Parallel Execution of AUTOSAR Legacy Applications on Multicore ECUs with Timed Implicit Communication

Sebastian Kehr[†], Eduardo Quiñones[‡], Bert Böddeker[†], Günter Schäfer[§]
[†]DENSO AUTOMOTIVE Deutschland GmbH, [‡]Barcelona Supercomputing Center (BSC),
[§]Telematics/Computer Networks Group, Ilmenau University of Technology
{s.kehr, b.boeddeker}@denso-auto.de, eduardo.quinones@bsc.es, guenter.schaefer@tu-ilmenau.de

ABSTRACT

Parallelization of AUTOSAR legacy applications is a fundamental step to exploit the performance of *multi-core ECUs* (MCEs). However, the migration of an application from a single-core ECU (SCE) to a MCE presents two challenges: first, the extraction of parallelism from an application (composed of tasks) is not always possible due to communication among tasks. Second, reproducing the same data-flow on all target MCEs is required to guarantee the same (predictable) functional behaviour without exhaustive validation and testing efforts. This paper introduces *timed implicit communication* (TIC) for decoupling task communication to allow parallel execution of producer and consumer, while the same data-flow is achieved on all MCEs. Therefore, AUTOSAR implicit communication is applied at task-level and *extended by defined communication times*, which are derived from the original SCE configuration. This is realized by storing produced data in a buffer with a publication timestamp attached. TIC is implemented at AUTOSAR RTE level and does not require modification of source code.

1. INTRODUCTION

Multi-core electronic control unit (ECU)s (MCEs) are seen as the hardware platform for current and future control applications in passenger cars [7]. One advantage of a MCE is the surplus of computational power, which allows to efficiently *exploit thread-level parallelism* of applications. This makes it possible to either execute more complex algorithms or execute the same application on multiple cores with a reduced clock rate (such that deadlines are still kept) to save energy. Unfortunately, existing automotive legacy software has been developed and optimized for the execution on single-core ECUs (SCEs). To take full advantage of multi-core platforms the migration of existing legacy applications must be supported. This is a fundamental step for the adaptation of MCEs in the automotive domain.

An AUTomotive Open System ARchitecture (AUTOSAR) application is composed of a set of *runnables* (i.e. elementary code pieces) that communicate with each other, and AUTOSAR tasks[1], which agglomerate runnables with the same release period and are the unit of scheduling of the AUTOSAR run-time environment (RTE). The execution order of runnables is constraint by two kinds of precedences [4]: (1) *simple precedence* and (2) *extended precedence*, which result from the communication among runnables with the same or different release period, respectively. As a result, simple precedences only exist within tasks, while extended precedences only exist among tasks.

Figure 1 shows the tasks and extended precedences of a common automotive application, an engine management system (EMS). Typically, extended precedences occur between all tasks of an automotive application. This causes poor performance on multi-cores, because of sequential execution of tasks. Removing or changing the extended precedences results in an unpredictable data flow, because it is not guaranteed anymore that input data are completely produced when the receiver starts execution. Therefore, a fundamental requirement for the migration of a legacy application from a SCE to a MCE is the *definition of a predictable and reproducible data flow*, i.e. the order in which tasks communicate. Additionally, this eases the validation and testing on the new platform.

In this paper, we propose a new communication mechanism called *timed implicit communication (TIC)* that decouples the execution of producer and consumer task, which allows them to execute in parallel. Additionally, an identical data flow is guaranteed for all MCEs. TIC works as follows: the producer task stores data in a buffer and attaches a publication timestamp, which is the end of the current period. Afterwards, the consumer task reads from the previous producer instance as compared to the SCE execution by selecting a value with the appropriate timestamp from the buffer. This guarantees that data is read from the same task instance on any MCE. Thus, the data flow remains the same on all target platforms. Moreover, tasks can be executed in parallel.

TIC is compliant to AUTOSAR, i.e. usage in a standard conform application is possible. Its integration in the AUTOSAR RTE requires no changes of application source code. The runnable-to-task mapping remains unchanged, which guarantees a correct data flow inside a task. Hence, TIC is *applicable to legacy applications*. We evaluate TIC with a real automotive legacy application: a diesel EMS. A worst-case execution time (WCET) speed-up of 2.7 and 4.5 is observed with TIC on a 4-core or 8-core MCE, respect-

[1]Task refers to an AUTOSAR task.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
DAC '15 June 07 - 11, 2015, San Francisco, CA, USA
Copyright 2015 ACM 978-1-4503-3520-1/15/06 ...$15.00
http://dx.doi.org/10.1145/2744769.2744889.

ively.

