Optimizing Inter-Core Data-Propagation Delays in Industrial Embedded Systems under Partitioned Scheduling

Lamija Hasanagić
lhc19001@student.mdh.se
Mälardalen University
Västerås, Sweden

Tin Vidović
tvc19001@student.mdh.se
Mälardalen University
Västerås, Sweden

Saad Mubeen
saad.mubeen@mdh.se
Mälardalen University
Västerås, Sweden

Mahammad Ashjaei
mohammad.ashjaei@mdh.se
Mälardalen University
Västerås, Sweden

Matthias Becker
mabecker@kth.se
KTH Royal Institute of Technology
Stockholm, Sweden

ABSTRACT

This paper addresses the scheduling of industrial time-critical applications on multi-core embedded systems. A novel scheduling technique under partitioned scheduling is proposed that minimizes inter-core data-propagation delays between tasks that are activated with different periods. The proposed technique is based on the read-execute-write model for the execution of tasks to guarantee temporal isolation when accessing the shared resources. A Constraint Programming formulation is presented to find the schedule for each core. Evaluations are preformed to assess the scalability as well as the resulting schedulability ratio, which is still 18% for two cores that are both utilized 90%. Furthermore, an automotive industrial case study is performed to demonstrate the applicability of the proposed technique to industrial systems. The case study also presents a comparative evaluation of the schedules generated by (i) the proposed technique and (ii) the Rubus-ICE industrial tool suite with respect to jitter, inter-core data-propagation delays and their impact on data age of task chains that span multiple cores.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**; **Embedded systems**.

KEYWORDS

Real-time systems, inter-core communication, data propagation delay, scheduling, time triggered.

ACM Reference Format:
Lamija Hasanagić, Tin Vidović, Saad Mubeen, Mahammad Ashjaei, and Matthias Becker. 2021. Optimizing Inter-Core Data-Propagation Delays in Industrial Embedded Systems under Partitioned Scheduling. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3394885.3431515

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431515

1 INTRODUCTION

The requirement for high computational power to support compute-intensive features in embedded system applications has drastically increased in the past few years [17]. To meet this requirement, the developers of these applications have started to employ multi-core architectures as opposed to the contemporary single-core architectures. However, the transition from single-core to multi-core is not trivial, especially in the case of real-time embedded systems, which have stringent timing requirements, e.g., timing constraints on the response times of tasks and data-propagation delays through chains of tasks [3, 5, 16, 21]. In the case of multi-core real-time embedded systems, the interference introduced due to the contention for shared resources such as the system bus and memories can lead to potentially unbounded delays [13, 31, 36]. Schedulability analysis techniques are used to ascertain if the specified timing constraints are met or not. Real-time scheduling and schedulability analysis for single-core real-time embedded systems are well developed and well adopted by the industry [10, 28]. Whereas, these techniques for multi-core real-time systems are still evolving [18].

When transitioning from single-core to multi-core architectures, many industrial systems need to retain legacy functionality, i.e., the transitioned system is required to be backward compatible with the legacy single-core functionality [7]. In this regard, partitioned scheduling [1] is well suited to such systems as it allows to statically allocate the tasks to cores and the tasks are not allowed to migrate among the cores at runtime. This, in turn, supports the reuse of well-established single-core real-time scheduling techniques, thereby enabling the reuse of the previously certified single-core Real Time Operating Systems (RTOSs) on each core. However, the downside of using this approach is that the schedules generated for individual cores are not optimized from the perspective of tasks communicating across core boundaries as the schedulers of the individual cores do not inherently take the inter-core interference into account. For example, this is the case of the ISO26262 [12] certified Rubus RTOS and corresponding tool suite (Rubus-ICE), which have been used in the automotive industry for over 25 years [19, 20]. Specifically, we focus on inter-core data propagation delays where register communication is used between the cores. That is, communicating tasks access a shared variable using the last-is-best semantic [2, 14]. As communicating tasks might execute at different periods and the communication does not require direct signaling between tasks the problem is not trivial.

This paper aims at providing a technique[1] to optimize the core schedules under partitioned scheduling by avoiding shared resource contention by design as well as by minimizing the *inter-core data-propagation delays*[2], which is the latency between reading of a communication variable at the consumer, and the time the variable was last written by the producer task. **The main contributions** in the paper are as follows.

- A scheduling technique based on a Constraint Programming (CP) formulation, structured according to the *Read-Execute-Write (REW) model*, under partitioned scheduling in multi-core architectures, such that the contention on shared memory is avoided by design and tasks are scheduled optimally from the perspective of inter-core data-propagation delays. The technique is usable with multi-rate communication, where the tasks are time-triggered and can have different activation periods.
- Extensive evaluation of the proposed technique from the perspective of assessing the scalability of the technique and the resulting schedulability ratio on a set of realistic synthetic test cases.
- An automotive industrial case study is conducted to perform a comparative evaluation of the schedules generated by (i) the proposed technique and (ii) the Rubus-ICE industrial tool suite. The evaluation parameters considered in the case study include the start time jitter of individual tasks, inter-core data-propagation delays, and age delays of task chains.

