g}$ is equal to 1, and has no effect if $\forall z$, $\text{CD}_{t,z,g} = 0$. Finally, the inequality **(iii)** guarantees that if $d_g(t)$ has no communications (i.e., $\text{DF}_{t,g} = 0$), then the immediately next DMA transfer will be empty too. □

Next, Constraint 5 states that, for each memory $M_k \in \mathcal{M}$, each label is allocated immediately below only one label, and immediately above another label. Dummy labels are provided at the beginning and at the end of the memory space to ensure the consistency of the constraint.

Constraint 5. *$\forall M_k \in \mathcal{M}, \forall \ell_a \in \mathcal{L}(M_k)$ it holds that:* $\sum_{\ell_s \in \mathcal{L}(M_k) \setminus \ell_a} A_{k,a,s} = 1$ *and* $\sum_{\ell_p \in \mathcal{L}(M_k) \setminus \ell_a} A_{k,p,a} = 1$

To obtain a unique position address for each label, we leverage the variable $\text{P}_{k,a}$ in the following constraint.

Constraint 6. *$\forall M_k \in \mathcal{M}, \forall \ell_a, \ell_b \in \mathcal{L}(M_k), \ell_a \neq \ell_b$, then:* $P_{k,a} + 1 - (1 - A_{k,a,b}) \cdot M \leq P_{k,b} \leq P_{k,a} + 1 + (1 - A_{k,a,b}) \cdot M$, *where M is a large positive constant value that represents infinity.*

Proof. If ℓ_b is mapped immediately below ℓ_a in M_k ($\text{A}_{k,a,b} = 1$), then the position index of ℓ_b is equal to the one of ℓ_a plus 1. If $\text{A}_{k,a,b} = 0$ the constraint has no effect. □

Next, we must ensure that all those labels that are involved in the same DMA transfer are contiguous in the same order, in both the source and destination memories. This is equivalent to check that, if any two communications $c_x(t)$ and $c_y(t)$ are in $d_g(t)$, such that $c_x(t)$ copies data from $\ell_a \in M_s$ to $\ell_{a'} \in M_d$ and $c_y(t)$ copies data from $\ell_b \in M_s$ to $\ell_{b'} \in M_d$, then either: **(i)** the labels of $c_x(t)$ and $c_y(t)$ are adjacent in both memories; or **(ii)** there exists at least one other communication in $d_g(t)$ whose labels are adjacent to either the ones of $c_x(t)$ or $c_y(t)$, in both memories. Both cases can be generalized by stating that it exists at least one pair of labels $\ell_c \in M_s$ and $\ell_{c'} \in M_d$, involved in a copy during $d_g(t)$, that are mapped immediately below the pair of either $c_x(t)$ or $c_y(t)$ in both memories, *where it can even be $\ell_c = \ell_a$ and $\ell_{c'} = \ell_{a'}$, or $\ell_c = \ell_b$ and $\ell_{c'} = \ell_{b'}$*. This is formally enforced in MILP formulation by Constraint 7.

Constraint 7. *$\forall t \in \mathcal{T}^*$, for each pair $c_x(t), c_y(t) \in \mathcal{C}(t)$, $x \neq y$, sharing the same source M_s and destination memory M_d, such that $c_x(t)$ involves copying data from ℓ_a to $\ell_{a'}$ and $c_y(t)$ involves copying data from ℓ_b to $\ell_{b'}$, $\forall g \in \{1, 2, \dots |\mathcal{C}(t)|\}$ it holds that:*

$$(CD_{t,x,g} \wedge CD_{t,y,g}) \leq \sum_{c_z(t) \in Z(t)} \left(ACD^z_{t,a,c,g} + ACD^z_{t,b,c,g}\right)$$

where $c_z(t)$ is any communication with source M_s and destination M_d, copying data from ℓ_c to $\ell_{c'}$ and $ACD^z_{t,\star,c,g} = (A_{s,\star,c} \wedge A_{d,\star',c'} \wedge CD_{t,z,g})$.