The remainder of this article is structured as follows: Section 2 describes the background to this work and explaines used notations. The problem and objectives are described in section 3. Timed implicit communication is explained in section 4 and evaluated in section 5. Finally, the conclusion is presented in section 6.

2. BACKGROUND AND RELATED WORK

This section describes related work, fundamental properties of automotive legacy applications, and defines used notations.

2.1 Parallelization Approaches

Kanehagi et al. [5] describe the automatic parallelising compiler OSCAR and its usage with an EMS. The program is decomposed into blocks and the code is analysed for data dependencies. A method called *Earliest Executable Condition* analysis is applied to extract parallelism. Programs must follow a set of development guidelines close to MISRA C (called *Parallelizable C*). The process is applied to individual tasks.

A similar approach is presented by Cordes et al. [3]. In contrast to OSCAR tasks are decomposed in a hierarchical task graph, i.e. an intermediate program representation used for program optimization and code generation. In a subsequent step the hierarchical task graph is parallelized with an integer linear programming solver. This approach can only be applied to individual tasks.

2.2 Communication

The communication within an AUTOSAR application is described by a virtual function bus (VFB) [1] and software-component (SW-C) model. Two communication mechanisms for the exchange of a single data element between runnables are defined[2]: *inter-runnable-variable (IRV)* and *sender-receiver (SR)* communication, used between runnables from the same or from different SW-Cs, respectively. The RTE guarantees data consistency for both mechanisms. The developer can define two modes [3]: by default, communication is *explicit*, i.e. a precedence is imposed from producer to consumer, defining a strict order of execution. The consumer uses the most recent value of the producer. Optionally, communication can be defined as *implicit*, in which data is distributed to all consumers after the producer execution has finished. On the consumer side data is buffered and calculations are performed on a copy. As a result, concurrent execution of runnables is possible, because data are buffered and delivered with a delay. This is a form of *asynchronous communication.*

2.3 Case study: Diesel EMS

Automotive control software has strict real-time constraints and the order in which output is produced matters. This mainly distinguishes the parallelization of such software from others (e.g. high performance computing). We consider a diesel EMS, as representative application, to motivate our approach. The examined EMS comprises ca. 1.200 runnables that implement the behaviour of numerous SW-Cs, and exchange data via SR and IRV communication. The internal state of the SW-Cs is updated at different rates, e.g. sensor values are polled with a greater or equal frequency than they are processed. Therefore, runnables with the same released period are mapped to the same task. The application contains 12 independent periodic tasks with different release periods in total.

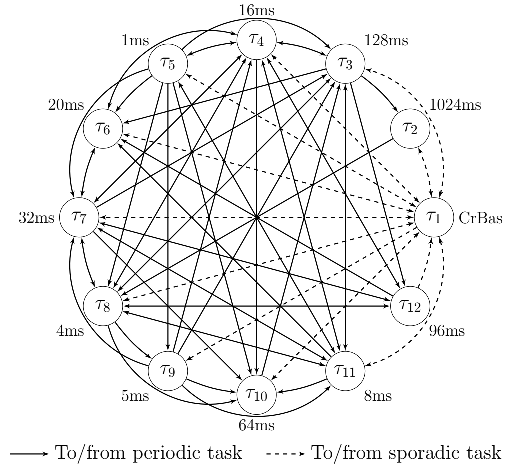


Figure 1: Communication in a diesel EMS.

Figure 1 provides a simplified description of the task set in the diesel EMS. τ_1 executes after an interrupt from the camshaft sensor (*crank-angle task*). The tasks τ_2 to τ_{12} execute with the period denoted by the label close to the node, e.g. task τ_5 has a period of 1ms. An arrow represents communication between the tasks, which is imposed by the runnables mapped to this task. Thus, communication takes place with different frequencies, but with a repetitive pattern that defines the *extended precedences.* The large amount of communication makes the EMS an ideal use case.

