Time Triggered Scheduling Analysis for Real-Time Applications on Multicore Platforms

Matthias Freier
Corporate Sector Research Schwieberdingen
Robert Bosch GmbH, Germany
Email: matthias.freier@de.bosch.com

Jian-Jia Chen
Department of Computer Science
TU Dortmund University, Germany
Email: jian-jia.chen@cs.uni-dortmund.de

***Abstract*—Scheduling of real-time applications for multicore platforms has become an important research topic. For analyzing the timing satisfactions of real-time tasks, most researches in the literature assume independent tasks. However, industrial applications are usually with fully tangled dependencies among the tasks. Independence of the tasks provides a very nice abstraction, whereas dependent structures due to the tangled executions of the tasks are closer to the real systems.**

This paper studies the scheduling policies and the schedulability analysis based on independent tasks by hiding the execution dependencies with additional timing parameters. Our scheduling policy relates to the well-known periodic task model, but in contrast, tasks are able to communicate with each other. A feasible task set requires an analysis for each core and the communication infrastructure, which can be performed individually by decoupling computation from communication in a distributed system. By using a Time-Triggered Constant Phase (TTCP) scheduler, each task receives certain time-slots in the hyper-period of the task set, which ensures a time-predictable communication impact.

In this paper, we provide several algorithms to derive the time-slot for each task. Further, we found a fast heuristic algorithm to calculate the time-slot for each task, which is capable to reach a core utilization of 90% by considering typical industrial applications. Finally, experiments show the effectiveness of our heuristic and the performance in different settings.

I. Introduction

The demand of more computing power for real-time systems carries on the research of multicore scheduling in academia and industries. In order to guarantee the schedulability of a system with real-time tasks, most researches in the literature assume independent tasks. Modeling the executions with independent tasks is a nice abstraction, which greatly simplifies the scheduling analysis. However, a typical industrial application consists of tasks with fully tangled dependencies among each other. Considering a distributed application, such dependencies include communications between cores. Such communications may require a lot of time, if the worst case (WC) is assumed. An explicit model of communication is required to produce a tight worst-case execution time (WCET) bound for each task on a specific core. Without expressing communication, real-time communication is more likely to become a bottleneck in the schedulability analysis with an increasing number of cores.

This paper uses a system model based on hiding such dependencies with an additional communication task set, i.e., a computational task set and a communication task are both present for scheduling. The communication task set models the dependencies among tasks. Each communication task has a deadline to constrain the completion of its transmission. A computational task set consists of periodic tasks, which is a well-known task model for real-time applications [7]. For scheduling computational tasks, we use a time-triggered constant phase (TTCP) scheduler, which helps to decouple the schedulability analysis.

A TTCP scheduler executes a task in predefined time-slots to guarantee a contention-free schedule for all tasks mapped to a specific core, i.e. at most one task is non-preemptively scheduled at any point in time. Such predefined time-slots provide the knowledge of the starting and guaranteed completion time of task executions, which procures *a priori* known communication impacts. A TTCP scheduler on each core ensures time-predictable communication impacts, which severely reduce the worst-case analytical effort in the communication analysis. For decoupling the communication, two time bounds specify the limits of the time-slots to preserve time for the communication to other tasks. The concept of fixing the starting times of the tasks has already been presented by Marouf and Sorel [10]. In contrast to the method storing the schedule in tables, the constant phase allows an efficient implementation, because not all starting times of all jobs need to be stored.

By using a TTCP scheduler, the essential step is to determine the starting times for each task for its time-slot. The analysis of the schedulability with known starting time has been proven [10]. In this paper, we provide and explore several algorithms to derive the starting times for each task. We develop a heuristic algorithm, named LPF-LBF, which can calculate the time-slots in a fast and efficient way. Further, we apply an Satisfiability Modulo Theories (SMT) solver, which can calculate a feasible solution, if one exists. Experiments show that our algorithm reach a core utilization above 90% by considering typical industrial applications.

II. Related Work

The benefits for a deterministic communication model has been shown several times [6], [11]. Schranzhofer et al. [11] evaluate different resource access patterns to a common shared resource and propose to separate the execution phase from the communication read-write phases in order to reduce the completion time. We adopt this acquisition-execution-replication (AER) model to our approach, in order to get a tight WCET.

Kopetz et al. [6] propose a time-triggered protocol to get a predictable and composable system. In this paper, we define explicitly the time-triggered communication on the task level to get a deterministic behavior. Due to these concepts, several problems appear. A problem is to determine a time-triggered non-preemptive schedule, capable to satisfy the real-time constraints given by the application. Considering a multicore platform, the design space becomes very large for assigning the time-slots for the time-triggered scheduling. This large design space lead to complexity problem, in which the trade-off between optimality, runtime and accuracy challenge the developer. Getting a flexible system, on-line adjustment of the time-slots is a difficult issue despite the composability of the system. In this paper, we focus on determining a feasible schedule for a time-triggered scheduler.