Proof. If $c_x(t)$ and $c_y(t)$, which move data from local memory M_s to memory M_d, 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)** the labels ℓ_a, ℓ_b in M_s are mapped in adjacent memory slots and $\ell_{a'}$, $\ell_{b'}$ in M_d are mapped in adjacent memory slots, or **(ii)** at least one between ℓ_a and ℓ_b is adjacent to another label ℓ_c in M_s that is part of the same g-th DMA transfer (the same holds for $\ell_{a'}$ and $\ell_{b'}$ with $\ell_{c'}$ in M_d). In both cases, for the constraint to hold, the RHS must also be set to at least 1. Note that $\text{ACD}^z_{t,a,c,g} = 1$ if and only if label ℓ_c is mapped below ℓ_a in M_s, and $\ell_{c'}$ is mapped below $\ell_{a'}$ in M_d ($\text{A}_{s,a,c} = \text{A}_{d,a',c'} = 1$) and $c_z(t)$ is in the DMA transfer of index g ($\text{CD}_{t,z,g} = 1$). This is consistent with the above points (i), when $c_z(t) = c_y(t)$, and (ii). A dual reasoning can be made if $\text{ACD}^z_{t,b,c,g} = 1$.

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

6.2.2 Linked-Lists Properties

This section presents the constraints related to the linked-list, to be used only when $\mathcal{B} = \texttt{LL-EOL}$ or $\mathcal{B} = \texttt{LL-EOT}$.

First, an arbitrary DMA transfer $d_g(t)$ can be mapped in a linked list only if at least one communication is mapped in that transfer. Then, since the indexing of the linked-lists matches their priority order, we also enforce that $d_g(t)$ cannot be mapped in a linked-list with index greater than g. Such properties are coded in Constraint 8.

Constraint 8. *$\forall t \in \mathcal{T}^*$, $\forall g \in \{1, 2, \dots |\mathcal{C}(t)|\}$, it holds that:* $\sum_{v=1}^{g} DL_{t,g,v} = DF_{t,g}$ *and* $\sum_{v=g+1}^{|\mathcal{C}(t)|} DL_{t,g,v} = 0$.

Since at least one DMA transfer is issued at each $t \in \mathcal{T}^*$, as well as at least one linked-list, the equality $\text{DL}_{t,1,1} = 1$ derives from Constraint 8.

Then, Constraint 9 enforces that the linked-lists are filled in order, i.e., an arbitrary DMA transfer $d_g(t)$, with $g > 1$, can be either mapped in the linked-list that contains $d_{g-1}(t)$, or in the list that follows next.

Constraint 9. *$\forall t \in \mathcal{T}^*$, $\forall g \in \{2, \dots |\mathcal{C}(t)|\}$, it holds that:* $DL_{t,g,v} + DL_{t,g,v+1} \leq DF_{t,g} + (1 - DL_{t,g-1,v}) \cdot M$, *and* $DL_{t,g,v} + DL_{t,g,v+1} \geq DF_{t,g} - (1 - DL_{t,g-1,v}) \cdot M$.

Proof. When $d_{g-1}(t)$ is mapped in the v-th linked-list, then $\text{DL}_{t,g-1,v} = 1$ and $\text{DL}_{t,g,v} + \text{DL}_{t,g,v+1} = \text{DF}_{t,g}$, which means that if $d_g(t)$ contains at least one communication ($\text{DF}_{t,g} = 1$), then $d_g(t)$ is eifther in the linked-list with index v or $v+1$. When $\text{DL}_{t,g-1,v} = 0$, the constraint has no effect. □

The definition of $\text{CL}_{t,z,v}$ is enforced by checking if $c_z(t)$ is mapped in any $d_z(t)$ that is listed in $\text{dl}_v(t)$, as follows.

Constraint 10. *$\forall t \in \mathcal{T}^*$, $\forall c_z(t) \in \mathcal{C}(t)$ and $\forall v \in \{1, 2, \dots |\mathcal{C}(t)|\}$, it holds that:* $CL_{t,z,v} = \sum_g (CD_{t,z,g} \wedge DL_{t,g,v})$.