2 RELATED WORK

The contention for shared resources in multi-core real-time embedded systems can lead to additional interference, which can incur unbounded delays [8, 33, 34] rendering the traditional schedulability analysis techniques for single-core systems not applicable to these systems [31]. Research in this area has received significant attention in recent years. Maiza et al. [18] provide an exhaustive survey of the state-of-the-art and categorize research into timing verification of multi-core systems into four main areas: *full integration*, *integrating interference effects into schedulability analysis*, *mapping and scheduling* and finally *temporal isolation*, on which we mainly focus in this paper. Temporal isolation provides an upper bound on the delays experienced by a single task, independent of the tasks executing on the other cores [13].

In this paper, we focus on the REW model [4], which allows each task to be split into memory access phases (typically a read and a write phase) and a computational phase. If tasks are structured in such a way, contention for the shared memory can be avoided by simply not overlapping the memory phases of concurrent tasks [25] [22] [4]. The execution semantics of this model are similar to those that are used in the AUTOSAR standard [30] (in the case of implicit communication) in the automotive domain and the Acquisition, Execution, and Restitution (AER) model considered for the avionics domain [11]. This model is shown to produce the best results among several other execution models [25].

Several works focus on the generation of schedules using resource optimization techniques [32] [23] [4] [6]. Tompkins [32] proposes a mixed integer linear programming approach for solving complex scheduling problems in distributed multi-agent systems. Puffitsch et al. [23] investigate ways of enforcing timing predictability in safety-critical multi-core embedded systems and use constraint programming techniques to create offline schedules. Becker et al. in [4] and [6] schedule REW structured tasks on clustered many-core platforms with memory constraints and use constraint programming techniques to generate a contention free schedule as well as the data allocation to memory.

In this paper we employ constraint programming techniques to create offline schedules under partitioned scheduling in multi-core architectures, where the contention for shared resources is avoided by design. The CP approach is shown to scale better than the Integer Linear Programming approach when it comes to industrial sized applications in [24] and [35]. In addition to this, the technique we propose also minimizes inter-core data-propagation delays between tasks of different periods, where the tasks are time-triggered.

3 SYSTEM MODEL

This section presents the system model. First, the platform assumptions are presented, followed by the application model.

3.1 Platform Model

The embedded system application is executed on a multi-core platform with M homogeneous cores. Each core has its own private cache and no shared caches are present. It is important to note that the code of all tasks is assumed to be prefetched into the local cache of the appropriate core, which is assumed to be large enough to hold the code binaries of all tasks assigned to it. This prevents additional memory accesses at runtime. The cores operate synchronously with the system clock, which therefore assures mutual synchronization among the cores. The task execution can be simultaneous for tasks running on different cores. Shared memory is used for communication between tasks and accessed via the system bus in a 32bit granularity. The memory controller uses round robin arbitration to arbitrate between memory accesses of the different cores.

While this architecture is generic, it is in line with the several prominent commercial off-the-shelf (COTS) micro controller architectures in the automotive domain, e.g., MPC5675K [26] and MPC5777C [27] micro-controller architectures.

3.2 Application Model

The application model considers the usage of partitioned scheduling. An application consist of N tasks that are statically allocated to n cores. Tasks use the *register-communication* paradigm [2, 14] where shared variables store the communication data. These communication variables are stored in the shared memory of the platform that is accessed by all cores. Such a communication form allows tasks to be executed independently as there is no direct signaling between them. In order to provide temporal isolation when accessing the shared memory, this paper considers that the tasks are executed according to the REW model [4]. According to this model, a task's execution is split into three distinct phases, *read*, *execute*, and *write*. In the read phase, local copies of the task's communication variables are created, and during the write phase the values of the local variables are written back to the shared memory. Thus, there is no

[1]The source code of the implemented technique is publicly available https://github.com/tvidovic1/optimization_engine

[2]We use the terms "inter-core data-propagation delay" and "inter-core communication delay" interchangeably throughout the paper.

need to access the shared memory in the *execution* phase [22] [4]. Each of these phases is executed *non-preemptively*.

A task τ_i in an application is represented by the tuple $\tau_i = (C_{R_i}, C_{E_i}, C_{W_i}, T_i, p_i)$, where C_{R_i}, C_{E_i} and C_{W_i} are the worst-case execution times (WCETs) of its read, execute and write phases, respectively, T_i is its period, and p_i defines the core that the task is allocated to. The relative deadline D_i of the task is not explicitly defined, but it is considered to be equal to the period. All tasks that are executing on a core m are grouped into a task set Γ_m. The hyperperiod HP of all N tasks is obtained by calculating the least common multiple (LCM) of their periods, i.e. $HP = LCM(T_i | \tau_i \in \{\tau_0, \ldots, \tau_{N-1}\})$.

Furthermore, the j^{th} instance of a task τ_i is denoted by $\tau_{i,j}$. Where $r_{i,j}$ is the absolute release time of the j^{th} instance of task τ_i. These parameters implicate the existence of the following parameters: $s_{(r)i,j}$, $s_{(e)i,j}$ and $s_{(w)i,j}$ which represent the absolute start times of the read, execute and write phases of the task instance respectively, and $e_{(r)i,j}$, $e_{(e)i,j}$ and $e_{(w)i,j}$ which represent the absolute end times, of the read, execute and write phases of the task instance respectively, and $d_{i,j}$ which represents the absolute deadline of the task instance.