A few papers are closely related to this paper. The pure schedulability analysis of the TTCP scheduler is done by Marouf and Sorel [10]. They use a similar task model and analyze the schedulability under given starting times. However, the problem of determining these starting times is an still open question. Kermia and Sorel [5] published a heuristic as well to determine the starting times, but unfortunately the heuristic of the paper is not repeatable. Nevertheless, our heuristic does not require to unroll the schedule till the hyper-period.

Biewer et al. [2] determine the starting times of a time triggered system by an SMT solver. We use this SMT solver to compare our heuristic against a solver based approach, but it does not scale to larger problems and takes much more time finding a solution. In this paper, we use an SMT solver and a fast heuristic algorithm to determine the time-slots.

III. System Model

This section presents dependent tasks and our scheduling policy to decouple the tasks among each other. Further, we assume a non-preemptive scheduler. An example demonstrates the usage of our system model.

A. Computational and Communication Task Model

All tasks are mapped to a multicore platform with m-cores for a given fixed mapping, and they are statically executed on the assigned core. The mapping is not the focus of this paper. Therefore, the approaches can also be used for heterogeneous cores after the task mapping is done.

For modeling the computation, we assume a periodic task set $\mathbf{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, which consists of n tasks considering their corresponding communications. Each task τ_i is specified by a worst-case execution time (WCET) C_i, a period P_{τ_i} and a relative deadline D_{τ_i}. Each task generates an infinite number of jobs $J_{i,k}$ of τ_i, in which $J_{i,k}$ arrives at $a_{i,k} = P_{\tau_i} \cdot k, \forall k \in \mathbb{Z}_0^+$. Each job starts executing at its starting time $s_{i,k}$ and should at least be completed at its absolute deadline $a_{i,k} + D_{\tau_i}$, otherwise the job violates its real-time constraint. By considering implicit deadlines $D_{\tau_i} = P_{\tau_i}$, two jobs generated by one task cannot legally occur at the same time. All tasks arrive synchronized at the same time $t = 0$ to release their first job $J_{i,1}$. The task model adopted in this paper is strictly periodic, in which the inter-arrival time of two consecutive jobs must be equal to P_{τ_i}.

For the communication between τ_i, we define a communication task set $\mathbf{C} = \{\kappa_1, \kappa_2, \ldots, \kappa_z\}$, which consists of z communication tasks. A communication task κ_j is characterized by its period P_{κ_j}, its worst-case traversal time WCTT_j, a source task τ_{SRC_j} for producing packets, a destination task τ_{DST_j} for consuming packets, and a relative deadline D_{κ_j}. Similar to computational tasks τ_i, each communication task κ_j generates an infinite number of packets $p_{j,l}$. A packet is emitted by a source task τ_{SRC_j} at its completion time and will be transmitted through the communication infrastructure to the destination task τ_{DST_j}. The time, when a packet is emitted is the communication arrival time $r_{j,l}$. Each packet $p_{j,l}$ transfers a specified amount of data, which requires at most the worst-case traversal time WCTT_j. If the packet $p_{j,l}$ is not transmitted completely within their absolute deadline $r_{j,l} + D_{\kappa_j}$, then the real-time constraint is violated. The corresponding period P_{κ_j} is determined according to

$$P_{\kappa_j} = \max(P_{\tau_{\text{SRC}_j}}, P_{\tau_{\text{DST}_j}}), \tag{1}$$

by using the higher period, unnecessary communications are avoided, if the sender or receiver cannot process a packet. In order to guarantee a transmission, we assume a constrained relative deadlines $D_{\kappa_j} \leq P_{\kappa_j}$.

B. Model of Decoupling the Dependencies with a Time-Triggered Constant Phase (TTCP) Scheduler

The problem has circular dependencies between computational analysis, which require the delays of κ_j, and communication analysis, which require the completion times of τ_i. In order to solve these circular dependencies, at first, each core is analyzed with a computational task set, and second, we analyze of the communication based on the determined parameters from the computational analysis. For getting two task sets, we introduce additional parameters to limit the time-slot assignment. Therefore, each computational task τ_i gets a lower bound B_{low_i} and an upper bound B_{up_i}. A task is only allowed to be executed between these two bounds. Moreover, the starting time for the communication is not defined.