The AUTOSAR Operating System [1] supports *periodic* and *sporadic* tasks, as defined in the EMS. Deadlines are defined implicit, i.e. the release of the next task instance. Typically a fixed priority-preemptive scheduling with rate monotonic priority assignment [6] is used in the SCE.

2.4 Notations

This paper uses the notation presented in [4] to express the extended precedences. A system $\mathcal{S}$ consists of a set of tasks, where each task τ_i has a set of real-time attributes (T_i, C_i, O_i, D_i). τ_i is instantiated periodically with period T_i; τ_i^p denotes the p-th iteration of task τ_i. C_i is the WCET of the task. O_i is the release time of the first instance of the task (i.e. the offset with respect to the start time of the system). $o_i^p = O_i + pT_i$ is the release time of τ_i^p. D_i is the relative deadline of the task. $d_i^p = o_i^p + D_i$ is the absolute deadline of τ_i^p. Let s_i^p be the start time and let f_i^p be the finish time of τ_i^p, the following relationship is accomplished $o_i^p \leq s_i^p < f_i^p \leq d_i^p$.

2.4.1 Extended Precedence

A simple precedence is represented by $\tau_i \to \tau_j$. In a periodic system as consider here, an extended precedence between two tasks τ_i and τ_j corresponds to a set of precedences between the instances of the tasks.

[2]Additionally, it is possible to use per-instance memory (PIM) for direct memory access. But, the RTE does not guarantee consistency for PIM. Therefore, it is not recommended and consequently not considered.

[3]Storing data in a queue is another possibility. But this is rarely used and it is therefore not considered.

Definition 1. Extended Precedence: For any $k \in \mathbb{N}$, let $\mathcal{I}_k = [0, k]$ and let $\mathrm{lcm}(a, b)$ be the least common multiple of a and b. Let $\tau_i^n \to \tau_j^{n'}$ denote a precedence from the instance n of τ_i to the instance n' of τ_j. Let $p_{i,j} = lcm(T_i, T_j)$. The extended precedences are:

$$M_{i,j} \subseteq \{(n, n') \mid \tau_i^n \to \tau_j^{n'}, (n, n') \in \mathcal{I}_{p_{i,j}/T_i} \times \mathcal{I}_{p_{i,j}/T_j}\} \quad (1)$$

Extended precedences appear in a repetitive pattern, which is defined as follows.

Definition 2. Periodic Extended Precedence: The periodic extended precedence $M'_{i,j}$ are imposed by the extended precedences $M_{i,j}$ such that:

$$M'_{i,j} = \left\{ (n, n') \middle| \begin{array}{l} \exists k \in \mathbb{N}, (m, m') \in M_{i,j}, \\ (n, n') = (m, m') + (k\frac{p_{i,j}}{T_i}, k\frac{p_{i,j}}{T_j}) \end{array} \right\} \quad (2)$$

In other words, the extended precedences express a subset of all possible communication patterns between instances of τ_i and τ_j. Figure 2 illustrates this with an example for $\tau_5 \xrightarrow{\{(0,0)\}} \tau_8$ of the EMS in fig. 1. This represents the precedences $\tau_5^0 \to \tau_8^0$, $\tau_5^4 \to \tau_8^1$, etc.

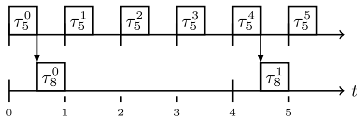


Figure 2: Example for periodic extended precedence.

2.4.2 Data Flow

Sensor data traverse different task instances until an output is produced. Thus, we define the data flow as a path through task instances.

Definition 3. Data Flow Path: Let $M'_{i,j}$ be periodic extended precedences. For any $k \in \mathbb{N}$, we define a *data flow path* $\tau_i^n \rightsquigarrow \tau_j^{n'}$ as a sequence of task instances $\tau_{l_0}^{n_0} \tau_{l_1}^{n_1} \ldots \tau_{l_k}^{n_k}$ such that $\forall q \in [0, k]$:

$$(n^{q-1}, n^q) \in M'_{l_{q-1}, l_q} \wedge \tau_{l_0}^{n_0} = \tau_i^n \wedge \tau_{l_k}^{n_k} = \tau_j^{n'} \quad (3)$$

Regardless of which data flow paths an application contains, they have to be identical in all MCEs.

