Time-Triggered Communication Scheduling Analysis for Real-Time Multicore Systems

Matthias Freier
Corporate Sector Research Renningen
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*—The demand for more computing power in current real-time systems carries on the development and research on multicore devices. Especially for hard real-time applications, like an engine control system, the software needs to be distributed and scheduled effectively. These applications consist of many tasks, which communicate data among each other. Considering a multicore system, communication between cores may require a lot of time. A bus architecture becomes a communication bottleneck with an increasing number of cores. Therefore, we consider a scalable communication structure like a Network-on-Chip (NoC). This paper studies the schedulability analysis of tangled tasks by resolving the communication dependencies with a Time-Triggered Constant Phase (TTCP) scheduler. A TTCP scheduler assigns periodic time slots for each computation and each communication entity. With the TTCP approach, we can highly utilize the NoC and the cores considering a tangled task model. However, this approach requires a method getting a feasible set of these time slots. We provide a schedulability analysis and a heuristic algorithm, that runs in pseudo-polynomial time complexity, for assigning the time slots. Experiments confirm this result and show the effectiveness of our heuristic algorithm for assigning the time slots for our approach. For typical industrial task sets with 1000 tasks, our approach can utilize the NoC by around 60%, while holding all real-time constraints.**

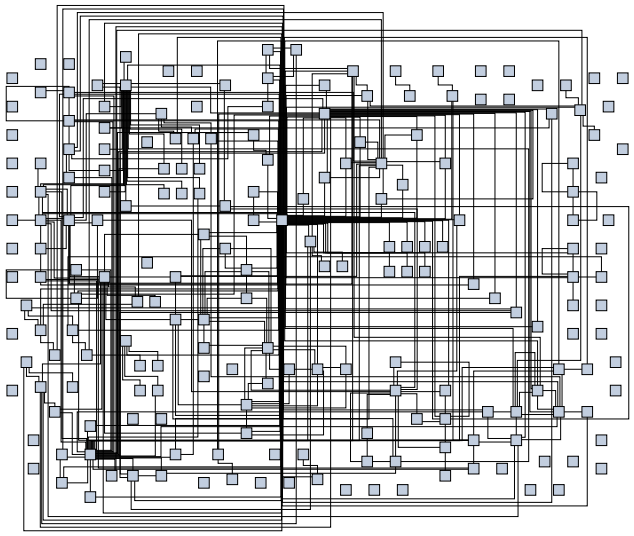

Figure 1: Part of the task graph of a current engine control application with 206 computational tasks and 334 dependencies among them.

I. Introduction

Current and future real-time embedded systems apply multicore devices to satisfy the demand for more computing power. Considering an engine control unit, new features need to be implemented in order to tackle for example the reduction of CO_2 emissions and fuel consumption. Due to the power wall [7], the multicore system is the most promising way to get more computing power than a single-core platform. Unfortunately, the resources of hard real-time platforms are limited, such that the platform has to be used efficiently. Especially, the software distribution and the schedule need to be adjusted to the platform appropriately.

Most researchers assume independent tasks for distribution and scheduling, but industrial applications indicate that tasks often communicate with each other. For example, Figure 1 shows a part of the task graph of a current engine control application for demonstrating the dependencies among the tasks. The communication between tasks composes a tangled task set, which is difficult to partition and to schedule. Regarding a multicore platform connected by a bus, the communication between cores may require much more time and causes a bottleneck with an increasing number of cores. Therefore, we consider a scalable communication structure, namely a Network-on-Chip (NoC).

There exist several approaches to schedule a NoC with real-time constraints [5], [11], [12]. The Æthereal NoC [5] approach inserts slot tables for switching the packets at each switch. Therefore, Lu and Jantsch [8] propose a global time-division multiplex schedule, which they apply on the Æthereal NoC to get a predictable and contention-free communication schedule. For a large application, which highly utilizes the platform, these slot tables would exceed its hardware limitations.

By using a time-triggered scheduling approach [4], [14], each task is statically scheduled in a time slot defined *a priori*. Considering a deterministic NoC, like the DSPIN NoC [11], the packets are sent at a certain time and traverse the NoC also at a certain time. The schedule is constructed, such that no contention neither in the cores nor in the NoC occurs. Therefore, only the sending times of the packets are required to be scheduled and analyzed in the communication entities. But, another problem is to represent the schedule for a large application with a lot of tasks (100–1000) and

978-1-4673-7711-9/15/$31.00

to verify the feasibility by an algorithm with polynomial runtime complexity. Therefore, the concept of a time-triggered scheduler with pure periodic time slots [4], [10], [15] is able to conquer this complexity issue.