We propose a time-triggered constant phase (TTCP) scheduler to process the computational task set $\mathbf{T}$ on each core, which defines the starting time for each task τ_i. The TTCP approach was also conceptionally and implicitly presented in [8]. For defining the starting times, a TTCP scheduler requires additional timing parameters called phases. A phase Φ_{τ_i} of a task τ_i defines the time shift $s_{i,k} - a_{i,k}$ between the arrival time $a_{i,k}$ and the corresponding starting time $s_{i,k}$ for each job $J_{i,k}$. The main idea of the TTCP is, that the phase Φ_{τ_i} is a constant parameter for all jobs of one task τ_i. The lower bound B_{low_i} and and upper bound B_{up_i} limit the execution to a certain time window $B_{\text{low}_i} \leq \Phi_{\tau_i}, \Phi_{\tau_i} + C_i < B_{\text{up}_i}$.

By defining the starting time of the computational tasks and preserving time for the communication the analysis can be performed step by step. So, each computational task τ_i gets a phase Φ_{τ_i}, a lower bound B_{low_i} and an upper bound B_{up_i} as additional parameters, see Figure 1.

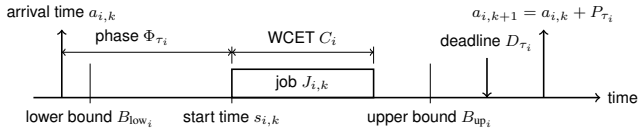

Figure 1: Computational task parameters under a TTCP scheduler: $\tau_i = f(C_i, D_{\tau_i}, P_{\tau_i}, \Phi_{\tau_i}, B_{\text{low}_i}, B_{\text{up}_i})$

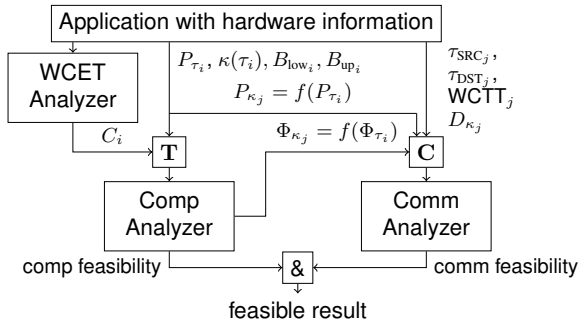

Figure 2: The design-framework between the computation and communication analyzer. The computation is analyzed, before the communication analysis is performed.

By using the calculated phases for computational task τ_i, each phase Φ_{κ_j} for the communicational task κ_j can be defined as

$$\Phi_{\kappa_j} = \Phi_{\tau_{\text{SRC}_j}} + C_{\tau_{\text{SRC}_j}}, \tag{2}$$

which can be used to set the communication arrival times $r_{j,l}$. A packet $p_{j,l}$ is periodically emitted at $\Phi_{\kappa_j} + l \cdot P_{\kappa_j}, \forall l \in \mathbb{Z}_0^+$, which is determined by its phase Φ_{κ_j}.

In the Design-Flow in Figure 2, all parameters for the computational task set $\mathbf{T}$ and the communication task set $\mathbf{C}$ can be extracted by the application. The WCET C_i can be determined with a WCET analyzer, e.g., [4]. Each period P_{τ_i} and the corresponding communication tasks assignments $\kappa(\tau_i)$, which represent the task-to-task communications, can directly be extracted from the application. B_{low_i} and B_{up_i} are determined by the application developer to set additional constraints for the phase assignment, e.g., a sensor triggered at time $t = 0$ requires some time for measuring, before a computational task can get the value for further processing. The computational analyzer (Comp Analyzer) returns the phases Φ_{τ_i} to schedule the computational tasks with a TTCP scheduling policy. With the feasible set of Φ_{τ_i}, the communication phases Φ_{κ_j} can be calculated. The relative deadline D_{κ_j} expresses the urgency of the communication from one task to another and strongly depends on the application. For example a communication from a less frequent to a more frequent task gets a relaxed $D_{\kappa_j} = P_{\kappa_j}$ and inverse communication gets a tight $D_{\kappa_j} \approx (1.5 \ldots 3) \cdot WCTT$. The communication analyzer (Comm Analyzer) checks the feasibility of the communication task set $\mathbf{C}$ based on the relative communication deadlines D_{κ_j}. In case of an negative feasibility result, further adjustments can been made by changing the parameters B_{low_i},B_{up_i}. If both analyzers return a feasible result the system is feasible under our TTCP scheduling policy.