3. PROBLEM DESCRIPTION

This section motivates our approach. Therefore, we distinguish two scenarios: (1) communication between sporadic and periodic tasks and (2) communication between tasks with different release periods.

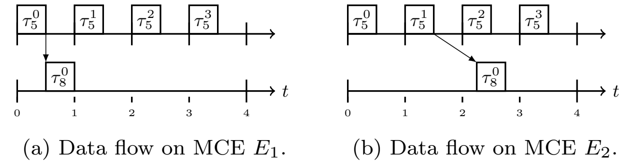
3.1 Sporadic and Periodic Tasks

The time when a sporadic and a periodic task communicate is generally unknown, as it depends on an external unforeseen event (e.g. the camshaft position in case of τ_1). The tasks are executed independent from each other already in the SCE and this is known to produce correct output. The sporadic task must execute immediately after the event and the periodic task executes with fix period to miss no deadline (the task does not wait for input from a sensor). Hence, no strict order of execution (precedence) can be determined, i.e. the data flow is in general neither predictable nor reproducible. Consequently, the asynchronous nature of AUTOSAR implicit communication can be used to decouple the tasks on the MCE and allow parallel execution.

3.2 Different Release Time of Periodic Tasks

The execution order of periodic tasks is defined by the extended precedences imposed from explicit and implicit communication (definition 3). For the task set in fig. 1 it is easy to see that the extended precedences cause sequential execution of all periodic tasks. A high number of extended precedences is not a particular characteristic of this use case, but rather a typical property of automotive control software. That means performance improvements of MCEs cannot be exploited, as long as such a high number of precedences must be respected by a scheduling policy.

Always using implicit communication at task level allows parallel execution due to its asynchronous nature. However, in this case the data flow depends on the scheduling on the target MCE; concretely, the actual start of a task. We illustrate this with an example. A system can have different valid sets of precedences. We schedule τ_5 and τ_8 from fig. 1 on the MCEs E_1 and E_2. On the former one is $s_8^0 = 0.5$ and we have $\tau_5^0 \rightsquigarrow \tau_8^0$. On E_2 is $s_8^0 = 2.25$ and we have $\tau_5^1 \rightsquigarrow \tau_8^0$. Both scheduling policies place τ_8^0 before its deadline, but the data flow is different on both platforms. The consequence is,



(a) Data flow on MCE E_1. (b) Data flow on MCE E_2.

Figure 3: Example for scheduling on MCEs E_1 and E_2 with implicit communication. The data flow is not reproducible.

each implementation for a MCE produces a different output. That is, the data flow paths in E_1 and E_2 are not identical as they are supposed to be (section 2.4.2). Guaranteeing the same output independent from the actual start of a task is essential to define a reproducible and predictable data flow in the MCE. The advantage of defining the same data flow for all target MCEs is the new system becomes reproducible and can be tested easier.

3.3 Objectives

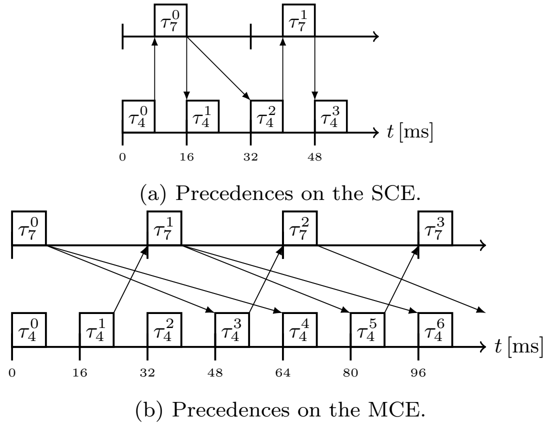
We propose to decouple the data flow from the extended precedences, such that tasks can be scheduled freely within their period. The data flow shall be reproducible for any MCE and shall be based on the execution on the SCE, because this configuration defines a correct functioning system. A communication mechanism that provides a reproducible and predictable data flow must fulfil the following requirements:

1. **Identify producer and consumer:** The mechanism must define the producer and consumer task instance.
2. **Data transport:** The mechanism shall transmit data from the producer to the consumer instance.
3. **AUTOSAR compliance:** It must be possible to integrate the mechanism in an AUTOSAR application.
4. **No change of runnables:** The communication mechanism shall be applicable without modifications of a runnables source code.