$$\mathrm{LAT}_{t,i} \geq (o_{\mathrm{IN}} + o_{\mathrm{DT}} + o_{\mathrm{ISR}}) \cdot \mathrm{LCDI}_{t,i} + \omega_C \cdot \left(\sum_{g=1}^{\overline{g}} \sum_{c_z(t) \in \mathcal{C}(t)} \sigma_l \cdot \mathrm{CD}_{t,z,g} \right) - (1 - \mathrm{LCD}_{t,i,\overline{g}}) \cdot M \quad (4)$$

$$\mathrm{LAT}_{t,i} \geq (o_{\mathrm{IN}} + o_{\mathrm{ISR}}) \cdot \overline{v} + o_{\mathrm{DT}} \cdot \left(\sum_{v=1}^{\overline{v}} \sum_{d} \mathrm{DL}_{t,d,v} \right) + \omega_C \cdot \left(\sum_{v=1}^{\overline{v}} \sum_{c_z(t) \in \mathcal{C}(t)} \sigma_l \cdot \mathrm{CL}_{t,z,v} \right) - \left(1 - \sum_{g} (\mathrm{LCD}_{t,i,g} \wedge \mathrm{DL}_{t,g,\overline{v}}) \right) \cdot M \quad (5)$$

$$\mathrm{LAT}_{t,i} \geq o_{\mathrm{IN}} \cdot \overline{v} + o_{\mathrm{ISR}} \cdot \mathrm{LCDI}_{t,i} + o_{\mathrm{DT}} \cdot \left(\sum_{v=1}^{\overline{v}} \sum_{d} \mathrm{DL}_{t,d,v} \right) + \omega_C \cdot \left(\sum_{g=1}^{\overline{g}} \sum_{c_z(t) \in \mathcal{C}(t)} \sigma_l \cdot \mathrm{CD}_{t,z,g} \right) - (2 - \mathrm{LCD}_{t,i,\overline{g}} - \mathrm{DL}_{t,\overline{g},\overline{v}}) \cdot M \quad (6)$$

6.2.3 LET Properties and Data Acquisition Deadlines

Property 1 is enforced through Constraint 11, by imposing that any LET write of τ_i is mapped in a DMA transfer with index lower than the one containing any LET read of τ_i.

Constraint 11 (Property 1). *$\forall t \in \mathcal{T}^*$, $\forall \tau_i \in \Gamma$, for each pair $c_w(t) \in G^W(t, \tau_i)$ and $c_r(t) \in G^R(t, \tau_i)$: $CDI_{t,w} < CDI_{t,r}$.*

Constraint 12 enforces Property 2, i.e., for each functional dependency, the corresponding LET write must be completed before the LET read.

Constraint 12 (Property 2). *$\forall t \in \mathcal{T}^*$, for each pair $c_w(t), c_r(t) \in \mathcal{C}(t)$ with $c_w(t) = W(\tau_p, d_x, t)$ and $c_r(t) = R(d_x, \tau_c, t)$, then: $CDI_{t,w} < CDI_{t,r}$.*

For those tasks that use local-to-local communications under the label mapping of Section 5.1.3, the LET writes are performed as pointer swaps by the task itself. In this case, Properties 1 and 2 must be constrained by design, guaranteeing that the pointer swaps are completed before performing the LET reads with the DMA.

The communication latency $\mathrm{LAT}_{t,i}$ experienced by τ_i at $t \in \mathcal{T}^*$ is computed by accumulating the delays of all the communications occurring at t until the completion of the ISR that marks τ_i as ready to execute. Constraint 13 provides a bound for the three possible DMA configurations.

Constraint 13. *$\forall t \in \mathcal{T}^*$, $\forall \tau_i \in \Gamma$, s.t. $\exists c_z(t) \in \mathcal{C}(t)$ with $\tau_i \in c_z(t)$, the inequality $LAT_{t,i} \leq \gamma_i$ must hold, and:*
- *if $\mathcal{B} =$ SIMPLE, $\forall \overline{g} \in \{1, 2, \ldots |\mathcal{C}(t)|\}$ Equation (4) holds;*
- *if $\mathcal{B} =$ LL-EOL, $\forall \overline{g} \in \{1, 2, \ldots |\mathcal{C}(t)|\}$ Equation (5) holds;*
- *if $\mathcal{B} =$ LL-EOT, $\forall \overline{g}, \overline{v} \in \{1, 2, \ldots |\mathcal{C}(t)|\}$ Equation (6) holds;*

where σ_l is the size of the label involved in $c_z(t)$.