The constant phase Φ_{τ_i} simplifies the scheduling problem

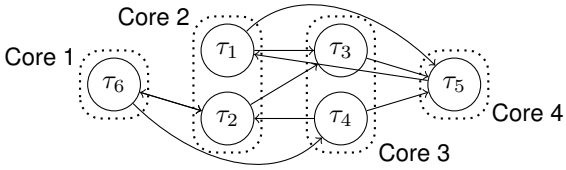

Figure 3: Example system: The task graph, where each arrow represent a communication task κ_j

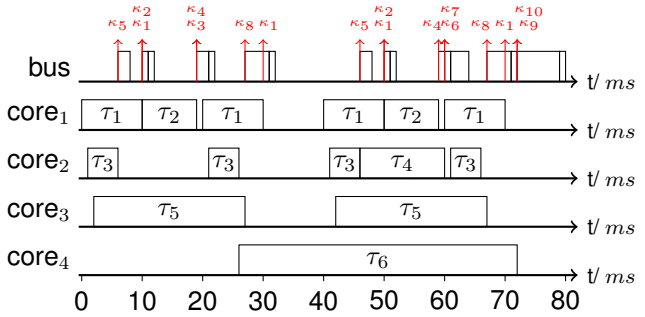

Figure 4: The complete schedule within one hyper-period $[0 \ldots 80] ms$ for the example task sets.

significantly. The construction of an overlap-free schedule by determining Φ_{τ_i} ensures a preemption- and contention-free schedule on each core. The contention-free schedule never has more than one job at any time to be scheduled, so that the scheduling itself is trivial. Each job is scheduled, when its starting time $s_{i,k} = \Phi_{\tau_i} + k \cdot P_{\tau_i}, \forall k \in \mathbb{Z}_0^+$ is reached. By knowing the arrival times $r_{j,l}$ of the packets through the TTCP scheduling of the tasks on each core, the analysis of the communication can be done very tightly.

C. Example

Here, we provide an example. We consider a platform with 4 cores connected by a bus with fixed-priority arbitration, e.g., a CAN-bus. Our example contains a computational task set $\mathbf{T}$ with 6 tasks and a communication task set $\mathbf{C}$ with 10 communication tasks.

The task partition is given, in which τ_1 and τ_2 are mapped to the same core, so are τ_3 and τ_4. The 10 communicational tasks model the inter-core communications for exchanging data among the tasks, which is shown in Figure 3. All computational tasks are scheduled in a TTCP policy, following their phases Φ_{τ_i}.The scheduling analysis can be performed without unfolding the entire schedule in the hyper-period, but for this small example the unfolded schedule is shown in Figure 4. The knowledge of the release of each communication task results in tight analysis of their worst-case traversal response times (WCTRT), which are shown for this example in Figure 5.

D. Problem Definition

For a computational task set $\mathbf{T}$ and a communication task set $\mathbf{C}$, the problem is to

- define all phases Φ_{τ_i} of τ_i, $\forall \tau_i \in \mathbf{T}$,
- define all phases Φ_{κ_j} of κ_j, $\forall \kappa_j \in \mathbf{C}$,

such that the resulting TTCP schedule is feasible, in which all tasks τ_i and κ_j satisfy their real-time constraints.

The solution of this problem requires a feasibility test to guarantee the feasibility of the computational task set $\mathbf{T}$. Based on this test, we construct a heuristic algorithm to determine the phases Φ_{τ_i}. The feasibility test, the heuristic algorithm and an SMT solver are presented in Sections V and VI, respectively.

IV. Communication Analysis

In this section, we describe the communication analysis, which represents the communication analyzer from Figure 2. The analysis has to find a feasible communication schedule, such that each communication task $\kappa_j \in \mathbf{C}$ is able to meet its deadline. As a precondition, each phase Φ_{κ_j} of the communication task set is already calculated, so that the phase Φ_{τ_i} of τ_i is also known. By knowing each relative deadline D_{κ_j} of communication task κ_j, the feasibility is calculated by verifying whether $\text{WCTRT}_j \leq D_{\kappa_j}, \forall \kappa_j$.

The worst-case traversal response time (WCTRT_j) of κ_j is defined as the time between the arrival of a communication request and its fully transmission through the network in the worst case. For this analysis, the decoupling into a computation and communication analysis generates a tighter bound to the worst-case, because the arrival times according to Equation (2) are known. However, the WCTRT_j needs to be determined by an upper bounded approximation.

For an *a priori* known communication impact, the analysis can be done by evaluating the schedule in the hyper-period. Thus, we iterate chronologically through time till the hyper-period is reached and determine the communication schedule. Hence, such approach provide a tight WCTRT_j for each κ_j inside the hyper-period. Due to the predictable and periodic communication impact, the evaluation of one hyper-period is sufficient to determine WCTRT_j. Thus, we can simply assume, that a suitable communication analyzer exists.