4. TIMED IMPLICIT COMMUNICATION

This section presents the main contribution of this paper: a new implementation of AUTOSAR implicit communication that guarantees a predictable and reproducible data flow on any MCE. It must be assumed that the developers intention with a particular SCE configuration was the definition of a correct functioning system. *Therefore, the extended precedences of the SCE provide a suitable basis to define the data flow on the MCE.* Let τ_i^n and $\tau_j^{n'}$ be periodic tasks with the same release time (but different release periods) and $\tau_i^n \rightarrow \tau_j^{n'}$. We propose that the task τ_i^n does not publish produced data until the end of its current period: $O_i + (n+1)T_i = d_i^n$. As a result, the receiver task $\tau_j^{n'}$ cannot read data produced by τ_i^n before d_i^n. But, $\tau_j^{n'}$ can read data from the previous instance of τ_i^{n-1} and therefore execute in parallel to τ_i^n. The motivation for using task periods is that this characteristic is the same for all platforms. Thus, the data flow paths are identical on all MCEs (section 2.4.2). Reading data from a previous instance should not cause harm due to the robustness of the software.

Figure 4 illustrates the idea of TIC for $\tau_4 \leftrightarrow \tau_7$ (fig. 1). We consider a SCE specification with $O_4 = O_7 = 0$, $M_{4,7} = \{(0,0)\}$, and $M_{7,4} = \{(0,1),(0,2)\}$. First, we con-



(a) Precedences on the SCE.

(b) Precedences on the MCE.

Figure 4: The data flow between τ_4 and τ_7 on the SCE (fig. 4a) and with TIC on the MCE (fig. 4b).

sider $\tau_4 \xrightarrow{(0,0)} \tau_7$ on the SCE (fig. 4a). When both tasks have the same release time, τ_4 executes first and produces input data for τ_7. In contrast, on the MCE with TIC τ_4 publishes data every 16ms. τ_7^1 has a release time $o_7^1 = 32$ and hence, data cannot be read from τ_4^2 earlier than $t = 48$. Instead, τ_7^1 reads the data from the previous instance: $\tau_4^1 \rightsquigarrow \tau_7^1$. As a result, τ_7^1 and τ_4^2 can execute in parallel. The data flows $M_{7,4} = \{(0,1),(0,2)\}$ for the MCE are defined accordingly.

The fact that τ_4^0 and τ_4^2 do not communicate with τ_7, does not mean the execution is useless. Each task maintains an internal state, which is progressively calculated such that each task instance communicates with the subsequent instance. This may include an (exclusive) query of sensor data. Moreover, both instances communicate with τ_6 ($\tau_4^0 \rightarrow \{\tau_6^1, \tau_6^3\}$), which is not shown in the figure. τ_7^0 consumes a default value instead.

In order to determine a data flow path on the MCE (definition 3) in more general terms, it is sufficient to specify the extended precedences (definition 1). Therefore, the precedences from the SCE are transformed by shifting the reception of data by one producer period.

Definition 4. Transformed Extended Precedences: Let $M_{i,j}$ be the extended precedences from the SCE (definition 1). The transformed extended precedences on the MCE $M_{i,j}^{\mathcal{MC}}$ are defined as:

$$M_{i,j}^{\mathcal{MC}} = \left\{ (n^*, n') \middle| \begin{array}{l} \exists n : (n,n') \in M_{i,j} \land \\ n^* = \max(n \in \mathcal{I}_{p_{i,j}/T_i} | d_i^n \leq o_j^{n'}) \end{array} \right\} \quad (4)$$

The transformed extended precedences have the same *hyperperiod* (the least common multiple of all task periods) as the precedences on the SCE. Hence, the periodic extended precedences $M_{i,j}^{\prime\mathcal{MC}}$ can be derived analogous to definition 2. That means, the consumer reads from the latest producer period that ends before the release of the consumer. At application start no data are produced, such that a task cannot read from the buffer. A default value is consumed instead.

4.1 Migration with TIC

In order to apply TIC to the use case in fig. 1, we make the following assumptions: (1) precedences between tasks can be described as a repetitive pattern according to definition 2. (2) During execution on the SCE input data of a task remain unchanged. (3) The kind of communication (explicit or implicit) is known. It is out of the scope of this paper to fulfil these assumptions and they are therefore not discussed further. The migration is performed in three steps.