If the applications only consist of independent clusters, it is possible to partially construct and analyze the time-triggered schedule, named divide and conquer method [16]. We assume that our application is fully tangled like shown in Figure 1, in which the divide and conquer method is not applicable.

The time-trigged scheduling analysis with pure periodic tasks is presented by Marouf and Sorel [10] for tasks placed on the same core. Kermia and Sorel [6] propose a heuristic algorithm to assign time slots based on a computational task set with precedence constraints. This model does neither include communication between tasks nor a communication fabric like a NoC.

Closely related, some works [1], [2], [9], [16], [17] propose to determine and verify the schedule by a solver. They formulate the problem of determining a feasible time-triggered schedule as a set of equations and use a solver to calculate the schedule. With regard to our application size, the solver-based approach does not scale to larger task sets because of the exponential time complexity of this method.

This paper studies the schedulability analysis of these fully tangled real-time tasks to adjust the typical industrial application effectively to the platform with a NoC. We resolve these dependencies with a predictable Time-Triggered Constant Phase (TTCP) scheduler. A TTCP scheduler assigns periodic time slots for each task and each communication entity, such that these constraints are satisfied. The benefit of these periodic *a priori* known time slots is to easily store them and to be able to analyze the system very tightly, because no over-approximation, like the worst-case communication pattern needs to be assumed [13]. Therefore, a TTCP scheduler can schedule packets in a NoC with a high utilization. The path for each packet is preserved *a priori* without unrolling till the hyper-period. However, this approach requires a method to determine a feasible set of these time slots. Regarding typical industrial applications with 100–1000 tasks and much more communication relations between them, a scalable analysis algorithm is needed to handle this complexity.

The **contributions** of this paper are

- an approach of timely preserving the path in a NoC for each packet such that the NoC and the core can be highly utilized,
- an efficient and tight feasibility analysis for a NoC without unfolding to the hyper-period,
- a heuristic algorithm to calculate a feasible set of time-slots for all tasks in pseudo-polynomial time complexity.

Experiments confirm that the NoC with a feasible TTCP schedule can be utilized with much more traffic than the maximum possible shared bus load. Furthermore, our heuristic finds a feasible slot assignment even for applications with a high utilization. For typical industrial task sets with 1000 tasks, our approach can utilize a 3×3 NoC by around 60%, i.e. on average a core sends 60% of its time data to another core without interfering other packets.

Outline. In Section II, we define our tangled task model including our platform. Section III presents the feasibility analysis of the computational and communication tasks. Our phase assignment approach of determining the time slots for each task is presented in Section IV. We examine our experiments in Section V to show the effectiveness of our heuristic algorithm. Finally, Section VI concludes this paper.

II. System model

In this section, we present our tangled task model, which consists of a computational task set $\mathbf{T}$ and a communication task set $\mathbf{C}$. Furthermore, the time-triggered scheduling approach is presented, which can schedule the system efficiently.

A. Platform model

We assume a platform described by a graph, as shown in Figure 2, in which each node is either a core $\mathcal{C}_c \in \mathcal{C}$ for computation or a switch $\mathcal{S}_s \in \mathcal{S}$ for communication, $c \in \{0, 1, \dots |\mathcal{C}| - 1\}$, $s \in \{0, 1, \dots |\mathcal{S}| - 1\} \subset \mathbb{Z}_0^+$. Each edge between two nodes is depicted as an unidirectional link $\mathcal{L}_l \in \mathcal{L}$, $l \in \{0, 1, \dots |\mathcal{L}| - 1\} \subset \mathbb{Z}_0^+$. $|\mathcal{C}|$, $|\mathcal{S}|$, $|\mathcal{L}|$ are the number of cores, the number of switches and the number of links, respectively.

A Network-on-Chip (NoC) is composed of these switches $\mathcal{S}$ and links $\mathcal{L}$. Furthermore, we assume that each core $\mathcal{C}_c$ is connected via two links (one for each direction) with its corresponding switch $\mathcal{S}_s$. The NoC establishes a 2D-mesh structure, as shown in Figure 2, thus $|\mathcal{C}| = |\mathcal{S}|$. Each switch has the same arbiter, like the DSPIN NoC [11] implementation, such that multiple paths can be established simultaneously.

Furthermore, the bandwidth of each link is upper-bounded by a specified constant $b_{\mathcal{L}}$. Each link delays the data by a certain amount of time, namely *link delay* $d_{\mathcal{L}}$. Similarly, to forward data via a switch, we assume an upper-bounded switch delay $d_{\mathcal{S}}$. We assume an inner switch architecture, as shown in Figure 3, which allows a full connectivity between the input and output ports like that used in the DSPIN NoC [11]. Each output port has its own arbiter and each input port has its own queue.