V. Feasibility Test for given $\Phi_{\tau_i}, \forall \tau_i \in \mathbf{T}$

In this section, we present an efficient algorithm to check the feasibility of our task model (Section III-A), which is based on the result from Marouf et al. [10]. In Theorem 1 the feasibility of task model can be found by comparing all the tasks among each other regarding to a time overlap.

Theorem 1. *(from Marouf et al. [9]) A task set* $\mathbf{T}$ *can feasibly be scheduled under a TTCP Scheduler if, and only if all pairs of tasks* τ_i *and* τ_j *satisfy*

$$C_j \leq (\Phi_{\tau_i} - \Phi_{\tau_j}) \ mod \ gcd_{i,j} \leq gcd_{i,j} - C_i, \qquad (3)$$

where $gcd_{i,j}$ *is the greatest common divisor of the period of task* τ_i *and* τ_j *and* $\Phi_{\tau_i} \geq \Phi_{\tau_j}$.

Proof: It is proved by [9]. ∎

For a better presentation, we rephrase Theorem 1 into Corollary 1.

Corollary 1. *For two periodic time-triggered constant phase scheduled tasks* τ_i *and* τ_j *with known* Φ_{τ_i} *and* Φ_{τ_j}*, suppose that* $\Psi_i = \Phi_{\tau_i} \ mod \ gcd_{i,j}$*,* $\Psi_j = \Phi_{\tau_j} \ mod \ gcd_{i,j}$*. These two tasks are feasibly scheduled by TTCP if, and only if,*

$$\begin{aligned}&((\Psi_i \geq \Psi_j) and (\Psi_i \geq \Psi_j + C_j) and (\Psi_j \geq \Psi_i + C_i - gcd_{i,j}))\\ or&((\Psi_i < \Psi_j) and (\Psi_j \geq \Psi_i + C_i) and (\Psi_i \geq \Psi_j + C_j - gcd_{i,j}))\end{aligned} \qquad (4)$$

Proof: Based on Theorem 1, we show that the conditions described in Equation (3) are equivalent to those in Equation (4). Therefore, we define the phases by $\Phi_{\tau_i} = \Psi_i + k \cdot gcd_{i,j}$ and $\Phi_{\tau_j} = \Psi_j + l \cdot gcd_{i,j}$, $k, l \in \mathbb{Z}_0^+$. The middle part of Equation (3) can be expressed by

$$(\Psi_i + k \cdot gcd_{i,j} - (\Psi_j + l \cdot gcd_{i,j})) \bmod gcd_{i,j} \qquad (5)$$

$$= ((k-l) \cdot gcd_{i,j} + (\Psi_i - \Psi_j)) \bmod gcd_{i,j}. \qquad (6)$$

Considering $\Phi_{\tau_i} \geq \Phi_{\tau_j}$ from Theorem 1, when $\Psi_i \geq \Psi_j$, Equation (6) is equal to $\Psi_i - \Psi_j$ otherwise, Equation (6) is equal to $\Psi_i - \Psi_j + gcd_{i,j}$. Thus, we yield 2 different cases:

- **case 1** ($\Psi_i \geq \Psi_j$)**:** $C_j \leq \Psi_i - \Psi_j \leq gcd_{i,j} - C_i$, which is equivalent to the first part of Equation (4).
- **case 2** ($\Psi_i < \Psi_j$)**:** $C_j \leq \Psi_i - \Psi_j + gcd_{i,j} \leq gcd_{i,j} - C_i$, which is equivalent to the second part of Equation (4).

As a result, we know that either case 1 or case 2 hold. ∎

Based on Corollary 1, it is easy to build an algorithm to get the feasibility under given phases Φ_{τ_i}. If there is no conflict of any two tasks, then the computational task set is feasible. Otherwise, we can conclude infeasibility of the resulting TTCP schedule. The additional parameters D_{τ_i}, B_{low_i} and B_{up_i} can easily be checked by testing $(B_{\text{low}_i} \leq \Phi_{\tau_i}) \wedge (\Phi_{\tau_i} + C_i \leq B_{\text{up}_i} \leq D_{\tau_i} \leq P_{\tau_i})$.

VI. Phase assignment algorithm

In this section, the problem of calculating the parameters $\Phi_{\tau_i} = f(\mathbf{T})$ for a TTCP scheduler is discussed. Due to the problem definition, a feasible set of Φ_{τ_i} needs to be determined. Here, we present a heuristic algorithm, called "lower periods first with lower bound first (LPF-LBF)", to calculate Φ_{τ_i} for all tasks $\tau_i \in \mathbf{T}$ and an exhaustive approach by using an Satisfiability Modulo Theories (SMT) solver.