Proof. The proof is provided for each DMA configuration.

For $\mathcal{B} =$ SIMPLE, assume that the last LET communication of τ_i at t occurs in the DMA transfer with index $\overline{g}$. If this is not the case, the last big-M term in the RHS of Equation (4) makes the constraint inactive. When the constraint is active, τ_i is released at the completion of the DMA transfer with index $\mathrm{LCDI}_{t,i}$; each transfer generates a (worst-case) overhead given by the programming plus interrupt costs, i.e., $o_{\mathrm{IN}} + o_{\mathrm{DT}} + o_{\mathrm{ISR}}$: 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.

For $\mathcal{B} =$ LL-EOL, the sum $\sum_g (\mathrm{LCD}_{t,i,g} \wedge \mathrm{DL}_{t,g,\overline{v}})$ in Equation (5) is equal to 1 if the last communication of τ_i at t occurs in any DMA mapped in the $\overline{v}$-th linked-list. The big-M term makes the constraint inactive if this condition does not hold. When the constraint is active, $\overline{v}$ linked-lists are processed until the release of τ_i. Each list requires an initialization overhead of o_{IN}, plus o_{ISR} for the interrupt and o_{DT} for each DMA transfer mapped in the list (accounted for in the first two terms in the RHS). The third term accounts for the cost of transferring the labels involved in the first $\overline{v}$ lists, by summing all variables $\mathrm{CL}_{t,z,v}$ multiplied by ω_C.

For $\mathcal{B} =$ LL-EOT, assume that the last LET communication of τ_i occurs at the $\overline{g}$-th DMA transfer mapped in the $\overline{v}$-th list. If this does not hold, the big-M term in the RHS of Equation (6) makes the constraint inactive. Each list requires an initialization overhead of o_{IN} and o_{DT} for each transfer in the list. Since interrupts are issued at the completion of each DMA transfer, τ_i experiences $\mathrm{LCDI}_{t,i}$ interrupts of cost o_{ISR} before being ready. Those components are the first three terms in the RHS. The contribution due to the transfer labels is then identical to the case of SIMPLE. □

Finally, Property 3 is enforced by Constraint 14.

Constraint 14 (Property 3). *$\forall t \in \mathcal{T}^*$, $\forall \tau_i \in \Gamma$, then:*

$$LAT_{t,i} \leq t^{next} - t,$$

where t^{next} is the instant in $\mathcal{T}^*$ occurring immediately after t.

6.3 Objective Function

Two objective functions are proposed: minimizing the maximum number of DMA transfers in any $t \in \mathcal{T}^*$, or the maximum ratio between the communication delay and the task's period, i.e.:

$$\textbf{minimize} \quad \max_{t \in \mathcal{T}^*, \tau_i \in \Gamma} \left(\mathrm{LCDI}_{t,i} \right), \quad (7)$$

$$\text{or} \quad \textbf{minimize} \quad \max_{t \in \mathcal{T}^*, \tau_i \in \Gamma} \left(\mathrm{LAT}_{t,i} / T_i \right). \quad (8)$$

The formulation can be extended to cope with other objectives and constraints, e.g., minimizing or limiting memory fragmentation.

7 Experimental Results

We apply the approaches presented in the paper to a case study representative of an autonomous driving application, proposed by Bosch for the WATERS 2019 Industrial Challenge [37]. The parameters of tasks, labels and the data dependencies are provided with the case study, while the task mapping is based on the challenge solution of [38]. Figure 5 shows a schematic representation of the application.

We performed the experiments considering $o_{\mathrm{IN}} + o_{\mathrm{DT}} = 3.36\mu s$ from the results of measurements from [25], and the relation $o_{\mathrm{IN}} \approx 2 \cdot o_{\mathrm{DT}}$ from [35], with the delay due to a DMA completion interrupt equal to $o_{\mathrm{ISR}} = 10\mu s$. The data acquisition deadlines of the tasks have been derived