Inter-core communication between a task τ_i and a task τ_k is realized by register communication and denoted by Ψ_j^i. Note that τ_i and τ_k may have different periods which leads to over-sampling or under-sampling introducing further complexity to the application model. The set of all inter-core communication is defined as $\mathcal{S}$.

4 OPTIMIZATION TECHNIQUE

We use CP with *interval decision variables* to create the schedules. These variables are tailored for constraint-based scheduling problems and are used to model activities that last for a certain period of time. These type of variables are for example supported by the state-of-the-art CP solver IBM ILOG CP Optimizer [15], which supports interval decision variables alongside some specialized scheduling constraints, to efficiently model and solve scheduling problems.

4.1 Interval Decision Variables

For each of the M task sets Γ_m, the read, execute and write phase of each task instance within the hyperperiod HP are modeled with an interval decision variable and stored in a set $\mathcal{J}_m$. Additionally, helper variables are created to represent the final instance of each task in the previous hyperperiod. As these helper variables represent the same execution as the last instance of each task in the created schedule, their timing parameters are linked by constraints. The reason for considering the final instance of each task in the previous HP is that an instance of a task may communicate with the last instance of the previous HP. The notions of the created interval decision variables are shown in Table 1. In addition, an interval decision variable is defined for the complete task instance, denoted by $\tau_{i,j}^{All}$. This is necessary as no second task is allowed to execute during potential idle time between the phases of $\tau_{i,j}$ as otherwise local memory might be compromised. The start of $\tau_{i,j}^{All}$ can therefore be linked with the start of $\tau_{i,j}^{R}$, the end can be linked with the end of $\tau_{i,j}^{W}$, and the length of $\tau_{i,j}^{All}$ is constrained by the interval $[C_{R_i} + C_{E_i} + C_{W_i}, T_i]$.

Table 1: Interval decision variables for the REW phases.

Phase	Notion	Start	length
Read	$\tau_{i,j}^{R}$	$[r_{i,j}, d_{i,j} - C_{R_i} - C_{E_i} - C_{W_i}]$	C_{R_i}
Execute	$\tau_{i,j}^{E}$	$[r_{i,j} + C_{R_i}, d_{i,j} - C_{E_i} - C_{W_i}]$	C_{E_i}
Write	$\tau_{i,j}^{W}$	$[r_{i,j} + C_{R_i} + C_{E_i}, d_{i,j} - C_{W_i}]$	C_{W_i}

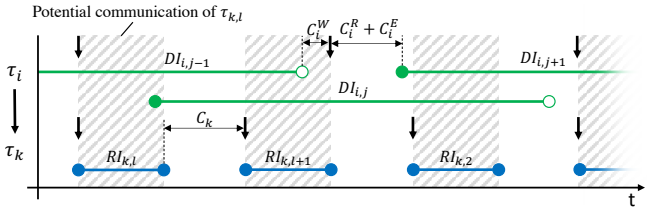

Figure 1: Potential communication between τ_i and τ_j.

4.2 Integer Decision Variables

A set of integer decision variables $\mathcal{I}$ is created that model the inter-core data-propagation delays between *potentially communicating* producer-consumer instance pairs for each inter-core communication $\in \mathcal{S}$. If two task instances $\tau_{i,j}$ and $\tau_{k,l}$, where $\Psi_k^i \in \mathcal{S}$, are able to *potentially* communicate, i.e. there exists such a schedule, an integer decision variable $Z_{k,l}^{i,j}$ is introduced that denotes the potential communication delay between the two instances (i.e. the difference between the start of the read time of $\tau_{k,l}$ and the end of the write time of $\tau_{i,j}$).

In order to determine task instances that potentially communicate, the concept of *data interval* and *read interval* is used [2]. The read interval $RI_{k,l} = [r_{k,l}, d_{k,l} - (C_{Rk} + C_{Ek} + C_{Wk})]$ describes the interval in which $\tau_{k,l}$ may read its input data, while still meeting its timing requirements. Similarly, the data interval $DI_{i,j} = [r_{i,j} + C_{Ri} + C_{Ei}, d_{i,j+1} - C_{Wi})$ describes the interval in which the data produced by $\tau_{i,j}$ may be available to other tasks. This is, from the earliest time $\tau_{i,j}$ may write the communication variables, until $\tau_{i,j+1}$ latest overwrites the data of $\tau_{i,j}$. Consequently, an instance $\tau_{k,l}$ is potentially communicating with any instance of τ_i whose data interval overlaps with its read interval, (i.e. there exists a schedule in which the instances communicate). Fig. 1 exemplifies this. Read and data intervals of several instances of τ_k and τ_i are shown. In addition, the overlap in the read intervals are highlighted using grey background. For example, $RI_{k,l}$ overlaps with $DI_{i,j-1}$ and $DI_{i,j}$. Thus, the variables $Z_{k,l}^{i,j-1}$ and $Z_{k,l}^{i,j}$ are added to $\mathcal{I}$. To ease the later presentation we denote the cardinality of $\mathcal{I}$ by c.