A. Heuristic LPF-LBF

Our greedy algorithm sorts the tasks according to their period $P_{\tau_1} \leq P_{\tau_2} \leq \ldots \leq P_{\tau_n}$, in which the first task τ_1 has the lowest period. If the tasks have the same period, then the tasks with the lower bound B_{low_i} are ordered first. The algorithm searches greedily for gaps in the schedule without reassigning a phase, shown in Algorithm 1.

After sorting the task set, the algorithm iterates over all tasks and assigns the phases Φ_{τ_i} step by step. For calculating a feasible Φ_{τ_i}, a hypothetical phase Ψ_i is assumed.

A feasibility test against all other tasks is necessary to ensure, that the hypothetical phase Ψ_i is valid. If Ψ_i is invalid, then the resulting conflict indicates the calculation of the next potential free time slot for Ψ_i, in which perhaps no conflict occurs. In case Ψ_i exceeds its own period P_{τ_i}, this algorithm cannot find a feasible solution for phase assignment in a TTCP scheduler. Corollary 1 defines the feasibility test. A tighter condition due to infeasibility is given by the upper bound B_{up_i}. Considering an all-over chronological search through the time

Algorithm 1 Algorithm LPF-LBF

Input: computational task set $\mathbf{T}$, $\tau_i \in f(C_i, P_{\tau_i}, B_{\text{low}_i}, B_{\text{up}_i})$;
Output: all phases Φ_{τ_i};

```
1: Sort τ_i according RM, same periods according lower B_low_i first;
2: for i = 1, ··· , n stepped by 1 do
3:     Ψ_i ← B_low_i; t ← false;
4:     while Ψ_i < P_τi and t = false do
5:         t ← true;
6:         if B_up_i − C_i < Ψ_i then
7:             return "not feasible";
8:         for j = 1, ··· , (i − 1) stepped by 1 do
9:             Calculate the gcd_i,j of P_τj , P_τi;
10:            Ψ'_i ← Ψ_i mod gcd_i,j;
11:            Ψ'_j ← Ψ_j mod gcd_i,j;
12:            if (Ψ'_j < Ψ'_i) then
13:                if (Ψ'_i < Ψ'_j+C_j) then
14:                    Ψ_i ← Ψ_i + Ψ'_j + C_j − Ψ'_i; t ← false;
15:                else if (Ψ'_j+gcd_i,j < Ψ'_i+C_i) then
16:                    Ψ_i ← Ψ_i + Ψ'_j + gcd_i,j + C_j − Ψ'_i; t ← false;
17:            else
18:                if (Ψ'_j < Ψ'_i+C_i) then
19:                    Ψ_i ← Ψ_i + Ψ'_j + C_j − Ψ'_i; t ← false;
20:                else if (Ψ'_i+gcd_i,j < Ψ'_j+C_j) then
21:                    Ψ_i ← Ψ_i + Ψ'_j + C_j − (Ψ'_i + gcd_i,j); t ← false;
22:    Φ_τi ← Ψ_i;
23: return all phases Φ_τi and task set is "feasible";
```

of Ψ_i, if $\Psi_i + C_i > B_{\text{up}_i}$ is true, this task τ_i indicates the problem to an infeasible task set.

B. Satisfiability Modulo Theories solver

The phase assignment problem can be solved by a Satisfiability Modulo Theories (SMT) solver, like [3]. Therefore, we formulate the phase assignment problem as an SMT problem, consisting of a set of logical and linear expressions. The SMT solver returns one valid parameter setting, which satisfies the problem, if a feasible parameter setting exists.

There are two possibilities to formulate the SMT setting. On one hand, the problem can be defined based on the task level by using all equations including the modulo operation from the feasibility test (Equation 4). On the other hand, we can unfold all jobs in the hyper-period and formulate the SMT setting on a job-level. Our experimental results show that formulating the problem in the task-level indeed requires more runtime in the SMT solver than the job-level formulation. Therefore, all the results in Section VII-B are based on the job-level formulation. Due to space limitation the detailed SMT setting formulation is not shown in this paper.

VII. Experiments

In this section, several experiments are presented to show the effectiveness of the TTCP scheduling approach and our proposed heuristic.

A. Application information and experimental setup

We assume a hard real-time application, like a control application, with a high demand of computational power, wherefore a multicore platform is required. The applications we focused on, are characterized by a large amount of tasks, i.e. 100–1000 tasks, in which there is no heavy task with high utilization. Due to the large amount of tasks, a lot of preemption would occur, if the system allows preemption. So our applications is scheduled in a non-preemptive manner. Moreover, we generate a number of synthetic task sets $\mathbf{T}$ and $\mathbf{C}$, which use real application characteristics to get reliable results and to test our approach under different conditions. The periods P_{τ_i} are harmonic, because industrial applications tend to use harmonic periods. For comparison, we use a Rate-Monotonic non-preemptive (RMnP) version, like [1].