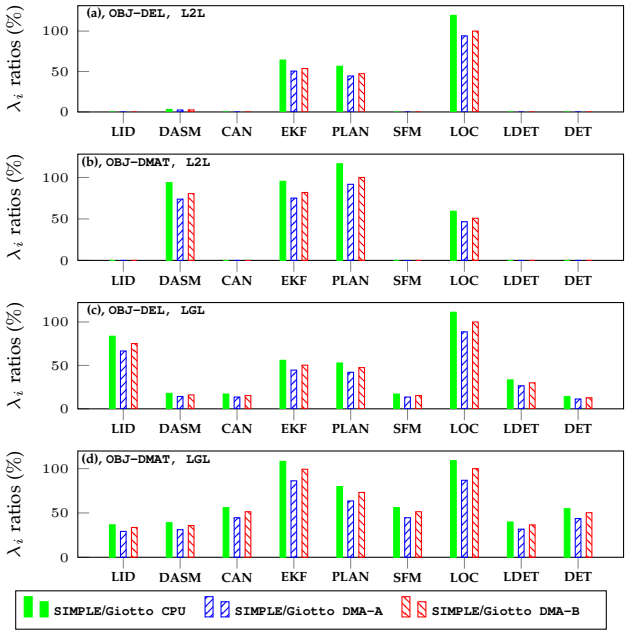


Figure 3: Delays λ_i obtained with our base DMA protocol (SIMPLE) compared with other Giotto-based proposals.

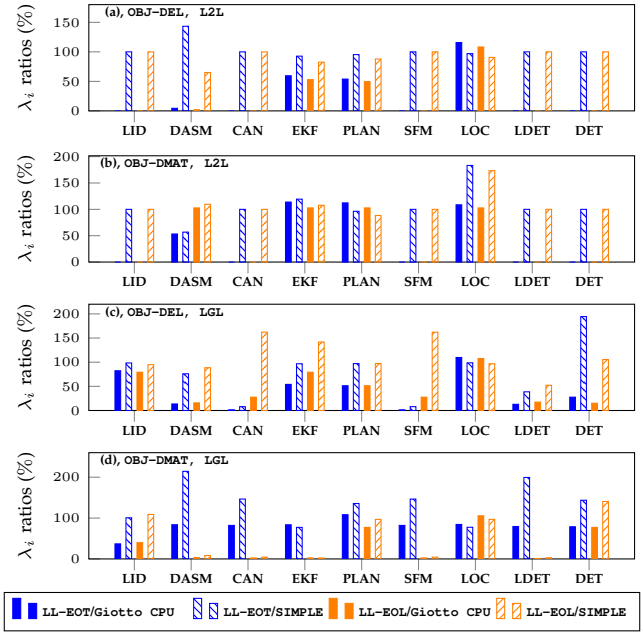


Figure 4: Delays λ_i obtained with the linked-list approaches compared with SIMPLE DMA behavior and Giotto-CPU.

with the following sensitivity analysis procedure. First, we computed the worst-case response time R_i of each task $\tau_i \in \Gamma$ of the Challenge, and the slack $S_i = D_i - R_i$. Then, we set $\gamma_i = \alpha \cdot S_i$, and we checked the schedulability using γ_i as a bound on the jitter, with $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Due to space limitations, we present only the results for $\alpha = 0.2$, which provides the shortest deadlines that still guarantee feasible solutions.

We evaluated six different variants of the approaches described in this paper, combining the case where **(a)** all

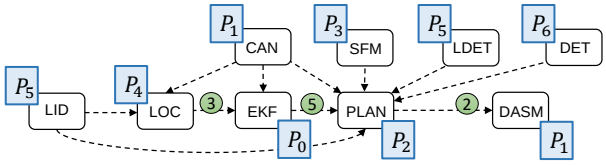


Figure 5: Task chains of the WATERS 2019 Challenge. P_x is the processor where the task is mapped, while in green are the number of labels exchanged (1 in the untagged arrows).

Table 1: Observed running times (T = timeout), number of DMA transfers (DT), and number of linked lists (LL).