4.1.1 Multi-core Scheduling

At first, a feasible schedule for the MCE is defined, i.e. $\forall \tau_i^p \in \mathcal{S} : o_i^p \leq s_i^p \land f_i^p \leq d_i^p$. For the scheduling it is assumed that task instances are independent, i.e. $M'_{i,j} = \varnothing$. This is possible, because the data flow between task instances is determined in a subsequent step and is guaranteed by TIC. Possible scheduling policies for the MCE are [8, 11].

4.1.2 Replacing Communication

Implicit communication between a sporadic task and a periodic task does not require any change, because it is generally unknown when the communication takes place (section 3.1). The purpose of explicit communication is to guarantee data consistency (prevent data corruption by another task). Thus, explicit communication is replaced by implicit communication to decouple the tasks. Changing the communication mode from explicit to implicit works well in the aforementioned case. However, this approach is not sufficient when both tasks are periodic (section 3.2). Hence, implicit and explicit communication *between periodic tasks* is replaced by TIC. A *publication timestamps* is attached to produced data. The appropriate read date is assigned to the receiver such that from the previous producer instance is read. In the following we explain how the data flow for TIC is determined.

4.1.3 Defining Timestamps

Storing produced data in a buffer is necessary to avoid overwriting. For all $(n,n') \in M_{i,j}^{\prime\mathcal{MC}}$ a publication time d_i^n for τ_i^n and a read time $o_j^{n'}$ is used for $\tau_j^{n'}$. This guarantees that tasks always consume input data at the beginning of a new period. The size of the buffer is limited (the details are discussed in the next subsection).

4.2 Buffer Size

A task produces one new element per period. Let b_i^n denote the time until data from τ_i^n must be available in the buffer. Let $\text{rcv}(\tau_i^n) = \{\tau_j \,|\, (n, n') \in \tau_i \xrightarrow{M_{i,j}} \tau_j\}$ be the receiver tasks of τ_i^n. An element must remain available in the buffer until the last receiver task period has finished: $b_i^n = \max_{\tau_j^{n'} \in \text{rcv}(\tau_i^n)} (d_j^{n'})$. Afterwards, the *out-dated* data can be removed or overwritten. It can be distinguished between the following two cases:

$T_{\text{producer}} < T_{\text{consumer}}$.

In the worst-case, the buffer stores elements from subsequent invocations until data for all receiver tasks has been produced. A new element is stored, if at least one receiver finishes. Then, an out-dated element is removed or overwritten. But, an additional element is required, because the new element is produced in parallel to the consumer task execution. Thus, the size of the buffer is limited to $|rcv(\tau_i^n)| + 1$.

$T_{\text{producer}} > T_{\text{consumer}}$.

Two buffer elements are required to hold the values of the previous and the current task instance. A third buffer element is required to allow parallel execution of producer and consumer task, when the execution overlaps. Thus, the buffer size is 3.

4.3 Implementation

We propose the integration of TIC in the AUTOSAR RTE to avoid changes in the application source code. No modifications of the application are required, because TIC maintains the data flow. In this case, both communication modes, i.e. implicit and explicit, behave in the same way and implement the TIC functionality. A mechanism is required to store data elements with timestamps. A buffer data structure with the following properties satisfies the needs: First, the read and write access is never blocked (wait-free). Second, the write operation $write(x, v, p)$ stores the value v for the variable x with the publication timestamp p. Third, the read operation $read(x, r)$ returns the value of the variable x with publication timestamp r. The actual buffer size and access time may vary depending on the implementation and it is no subject of this paper to optimize these parameters. Instead, we consider different efficient buffer implementations in the evaluation in section 5.

5. EVALUATION

We apply TIC to the diesel EMS presented in section 2 to evaluate our approach. The EMS contains client-server (CS) communication. A server-runnables typically maintains an internal state, which is changed during invocation. Thus, memory coherency must be guaranteed. This is out of the scope of our approach and in order to still perform performance evaluation we propose to enclose calls to a server in *ticket-locks* [10]. This blocks other runnables until the execution of the server-runnable has finished. These locks can be integrated within the RTE and are transparent to the client and the server. Thus, no changes of the application are required and data consistency as well as data coherency are guaranteed.