Since the tasks communicate via register communication with no additional signaling between them, in the final schedule a task only consumes the data produced by exactly *one* instance of the producer task. Even though an instance of a consumer task can potentially communicate with multiple instances of a producer task. Thus, a consumer instance $\tau_{k,l}$ is considered to be communicating with a producer instance $\tau_{i,j}$ if, in the final schedule, $\tau_{k,l}$ reads the data from the shared memory that was written there by $\tau_{i,j}$. Therefore, for each potential communication $Z_{k,l}^{i,j}$ an additional

helper integer decision variable $H^{i,j}_{k,l}$ is created to distinguish if the producer-consumer instance pair actually communicates in the final schedule. The set of all such variables is denoted by $\mathcal{H}$ and the total number of created H variables is also equal to c.

4.3 Basic Scheduling Constraints

In order for the generated schedule to be considered valid we introduce some basic scheduling constraints. Firstly, for each task instance we introduce two precedence constraints.

The $\texttt{endBeforeStart}(a, b, t)$ constraint specifies that the interval decision variable a must end at least t time units before the start of the interval decision variable b. This constraint type is used to define a precedence order between the read phase of each instance and its execute phase. That is, a read phase should be scheduled at least 0 time units before its execute phase:

$$\forall m \in [0, 1, ..., M-1] \wedge \forall \tau^R_{i,j} \in \mathcal{J}_m : \\ \texttt{IloEndBeforeStart}(\tau^R_{i,j}, \tau^E_{i,j}, 0) \quad (1)$$

Similarly, the second precedence constraint specifies that the execute phase of each instance should be scheduled at least 0 time units before its write phase:

$$\forall m \in [0, 1, ..., M-1] \wedge \forall \tau^E_{i,j} \in \mathcal{J}_m : \\ \texttt{IloEndBeforeStart}(\tau^E_{i,j}, \tau^W_{i,j}, 0) \quad (2)$$

The operator $\texttt{noOverlap}(a_1, a_2, ..., a_n)$ provides a constraint specifying that no two interval decision variables in the set $\{a_1, a_2, ..., a_n\}$ can overlap. As we consider non-preemptive scheduling, the following constraint specifies that task instances on the same core are not allowed to overlap:

$$\forall m \in [0, 1, ..., M-1] : \texttt{noOverlap}(\tau^{All}_{i,j} \in \mathcal{J}_m) \quad (3)$$

4.4 Memory Access Constraints

As previously mentioned the access to the shared memory is exclusive between all the tasks executing on all cores. To model this, we introduce an additional noOverlap constraint between all the read and write phases of all the task instances:

$$\texttt{noOverlap}(\tau^R_{i,j}, \tau^W_{i,j} | \tau_{i,j} \in \mathcal{J}_m \forall m \in [0, 1, ..., M-1]) \quad (4)$$

4.5 Inter-Core Delay Constraints

The inter-core delay constraints are used to assign values to the integer decision variables that model the communication delay. Each value $Z^{i,j}_{k,l} \in \mathcal{I}$ is constrained to represent the write-read delay of the corresponding, potentially communicating, producer-consumer task instances:

$$\forall Z^{i,j}_{k,l} \in \mathcal{I} : Z^{i,j}_{k,l} == \texttt{IloStartOf}(\tau^R_{k,l}) - \texttt{IloEndOf}(\tau^W_{i,j}) \quad (5)$$

The operator $\texttt{IloIfThen}(a, b)$ introduces the constraint b if the condition a is satisfied. The second set of constraints ensure that $H = Z$, if the corresponding Z variable represents an actually communicating pair, and that $H = 0$ otherwise:

$$\forall H_i : \texttt{IloIfThen}(\texttt{isCommunicatingPair}(Z^{i,j}_{k,l}), H^{i,j}_{k,l} = Z^{i,j}_{k,l})$$

$$\forall H_i : \texttt{IloIfThen}(!\texttt{isCommunicatingPair}(Z^{i,j}_{k,l}), H^{i,j}_{k,l} = 0)$$

where $\texttt{isCommunicatingPair}(Z^{i,j}_{k,l})$ is a function that returns true if the two task instances $\tau_{i,j}$ and $\tau_{k,l}$ represented by $Z^{i,j}_{k,l}$ are actually communicating. The function evaluates all Z variables of the inter-core communication Ψ^i_k that terminate in the instance $\tau_{k,l}$, and returns true, if $Z^{i,j}_{k,l}$ has the smallest positive value among these variables. In this case, a negative value indicates that data is written by a future instance of τ_i, and a positive value that is not the smallest positive value indicates the data is overwritten before $\tau_{k,l}$ reads it.