Strategy	MILP runtime		# DT/ # LL	
	OBJ-DMAT	OBJ-DEL	OBJ-DMAT	OBJ-DEL
SIMPLE-L2L	9 sec	38 sec	11 / -	13 / -
EOT-L2L	73 sec	31 sec	11 / 5	13 / 3
EOL-L2L	112 sec	1h 48min	11 / 1	11 / 5
SIMPLE-LGL	29 min	12 min	12 / -	15 / -
EOT-LGL	1h 3min	4 hours (T)	12 / 10	16 / 3
EOL-LGL	1h 59min	4 hours (T)	12 / 7	13 / 10

communications are performed as local-global-local (labeled LGL) and **(b)** all communications are performed as local-to-local (L2L), with the three DMA configurations considered in this work, namely: the SIMPLE DMA behavior, LL-EOT, and LL-EOL. For each combination, we also considered three different baseline approaches for comparison: **(i)** the state-of-the-art Giotto approach [1], with LET copies performed by the CPU (Giotto-CPU), **(ii)** the Giotto approach enhanced with the usage of a DMA but without the communication reordering 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 **(iii)** the Giotto approach with DMA and using the memory layout and transfer grouping found by the optimization problem for the case at point (i) (Giotto-DMA-B), but without the communication reordering (i.e., still requiring all LET writes to be completed before the LET reads).

Each combination is applied to the two objective functions presented in Section 6.3: minimizing the number of DMA transfers (Equation (7), OBJ-DMAT), and minimizing the λ_i/T_i ratio (Equation (8), 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 formulation has been coded in C++ and solved with IBM CPLEX, setting a timeout of 4 hours.

Evaluation of SIMPLE-based Approaches. The results of the first set of experiments are reported in Figure 3, showing the advantages of the baseline approach SIMPLE when compared with the Giotto-based solutions. Four representative configurations are reported in Fig. 3. The X-axis reports the nine tasks of the WATERS 2019 Challenge, namely, LID, DASM, CAN, EKF, PLAN, SFM, LOC, LDET, and DET, while the Y-axis shows the ratio of the data acquisition latency λ_i of each task τ_i obtained solving our optimization, over the one obtained with the three alternative approaches. Fig. 3 (a) and (c) show the results obtained when minimizing the λ_i/T_i ratio, targeting the L2L and LGL case, respectively. The plots show considerable improvements to all previously available approaches. In both cases, our

solution enables improved parallelism and reorders the communications allowing an early start for most of the tasks. Among the baseline approaches, Giotto-CPU has values closest to SIMPLE, then Giotto-DMA-B performs better than Giotto-DMA-A. This can be explained as in the case under analysis, the communication reordering has the highest impact in reducing the latencies, while Giotto-CPU benefits from not having the programming overhead.

In the L2L case, all write-only tasks (LID, CAN, SFM, LDET, and DET) achieve a communication delay equal to zero, since all their LET writes are managed by switching pointers (with negligible cost), and our protocol allows them to be immediately ready. Still in the L2L case, tasks with LET reads achieves improvement up to 98% (i.e., DASM task in Fig. 3 (a)). Conversely, in the LGL case, the maximum improvement of 88% is achieved by the DET task (Fig. 3 (c)).

Similar trends can be observed in Fig. 3 (b) and (d), which minimize the number of DMA transfers. In these charts, the ratios λ_i are slightly higher because the solver tends to privilege a lower number of DMA transfers over an improved λ_i. Table 1 summarizes the number of DMA transfers and the running times for all experiments.

Evaluation of Linked-Lists Approaches. The second set of experiments targets LL-EOT and LL-EOL configurations. Here, the complexity of the formulation increases with the addition of the variables and constraints associated to the linked-list. Two tests did not converge to the optimal mapping before the timeout (see Table 1), but the solver was still able to provide feasible (even if suboptimal) solutions. Figure 4 summarizes the results of the latency, comparing the one found using the linked-list approaches, with the one of SIMPLE strategy (dashed columns) and with the baseline Giotto-CPU approach (filled columns). Table 1 reports both the number of transfers and lists.