5.1 WCET Analysis

For automotive applications the behaviour under worst-case conditions is of high importance and the WCET of tasks is therefore considered. In order to perform a quantitative evaluation, the WCET of a task is determined by static analysis with the tool OTAWA [2]. A model of the target processor, the source code, and the compiled executable are used to calculate a trustworthy upper bound for the WCET. The MCE environment is represented by defining a maximum delay for a request to the processors communication subsystem and memory resources [9]. Hence, the WCET estimates of tasks are time-composable. That means the timing behaviour is independent of simultaneously executed tasks, insensitive to the core allocation, and independent of sharing the cache state with other tasks.

5.2 Processor Model

As target platform an analysable multi-core processor like [12] is considered. The maximum time a request of shared resources can take is bound by an upper bound delay (UBD). Every core has a private instruction scratchpad, a write-through data cache, a connection to an on-chip SDRAM memory device through a tree network-on-chip (NoC), and is assumed to exhibit no timing anomalies [14]. The tree is a wormhole-based topology with three pipelined 2-to-1 routers. A message from a core needs 2 hops to reach the on-chip memory [13]. The NoC and the on-chip memory are sources of interference. Given the tree traversal time L_t, the memory latency L_m, and the number of cores c the UBD can be calculated as $\text{UBD} = L_t + (c - 1) \cdot L_m$. For each router we consider a round-robin arbitration policy, which makes L_t independent from the number of cores. A memory request is at the most stalled by all other cores.

5.3 Scheduling

We consider a nonpreemptive scheduling scheme on the MCE. For the estimation of the speed-up the *worst-case scenario* is scheduled, i.e. the situation when all tasks have the same release time. Placing the crank-angle task in a separate core guarantees that interrupts can always be handled with low latency. The periodic tasks are scheduled with the allocation algorithm presented in [11]. It is based on a variant of the worst-fit decrease heuristic, which prioritizes tasks with higher combined utilization.

5.4 Experimental Setup

Parameter	S_1	S_2	S_3
Cores	2	4	8
Data cache size	256 KB		
Cache latency	1 c		
L_t	1 c	2 c	3 c
L_m	10 c	10 c	10 c
UBD	11 c	32 c	73 c
C_{lock}	211 c	337 c	538 c
o_B	0, 50, 100, 150, 200, 250, 300, 350 c		

Table 1: Experiment parameters. c stands for clock cycles.

We consider three different processor setups: 2, 4, and 8 cores. The values for the experimental setup of the different considered processor architectures are summarized in table 1. The parameter o_B is the worst-case overhead for a single buffer operation. o_B is implementation specific and we consider different values. The **speed-up** ($s = t_{\text{seq}}/t_{\text{par}}$) is used as performance metric. The speed-up compares the makespan without TIC (sequential execution t_{seq}) and with

TIC (parallel execution t_{par}, including overhead for buffer accesses and ticket locks section 5.1), when all tasks have the same release time; and so the CPU utilization is maximized. Hence, the speed-up reflects the CPU utilization reduction achieved by TIC.

5.5 Results

Figure 5 shows the speed-up of the EMS on different MCEs. The S_3 provides the highest speed-up of 4.5 for

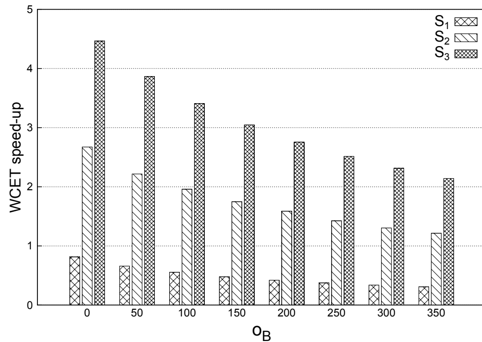


Figure 5: The speed-up achieved with TIC.

$o_B = 0$ cycles. The values for the speed-up decrease down to 2.1 as o_B increases to 350 cycles. For S_2 a maximum speed-up of 2.7 is achieved for $o_B = 0$ cycles. The speed-up decreases down to 1.2 as o_B increases to 350 cycles. The speed-up for S_1 is smaller than 1 for all values of o_B. The reason is, the core 1 only executes τ_1 and all other tasks execute sequentially on core 2. Additionally, buffer operations and locks introduce overhead. Summarizing, the highest speed-up is achieved with the S_3 (4.5). The highest efficiency is achieved with S_2 (0.675). For this reason, the S_2 is the preferable target platform for the parallelized application. The S_1 is no suitable target platform, because of a speed-down.