4.6 Objective Function

Once all the decision variables and constraints have been specified the application is ready to be scheduled. The main objective of the schedule generation is to find the optimal schedule from the perspective of inter-core data-propagation delays. With this in mind we define the objective function $f(x)$ as follows:

$$f(x) = argMin \sum_{\forall H^{i,j}_{k,l} \in \mathcal{H}} H^{i,j}_{k,l}. \quad (6)$$

5 EVALUATION

In this section we evaluate the proposed technique. Synthetic experiments with a large number of task sets are used to evaluate scalability as well as the resulting schedulability ratio. A case study illustrates the impact of our method on jitter, inter-core data-propagation delays and their impact on data age of task chains that span multiple cores.

5.1 Synthetic Experiments

5.1.1 Experiment Setup. Synthetic task sets are created with parameters that conform to automotive applications [14]. The utilization for each task is created using the Randfixedsum algorithm [29] and periods are selected out of the set [2, 5, 10, 20, 50, 100, 200, 1000] ms according to the real-world automotive benchmarks [14]. The size of communication labels is selected between 1 and 64 bytes. The probability distribution for period values as well as the size of communication labels are selected based on the values presented in the benchmarks. Based on the size of communication variables, the read and write times are determined, assuming memory access consumes 10 clock cycles per byte and a clock frequency of 300MHz. These values are consistent with an industrial engine management system and with MPC5777C micro-controller architecture, which is a platform used in the automotive domain. To evaluate the CP problem size, a varying number of cores is considered. For the remaining experiments a platform with 2 cores is used. Finally, tasks are partitioned to cores based on the utilization distribution stated in the respective experiment. A varying number of inter-core communication between tasks on different cores is specified. For each data point, 200 random systems are generated and the average values are presented.

IBM ILOG CP Optimizer is used to solve the CP problem. The bound on the solving time is set to 3 minutes. If no solution is found within this time, the task set is deemed unschedulable.

5.1.2 CP Problem Size. In order to evaluate the impact of the system configuration on the size of the CP problem, two experiments are performed.

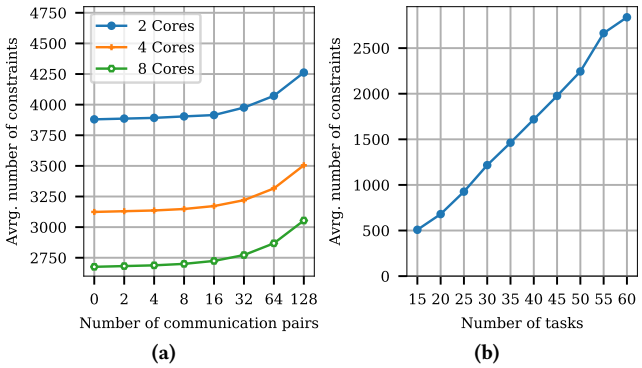

Figure 2: Average number of constraints with respect to: (a) the number of inter-core communicating task pairs, and (b) the number of tasks.

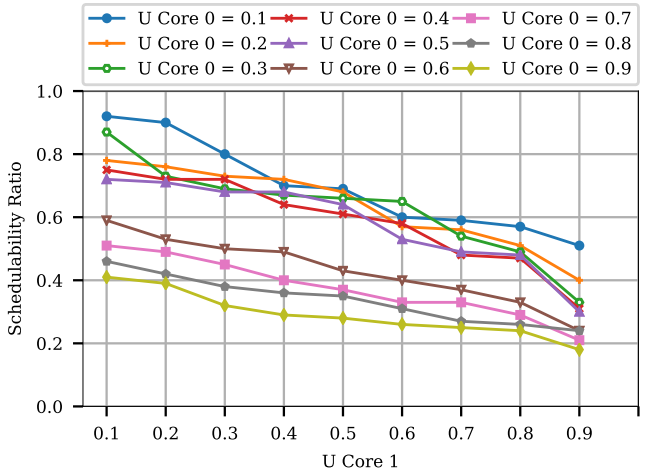

Figure 3: Schedulability ratio with varying utilization on core 0 and core 1.

Fig. 2a depicts the evaluation of the impact that inter-core communication has on the CP problem size. For this experiment, task sets with 60 tasks are created and partitioned to 2, 4, and 8 cores with a resulting core utilization of 50% respectively. The amount of inter-core communication is then varied in the range [0, 2, 4, 8, 16, 32, 64, 128]. The results show that the number of constraints linearly increases with the number of inter-core communications (note the scale of the x-axis). It can also be seen that fewer cores result in a larger number of constraints. This is because the number of tasks in this experiment is static, which leads to fewer tasks to be scheduled on individual cores with increasing core count.

In comparison, the impact of the number of tasks on the CP problem size is shown in Fig. 2b. Here the amount of inter-core communication is kept static at 32 and the number of tasks is varied from 15 to 60 tasks in steps of 5 tasks, while the utilization of each core is set to 50%. It can be seen that the impact of the number of tasks on the CP problem size is also linear. However, adding additional tasks increases the problem size by an average of 51.78 constraints, while adding inter-core communication increases the number of constraints in average by only 2.9 constraints, irrespective of the number of cores. Thus, the proposed CP formulation to optimize the inter-core communication delays is efficient in the number of resulting constraints.