In Figure 4(a) and (b), we target the L2L mapping. Here, Figure 4(a) shows that LL-EOT and LL-EOL provide similar latencies with respect to the mapping obtained with SIMPLE (most dashed columns are close to 100%), while still beating the Giotto-based approach for the vast majority of the tasks. We can see, for example, how the DASM task of LL-EOL in inset (a) has a worse latency with respect to the SIMPLE one, but in reality their difference in absolute terms is little ($39\mu s$ vs. $27\mu s$), and still outperforms the latency obtained with Giotto-CPU ($892\mu s$). Figure 4(b) shows slightly higher ratios because again the solver privileges a lower number of DMA transfers with respect to latency. In the L2L case, the linked-list approaches get the best improvements over SIMPLE for LL-EOT in Figure 4(b), up to 45%, and for LL-EOL in Figure 4(a), up to 35%. Figure 4(c) and (d) show the results for the LL-LGL mapping. Here, the behavior is varied, also related to the fact that two experiments for OBJ-DEL reached timeout and the solution is possibly not optimal. This is more evident, for example, for the DET task in the LL-EOL case (Figure 4(c)) or for the CAN and SFM tasks for LL-EOT (Figure 4(d)). Comparing the linked-list approaches with SIMPLE in the LGL case, the best improvements are achieved in Figure 4(c), reaching up to 93% and 50% for LL-EOT and LL-EOL, respectively (for the CAN and LDET tasks).

Finally, Table 1 gives additional insight about the number of DMA transfers and lists. Both linked-lists approaches provided the same number of transfers obtained with SIMPLE when testing OBJ-DMAT, with LL-EOT preferring a mapping in a higher number of lists with respect to LL-EOL. Conversely, when minimizing λ_i/T_i, as expected, LL-EOL is driven to increase the number of lists with respect to LL-EOT, to provide an early release for the tasks.

8 Conclusions

DMA engines offer excellent opportunities to enhance the performance of automotive systems. However, their introduction calls for re-purposing some consolidated assumptions in implementing the core principles of LET. To this end, we presented a set of new protocols to perform LET communications while leveraging the parallelism offered by a DMA engine. This paper provided methods for finding an optimized scheduling and memory allocation of the LET communications of different tasks using a MILP formulation. The proposed methods have been compared experimentally with the Giotto approach with core-commanded data transfers, providing solutions for the WATERS 2019 Challenge task set that allow respecting the data acquisition deadline constraints.

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] M. Hassan and R. Pellizzoni, "Bounding dram interference in cots heterogeneous mpsocs for mixed criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[4] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "A Holistic Memory Contention Analysis for Parallel Real-Time Tasks under Partitioned Scheduling," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

[5] 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.

[6] 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)*, 2019.

[7] Infineon, "General Purposes Direct Memory Access (GPDMA)."

[8] ——, "STM32U5-System-DMA Linked list (DMALL) Rev1.0."

[9] ATMEL, " AT17417: Usage of XDMAC on SAM S/SAM E/SAM."

[10] P. Pazzaglia, D. Casini, A. Biondi, and M. Di Natale, "Optimal memory allocation and scheduling for dma data transfers under the let paradigm," in *58th Design Automation Conference*, 2021.

[11] R. Ernst, S. Kuntz, S. Quinton, and M. Simons, "The logical execution time paradigm: New perspectives for multicore systems (dagstuhl seminar 18092)," *Dagstuhl Reports*, vol. 8, 2018.

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

[13] S. Igarashi, T. Ishigooka, T. Horiguchi, R. Koike, and T. Azumi, "Heuristic contention-free scheduling algorithm for multi-core processor using let model," in *IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications*, 2020.

[14] A. Yano, S. Igarashi, and T. Azumi, "Contention-free scheduling algorithm using let paradigm for clustered many-core processor," in *2021 IEEE/ACM 25th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2021, pp. 1–4.

[15] K.-B. Gemlau, L. Köhler, R. Ernst, and S. Quinton, "System-level logical execution time: Augmenting the logical execution time paradigm for distributed real-time automotive software," *ACM Trans. Cyber-Phys. Syst.*, 2021.