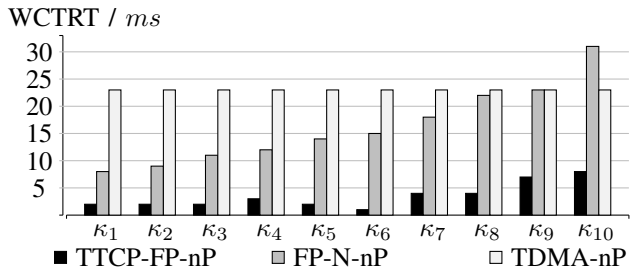

Figure 5: WCTRT for the motivative example: The analysis for the WCTRT in comparison of different non-preemptive (nP) versions of communication arbitration policies.

All experiments were performed on an Intel i5-2520M processor with 2.5GHz and 8GB RAM. For the SMT solver, we use the Z3 solver (version 4.3.2.0) [3] with a 10 min timeout, i.e. after this time the solution is neither feasible nor infeasible. The time out results are shown as gray areas in the plots. It is worth to be mention, that for larger task sets the SMT solver failed, because of its memory requirement. For such cases only the heuristic can provide a solution, but this is not included in our presentation, due to space limitation.

B. Experimental results

1) WCTRT for the motivative example: We demonstrate the reduction of the WCTRT for the time-triggered approach, which indicate an increased predictability of the system. Figure 5 illustrates the WCTRT for all communication tasks κ_j from our example (Figure 3). The priority based communication arbiter uses the index j as the priority of the communication tasks κ_j where κ_1 has the highest priority.

We compare our proposed TTCP scheduler, which releases communication task with fixed-priority arbitration (TTCP-FP), against a time division multiple access arbitration (TDMA) and a fixed-priority arbitration (FP-N). If the starting times of the communication tasks are unknown, a simultaneous release is assumed in order to bound the worst-case. In Figure 5, we calculated the WCTRT from our motivative example.

2) Dynamic scheduling approach: In order to evaluate our heuristic algorithm LPF-LBF, we compare it against a Rate-Monotonic non-preemptive (RMnP) scheduler. The RMnP feasibility test is implemented according to [1]. Figure 6 shows the comparison, in which the simulator generates 100 computational task sets, assigns the priorities or phases and plots the feasibility ratio. Our heuristic can reach a utilization of above 90%, which is close the SMT solver approach.

If there are only small task with low utilization, RMnP is nearly Rate-Monotonic preemptive, which gets a very high

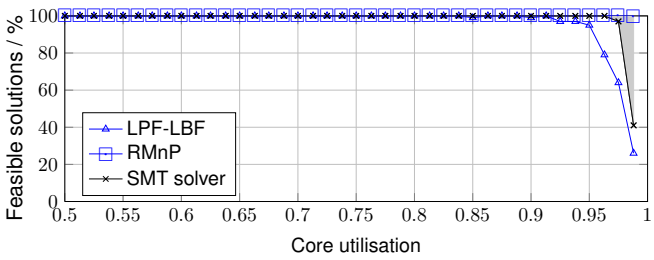

Figure 6: Dynamic scheduling approach: Our heuristic approach against Rate Monotonic non-Preemptive (RMnP).

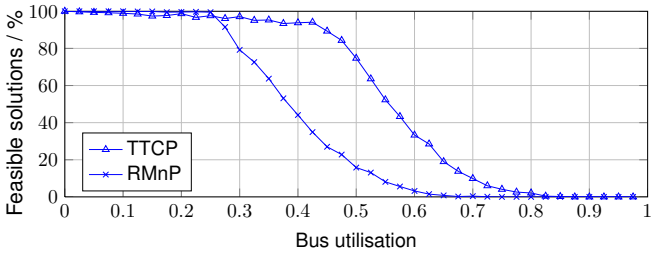

Figure 7: Communication enhancement: Different bus utilizations by constraining D_{κ_j}.

utilisation rate $U_{\text{RMnP}} \approx 100\%$. The LPF-LBF algorithm performs very well and is capable to reach utilization above 90%, which reaches almost the same maximal utilization as RMnP.