5.1.3 Schedulability Ratio. This experiment evaluates the schedulability ratio (i.e. the ratio of schedulable to unschedulable task sets) of the proposed CP technique for varying utilization of core 0 and core 1. For these experiments, task sets with 30 tasks are created that are subject to 16 inter-core communications. Fig.3 presents the results of the experiments, where the x-axis denotes the utilization of core 1 and for each examined utilization value of core 0 one curve is shown respectively. It can be seen that, with an increase in utilization in any of the two cores, the schedulability ratio decreases, with 20% of all evaluated systems being deemed schedulable at a utilization of 90% on each of the cores while read and write phases of all tasks are non-overlapping and inter-core communication delays are optimized.

5.2 Automotive Application Case Study

To evaluate different performance characteristics, a case study is performed. The examined application is based on the model engine control application presented in [9]. The application is scheduled on a microcontroller platform found in the automotive industry, the MPC5777C [27]. The MCP5777C has 2 CPUs which operate at a clock frequency of 300 MHz. The application's parameters have been adjusted to fit the selected platform. The application consist of 18 tasks whose parameters are given in Table 2. The application also includes 3 multi-rate task chains, Chain A, Chain B, and Chain C. The mapping of tasks to cores results in each chain traversing core boundaries at least once, as shown in Fig. 4. In addition, periods of tasks are visualized using blue (50 ms), green (100 ms), and red (1000 ms) color. This highlights the multi-rate behavior of the different task chains.

For comparison with our CP based method, the application is also scheduled using the industrial tool suite Rubus-ICE and Rubus RTOS that have been used in the automotive industry for over 25 years [19, 20]. In contrast to our technique, Rubus does not consider inter-core communications and each core is managed and scheduled independently by its certified single-core real-time RTOS.

5.2.1 Start Time Jitter. The schedule generation of Rubus-ICE attempts to minimize the start time jitter of each task. Indeed Rubus-ICE finds a schedule that results in each task having no start time jitter, i.e. consecutive instances of each task τ_i are scheduled exactly T_i time units apart. In contrast, our technique does not consider start time jitter during the schedule creation. Thus, jitter is present in the schedule created by our technique. The resulting jitter values are reported in Table 2, with a maximum start time jitter observed for the task `TransFuelMass` with 43.77 ms, 43.77% of the tasks period. The observed jitter in our technique can be considered as a trade-off in the optimization for inter-core data propagation delays and will be further investigated in future work.

5.2.2 Inter-Core Data Propagation Delay. Inter-core communication takes place 6 times, as shown in Fig. 4. Our CP technique successfully minimizes all inter-core communication. Rubus-ICE builds

Table 2: Tasks used in the automotive application case study and the resulting jitter by the two methods.

Task	C_{Ri}[ns]	C_{Ei}[ns]	C_{Wi}[ns]	C_i[ns]	T_i[ms]	p_i	Jitter CP [ms]	Jitter RCM [ms]
CylNumObserver	908	573000	25	573933	1000	0	1.95	0
IgnitionSync	958	2461000	195	2462153	1000	1	8.04	0
MassAirFlow	908	86000	28	86936	50	0	7.48	0
ThrottleSensor	908	169000	55	169963	50	1	1.65	0
APedSensor	908	482000	55	482963	50	0	0.39	0
APedVoter	55	144000	28	144083	100	0	1.81	0
ThrottleCtrl	990	2892000	55	2893045	100	1	3.59	0
ThrottleActuator	1013	2957000	83	2958096	100	0	3.04	0
BaseFuelMass	990	2892000	55	2893045	100	1	42.61	0
ThrottleChange	1013	2957000	83	2958096	100	0	8.06	0
TransFuelMass	1148	3188000	28	3189176	100	1	43.67	0
Ignition	1060	2269000	25	2270085	1000	0	2.53	0
TotalFuelMass	988	677000	28	678016	100	0	3.04	0
OperatingMode	965	19641000	390	19642355	200	1	1.29	0
IdleSpeedCtrl	833	843000	240	844173	200	0	0.31	0
APedSensorDiag	908	118000	0	118908	1000	0	0.09	0
InjBattVoltCorr	28	274000	28	274056	1000	0	0.21	0
Injection	985	1651000	195	1652180	1000	1	0	0

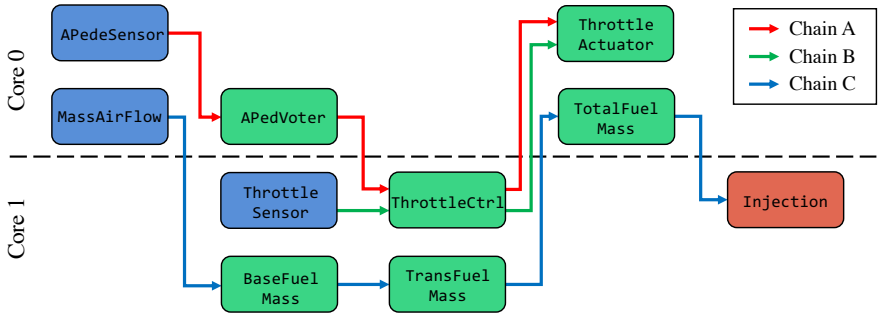

Figure 4: Task chains of the model engine controller and their mapping to cores.

the schedule for each core independently, thus inter-core communication delay is not directly optimized. This is for example visible in the third inter-core communication between `ThrottleCtrl` to `ThrottleActuator` tasks in Chains A and B, that has a maximum communication delay of 91.43 ms. Both tasks have an execution period of 100 ms. As Rubus-ICE is unaware of the inter-core communication, the two tasks are executed in such a way that the receiver is always scheduled just before the sender, thus the data the receiver task reads is written almost a full period earlier. The inter-core communication delay for each communication is shown in Table 3.