6. CONCLUSION

This paper presents and evaluates a novel communication mechanism called *timed implicit communication* for the execution of automotive legacy applications on multi-core ECUs. An identical data flow is ensured for all target platforms. Predictability and reproducibility are guaranteed. Our approach is compliant to AUTOSAR, allows current legacy applications to execute tasks in parallel without any modification at source code level, and is independent from the workload of the application. Moreover, the runnable-to-task mapping remains unchanged, which guarantees a correct data flow inside a task. Producer and consumer task are decoupled, which makes it possible to treat them as independent during scheduling. TIC is evaluated with an diesel EMS as example. We observed a WCET speed-up of 2.7 on a 4-core MCE and 4.5 on a 8-core MCE. However, TIC enlarges the end-to-end path delay as well. The results indicate that the speed-up outweighs the increase in the delay. Nevertheless, future work considers a more detailed investigation of the impact of TIC on the end-to-end path delay.

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no. 287519 (parMERASA) and the ARTEMIS Joint Undertaking no. 621429 (EMC^2).

References

[1] AUTOSAR consortium. AUTomotive Open System ARchitecture (AUTOSAR). Standard (v4.1). 2014. URL: www.autosar.org.

[2] C. Ballabriga, H. Cassé et al. OTAWA: An Open Toolbox for Adaptive WCET Analysis. In *Proc. Software Technologies for Embedded and Ubiquitous Systems*. S. L. Min, R. Pettit et al., editors. Vol. 6399. In Lecture Notes in Computer Science. Springer, 2011.

[3] D. Cordes, P. Marwedel and A. Mallik. Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming. In *Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM Press, New York, USA, 2010.

[4] J. Forget, F. Boniol et al. Scheduling dependent periodic tasks without synchronization mechanisms. In *Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2010.

[5] Y. Kanehagi, D. Umeda et al. Parallelization of automotive engine control software on embedded multicore processor using OSCAR compiler. In *Proc. IEEE Cool Chips XVI (COOL Chips)*, 2013.

[6] J. Lehoczky, L. Sha and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proc. IEEE Real-Time Systems Symposium*, 1989.

[7] A. Monot, N. Navet et al. Multicore scheduling in automotive ECUs. In *Proc. 3AF/SEE/SIA Embedded Real Time Software and Systems Congr. (ERTSS)*, 2010.

[8] F. Nemati. Partitioned Scheduling of Real-Time Tasks on Multi-core Platforms. Licentiate. Mälardalen University, Department of Computer Science and Engineering, 2010.

[9] H. Ozaktas, C. Rochange and P. Sainrat. Automatic WCET Analysis of Real-Time Parallel Applications. In *Proc. 13th Int. Workshop Worst-Case Execution Time Analysis (WCET)*. Vol. 30. In OpenAccess Series in Informatics. Schloss Dagstuhl, Leibniz-Zentrum Informatik, Dagstuhl, Germany, 2013.

[10] H. Ozaktas, C. Rochange and P. Sainrat. Minimizing the cost of synchronisations in the WCET of real-time parallel programs. In *Proc. 17th Int. Workshop on Software and Compilers for Embedded Systems (SCOPES)*. ACM Press, New York, USA, 2014.

[11] M. Panić, S. Kehr et al. RunPar: an allocation algorithm for automotive applications exploiting runnable parallelism in multicores. In *Proc. 14th IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM Press, New York, USA, 2014.

[12] M. Paolieri, E. Quiñones and F. J. Cazorla. Timing effects of DDR memory systems in hard real-time multicore architectures: Issues and solutions. *ACM Trans. Embed. Comput. Syst. (TECS)*, 12(1s), 2013.

[13] A. Roca, C. Hernandez et al. Enabling High-Performance Crossbars through a Floorplan-Aware Design. In *41st IEEE Int. Conf. on Parallel Processing (ICPP)*, 2012.

[14] R. Wilhelm, D. Grund et al. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Trans. Comput.-Aided Design Integr. Circuits. Syst. (CAD)*, 28(7), 2009.