[16] J. Martinez, I. Sañudo, and M. Bertogna, "Analytical characterization of end-to-end communication delays with logical execution time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
[17] ——, "End-to-end latency characterization of task communication models for automotive systems," *Real-Time Systems*, 2020.
[18] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016.
[19] ——, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, 2017.
[20] M. Günzel, K.-H. Chen, N. Ueter, G. v. d. Brüggen, M. Dürr, and J.-J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021.
[21] A. M. Kordon and N. Tang, "Evaluation of the Age Latency of a Real-Time Communicating System Using the LET Paradigm," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, 2020.
[22] S. Saidi, P. Tendulkar, T. Lepley, and O. Maler, "Optimizing explicit data transfers for data parallel applications on the cell architecture," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, pp. 1–20, 2012.
[23] ——, "Optimizing two-dimensional dma transfers for scratchpad based mpsocs platforms," *Microprocessors and Microsystems*, vol. 37, no. 8, pp. 848–857, 2013.
[24] S. Wasly and R. Pellizzoni, "Hiding memory latency using fixed priority scheduling," in *19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.
[25] 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.
[26] 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.
[27] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for cots-based embedded systems," in *17th Real-Time and Technology and Applications Symposium (RTAS)*, 2011.
[28] S. Wasly and R. Pellizzoni, "A dynamic scratchpad memory unit for predictable real-time embedded systems," in *2013 25th Euromicro Conference on Real-Time Systems*, 2013.
[29] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, and M. Caccamo, "A Real-Time Scratchpad-Centric OS for Multi-Core Embedded Systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.
[30] 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.
[31] 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.
[32] 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.
[33] 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.
[34] 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.
[35] S. Saıdi, "Optimizing dma data transfers for embedded multi-cores," *PhD disseratation, univercity of Grenovale*, 2012.
[36] J.-J. Chen *et al.*, "Many suspensions, many problems: a review of self-suspending tasks in real-time systems," *Real-Time Systems*, 2018.
[37] A. Hamann, D. Dasari, F. Wurst, I. Sañudo, N. Capodieci, P. Burgio, and M. Bertogna, "WATERS Industrial Challenge 2019," Re-uploaded at https://retis.sssup.it/~d.casini/resources/WATERS2019/WATERS_Industrial_Challenge_2019_final.pdf.
[38] 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.

Paolo Pazzaglia (IEEE Member) is a Postdoctoral researcher at the Computer Science Department, Saarland University. He completed his Ph.D. in computer engineering (with honors), at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna, Pisa, in 2020. He was a visiting Ph.D. student at the Department of Automatic Control, Lund University, in the winter semester 2018/19. His research is at the intersection of control systems and real-time systems, with the goal of improving robustness and enforcing determinism in modern embedded control applications.

Daniel Casini (IEEE Member) is Assistant Professor at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. He graduated (cum laude) in Embedded Computing Systems Engineering, a joint Master degree by the Scuola Superiore Sant'Anna of Pisa and University of Pisa, and received a Ph.D. in computer engineering at the Scuola Superiore Sant'Anna of Pisa (with honors). In 2019, he has been visiting scholar at the Max Planck Institute for Software Systems (Germany). His research interests include software predictability in multi-processor systems, schedulability analysis, synchronization protocols, and the design and implementation of real-time operating systems and hypervisors.

Alessandro Biondi (IEEE Member) is Assistant Professor at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna. He graduated (cum laude) in Computer Engineering at the University of Pisa, Italy, within the excellence program, and received a Ph.D. in computer engineering at the Scuola Superiore Sant'Anna. In 2016, he has been visiting scholar at the Max Planck Institute for Software Systems (Germany). His research interests include design and implementation of real-time operating systems and hypervisors, schedulability analysis, cyber-physical systems, synchronization protocols, and component-based design for real-time multiprocessor systems. He was recipient of six Best Paper Awards, one Outstanding Paper Award, the ACM SIGBED Early Career Award 2019, and the EDAA Dissertation Award 2017.

Marco Di Natale (IEEE Senior Member) is a Full Professor at the Scuola Superiore SantAnna and IEEE Senior member. He has been visiting researcher at the University of California, Berkeley, in 2006 and 2008/09 and a researcher in the area of real-time systems and embedded systems for more than 20 years; winner of six Best Paper Awards and the Archie T. Colwell Award. He has served as Program Chair, Track Chair, and General chair and has been organizer of tutorials and special sessions for the main conferences in the area, including the Real-time Application Symposium (RTAS), the Design Automation Conference (DAC), the Design Automation and Test in Europe (DATE) and the IEEE SIES and ICESS. He is currently on the Editorial Board of the IEEE Transactions on Industrial Informatics and the Springer Real-Time Systems Journal.