Table 3: Maximum inter-core data-propagation delays.

Chain	(Producer, Consumer)	CP[ms]	Rubus-ICE[ms]
Chain A	(APedVoter,ThrottleCtrl)	0	54.86
	(ThrottleCtrl,ThrottleActuator)	0	91.43
Chain B	(ThrottleCtrl,ThrottleActuator)	0	91.43
Chain C	(MassAirFlow,BaseFuelMass)	0	2.79
	(TransFuelMass,TotalFuelMass)	0	47.21
	(TotalFuelMass,Injection)	0	1.38

5.2.3 Impact on Data Age. One of the most important metrics in the automotive industry is the data age of task chains. The data age describes the maximum time for data to propagate through a task chain [10]. That is, how long an input value has effect on the output of the task chain. Table 4 reports the maximum data age for each of the task chains with the proposed CP method as well as with Rubus-ICE. For all the chains, Chain A, Chain B and Chain C, the maximum data age using our technique is improved by 97.04 ms, 91.41 ms and 50.41 ms respectively. However, it is important to note that optimizing inter-core data-propagation delays does not necessarily mean that the data age is minimized as well. For example, in all of the chains there are at least two successive tasks allocated to the same core. Since the proposed approach only minimizes inter-core communication delays, these intra-core communication delays can negatively impact the overall data age of the chain. This illustrates that, while optimizing inter-core delays can have a positive impact on the resulting data age, it is not solely influenced by it.

Table 4: Maximum age delays for the task chains.

Chain	CP[ms]	Rubus-ICE[ms]
Chain A	56.49	153.53
Chain B	9.61	100.4
Chain C	101.65	152.06

6 CONCLUSIONS AND FUTURE WORK

Many industrial domains face the challenge of transitioning to multi-core platforms while reducing the impact on their legacy systems. In this work we focus on partitioned scheduling, where each core of a multi-core platform hosts its own real-time operating system or scheduler. A constraint programming model is developed to generate a time triggered schedule for each of the cores, following the Read-Execute-Write execution model. Memory phases of the cores are scheduled exclusively, which allows each core to be analyzed in isolation. In addition, our technique orchestrates the execution of tasks that communicate across core boundaries in such a way that inter-core data propagation delays are minimized.

The proposed technique is evaluated using a large number of synthetic experiments that demonstrate the scalability as well as the resulting schedulability ratio of the technique. Indeed, 18% of all investigated task sets remained schedulable for a utilization of 90% on each core in our experiments, with 30 tasks and 16 inter-core communication instances. Additionally an industrial case-study is performed that highlights key properties of the proposed method and provides a comparative evaluation of the proposed technique and an industrial tool suite accompanied by a certified real-time operating system that are used in the automotive domain.

Future work will incorporate data propagation delay constraints of task chains, as well as the optimization of jitter together with the optimization of inter-core data propagation delays.

ACKNOWLEDGMENTS

The work in this paper is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the projects PROVIDENT and DESTINE and the Swedish Knowledge Foundation through the projects HERO, FIESTA and DPAC. We thank all of our industrial partners involved in these projects, especially Arcticus Systems, Volvo Construction Equipment, and HIAB Sweden.

REFERENCES

[1] S. Baruah, M. Bertogna, and G. Buttazzo. 2015. *Multiprocessor Scheduling for Real-Time Systems.* Springer International Publishing Switzerland. 103–119 pages.

[2] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte. 2016. 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).* 159–169.

[3] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte. 2017. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture* 80 (2017), 104–113.

[4] M. Becker, D. Dasari, B. Nicolic, B. Åkesson, V. Nélis, and T. Nolte. 2016. Contention-Free Execution of Automotive Applications on a Clustered Many-Core Platform. In *28th Euromicro Conference on Real-Time Systems (ECRTS).*

[5] M. Becker, S. Mubeen, D. Dasari, M. Behnam, and T. Nolte. 2017. A generic framework facilitating early analysis of data propagation delays in multi-rate systems (Invited paper). In *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA).* 1–11.

[6] M. Becker, S. Mubeen, D. Dasari, M. Behnam, and T. Nolte. 2018. Scheduling multi-rate real-time applications on clustered many-core architectures with memory constraints. In *23rd Asia and South Pacific Design Automation Conference (ASP-DAC).*

[7] A. Bucaioni, S. Mubeen, F. Ciccozzi, A. Cicchetti, and M. Sjödin. 2017. Technology-Preserving Transition from Single-Core to Multi-core in Modelling Vehicular Systems. In *Modelling Foundations and Applications.* Springer, Cham.