3) Communication enhancement: Another experiment focuses to the network analysis. Similar to our motivational example, we consider a 4-core platform with 250 computational tasks and 500 communication tasks randomly mapped to each other. Figure 7 presents a feasibility ratio for the system, in which we explore different bus utilizations. Each of the four cores has a utilisation of 50% in this experiment. The bus is scheduled with a non-preemptive fixed-priority arbitration, e.g. a CAN bus, in which the communication arrival time $r_{j,l}$ is calculated with different approaches. Figure 7 shows that TTCP reaches a higher maximum utilization of the bus. In contrast to RMnP, the *a priori* knowledge of the arrival times of the communication tasks facilitates a tighter communication analysis, which increase the feasibility ratio of the bus in the experiment. As otherwise, the worst case has to be considered with bursty behavior.

4) Runtime measurements: In this experiment, we measure the time to find a solution for a certain task set with the SMT solver and the LPF-LBF heuristic. We generate 50 task sets with 75% utilization with harmonic non-heavy tasks and messure the runtime of getting the phase assignment. The algorithm terminates sometimes earlier, if the task set is infeasible. Note that, the runtime also depends on the overall utilization of the task set, because particularly the SMT solver requires much more time, if the solution space is small. Considering a larger hyper-period or an increasing number of task, the SMT solver requires more resources and, hence, it is incapable to derive a solution. With such conditions, our heuristic algorithm may still provide a solution using moderate resources. Figure 8 shows that in general the heuristic is approximately 3 magnitudes (10^3) faster, than the SMT solver.

Furthermore, a similar experiment shows, that LPF-LBF

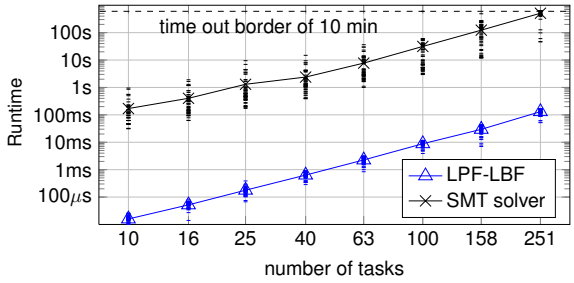

Figure 8: Runtime measurements: We measure the run time of the phase assignment using our heuristic or an SMT solver for different sizes of task sets.

runs approximately $5s$ for large computational task sets with 1000 tasks and $U = 75\%$.

VIII. Conclusion

This paper describes a system model, which is capable to handle a fully tangled application on a multicore platform. Our approach is to decouple of the dependencies among tasks into two independent task sets, which is closer to real systems. This paper presents a scheduling policy, named TTCP scheduler, to schedule such applications. The phase assignment plays a critical part getting a feasible task scheduled under a TTCP scheduler. We provide a heuristic algorithm which is capable to find a feasible solution for a high utilized task set with typical industrial characteristics. Several experiments reveal the effectiveness and the limits of different approaches. Our heuristic algorithm LPF-LBF is very fast, reaching a high utilization of above 90% utilization. An SMT solver find a solution of our problem, if one exists, but it takes a lot of time and resources to perform the analysis. Our measurements show, that the SMT solver takes approximately 3 magnitudes more time (10^3) in comparison to our heuristic.

References

[1] M. Bertogna, G. Buttazzo, and G. Yao. Improving feasibility of fixed priority tasks using non-preemptive regions. In *RTSS'11*, pages 251–260, Washington, DC, USA, 2011.

[2] A. Biewer, J. Gladigau, and C. Haubelt. Towards tight interaction of asp and smt solving for system-level decision making. In *Architecture of Computing Systems (ARCS)*, pages 1–7, Feb 2014.

[3] L. M. de Moura and N. Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[4] C. Ferdinand. AbsInt Angewandte Informatik GmbH. aiT: worst-case execution time analyzers. *http://www.absint.com/ait*, 2012.

[5] O. Kermia and Y. Sorel. A rapid heuristic for scheduling non-preemptive dependent periodic tasks onto multiprocessor. In *ISCA PDCS*, 2007.

[6] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[7] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.

[8] M. Lukasiewycz, R. Schneider, D. Goswami, and S. Chakraborty. Modular scheduling of distributed heterogeneous time-triggered automotive systems. In *ASP-DAC'12*, pages 665–670, 2012.

[9] M. Marouf, L. George, and Y. Sorel. Schedulability analysis for a combination of non-preemptive strict periodic tasks and preemptive sporadic tasks. In *ETFA'12*, pages 1–8, Sept 2012.

[10] M. Marouf and Y. Sorel. Schedulability conditions for non-preemptive hard real-time tasks with strict period. In *RTNS'10*, France, 2010.

[11] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo. Worst-case response time analysis of resource access models in multi-core systems. In *DAC '10*, pages 332–337, NY, USA, 2010. ACM.