[8] D. Dasari, B. Akesson, V. Nélis, M. A. Awan, and S. M. Petters. 2013. Identifying the sources of unpredictability in COTS-based multicore systems. In *8th IEEE International Symposium on Industrial Embedded Systems (SIES).*

[9] P. Dziurzanski, A. Singh, L. Indrusiak, and B. Saballus. 2016. Benchmarking, System Design and Case-studies for Multi-core based Embedded Automotive Systems. In *2nd International Workshop on Dynamic Resource Allocation and Management in Embedded, High Performance and Cloud Computing DREAMCloud.*

[10] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. 2008. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS).*

[11] Sylvain Girbal, Guy Durrieu, Madeleine Faugere, Daniel Gracia Pérez, Claire Pagetti, and Wolfgang Puffitsch. 2014. Predictable Flight Management System Implementation on a Multicore Processor. In *Embedded Real Time Software (ERTS).*

[12] International Organization for Standardization (ISO). 2011. ISO 26262: Road Vehicles - Functional Safety.

[13] J. Zhang, Z. Gu, M. Zhang. 2016. Reducing the Upper Bound Delay by Optimizing Bank-to-Core Mapping. *Journal of Computer Science and Technology* 31 (2016).

[14] S. Kramer, D. Ziegenbein, and A. Hamann. 2015. Real world automotive benchmarks for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS).*

[15] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. 2018. IBM ILOG CP Optimizer for Scheduling. *Constraints* 23, 2 (2018), 210–250.

[16] P. A. Laplante. 2004. *Real-Time System Design and Analysis.* John Wiley Sons, Inc., Hoboken, NJ, USA.

[17] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara. 2019. Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems. *IEEE Transactions on Industrial Informatics* 15, 2 (2019), 1038–1051.

[18] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, and R. I. Davis. 2019. A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. *ACM Comput. Surv.* 52, 3 (2019), 56:1–56:38.

[19] S. Mubeen. 2019. Developing Predictable Embedded Systems in the Vehicle Industry: Results and Lessons Learned. In *IEEE International Conference on Industrial Technology (ICIT).* 1063–1065.

[20] S. Mubeen, H. Lawson, J. Lundbäck, M. Gålnander, and K. Lundbäck. 2017. Provisioning of Predictable Embedded Software in the Vehicle Industry: The Rubus Approach. In *4th IEEE/ACM International Workshop on Software Engineering Research and Industrial Practice.*

[21] S. Mubeen, J. Mäki-Turja, and M. Sjödin. 2013. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems* 10 (2013), 453–482.

[22] R. Pellizzoni, E. Betti, S. Bak, Y. Gang, J. Criswell, M. Caccamo, and R. Kegley. 2011. A Predictable Execution Model for COTS-Based Embedded Systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* 269–279.

[23] W. Puffitsch, E. Noulard, and C. Pagetti. 2015. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Systems* 51 (July 2015).

[24] E. Noulard C. Pagetti P. Sainrat Q. Perret, P. Maurere and B. Triquet. 2016. Mapping Hard Real-Time Applications on Many-Core Processors. In *Proceedings of the 24th International Conference on Real-Time Networks ans Systems (RTNS).* 235–244.

[25] Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. 2010. Timing Analysis for TDMA Arbitration in Resource Sharing Systems. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* 215–224.

[26] NXP Semiconductors. 2020. MPC5675K Reference Manual. https://www.nxp.com/docs/en/data-sheet/MPC5675K.pdf Accessed on 7th July 2020.

[27] NXP Semiconductors. 2020. MPC5777C Reference Manual. https://www.nxp.com/docs/en/data-sheet/MPC5777C.pdf Accessed on 7th July 2020.

[28] L. Sha, T. Abdelzaher, K.-E. Å rzén, A. Cervin, T. P. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. P. Lehoczky, and A. K. Mok. 2004. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems* 28, 2/3 (2004), 101–155.

[29] R. Stafford. 2006. Random Vectors with Fixed Sum. MATLAB Central File Exchange, available at (https://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum), accessed on April 20, 2020.

[30] The AUTOSAR Consortium. 2016. AUTOSAR Technical Overview. In *Version 4.3.* http://autosar.org.

[31] L. Thiele and R. Wilhelm. 2004. Design for Timing Predictability. *Real-Time Systems* 28, 2 (2004), 157–177.

[32] M. F. Tompkins. 2003. *Optimization Techniques for Task Allocation and Scheduling in Distributed Multi-Agent Operations.* Technical Report. Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science.

[33] P. Kumar Valsan, H. Yun, and F. Farshchi. 2016. Taming Non-Blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* 1–12.

[34] P. K. Valsan, H. Yun, and F. Farshchi. 2017. Addressing isolation challenges of non-blocking caches for multicore real-time systems. *Real-Time Sys.* 53 (2017).

[35] E. Noulard W. Puffitsch and C. Pagetti. 2015. Off-line mapping of multirate dependent task sets to many-core platforms. *Real-Time Systems* 51, 5 (2015), 526–565.

[36] R. Wilhelm and J. Reineke. 2012. Embedded systems: Many cores — Many problems. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES).* 176–180.