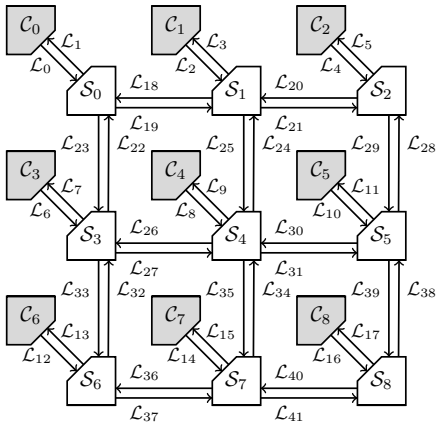

Figure 2: Example of a platform with 9 cores connected by a 3×3 2D-mesh NoC

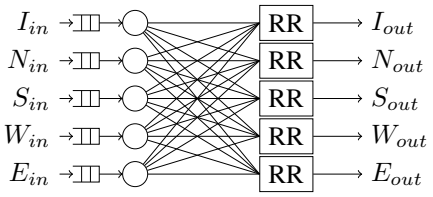

Figure 3: Inner switch structure: Each output port has its own arbiter (e.g. round robin (RR)) and each input port has its own queue. This type is used in the DSPIN NoC [11]. The ports are named based on their direction with inner (I), north (N), south (S), west (W) and east (E).

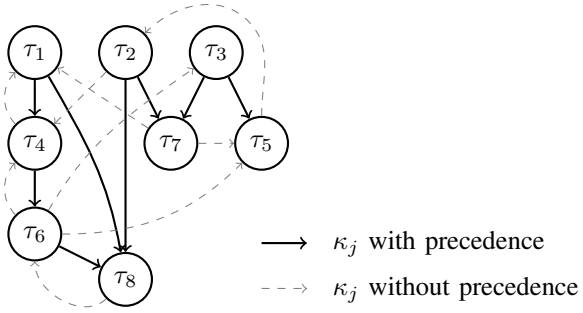

Figure 4: Example DAG of an application with 8 tasks and 17 communication tasks

Each core has a Network Adapter (NA) to handle incoming and outgoing data. The NA is able to shift outgoing data by a certain delay, which is used by our approach later on. Each core owns sufficient local memory for storing its program code and temporally data. This local memory is only directly accessible by the assigned core and the corresponding NA.

B. Computational task model

A computational task $\tau_i \in \mathbf{T}$ is defined by its worst-case execution time (WCET) W_{τ_i}, its period P_{τ_i}, its relative deadline D_{τ_i}, its assigned core c_i and its predecessor tasks $Q_i = \{\tau_{v_1} \dots \tau_{v_q}\} \in \mathbf{T}$, $i \in \{0, 1, \dots |\mathbf{T}|\} \subset \mathbb{Z}_0^+$. The computational task set $\mathbf{T}$ contains $|\mathbf{T}|$ computational tasks. Each task τ_i releases an infinite number of jobs $J_{\tau_i,k}$ with $J_{\tau_i,k+1} = J_{\tau_i,k} + P_{\tau_i}$, which arrive at their arrival time $a_{\tau_i,k}$, $k \in \mathbb{Z}_0^+$. Each job $J_{\tau_i,k}$ starts its execution at its starting time $S_{\tau_i,k}$. The periods of the tasks are harmonic, i.e. each period is an integer multiple of the lower periods in the task set $\mathbf{T}$. Due to the real-time constraints, each job has to be completed before its absolute deadline $a_{\tau_i,k} + D_{\tau_i}$, whereas we assume implicit relative deadlines $D_{\tau_i} = P_{\tau_i}$. In this paper, we assume a given task to core mapping, which is given by the parameter c_i.

A task can have a set of predecessor tasks Q_i, but all precedence relations between the tasks must be resolvable in a directed acyclic graph (DAG). An example of a simple DAG is given in Figure 4.

Definition 1. *(Precedence) If τ_i is a predecessor of τ_j, then $J_{\tau_i,l}$ needs to complete its execution and transmits its data, before $J_{\tau_j,k}$ starts, if $a_{\tau_i,l} \leq a_{\tau_j,k}$, where $a_{\tau_i,l}$ and $a_{\tau_j,k}$ are the corresponding arrival times of the jobs, $\forall l, k$.*

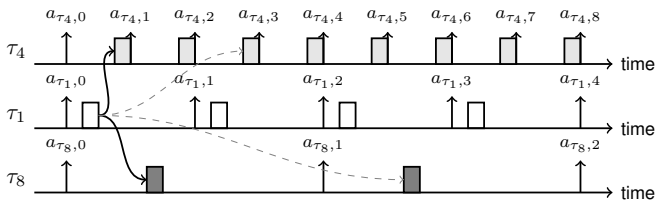

Figure 5: Example of precedence and non-precedence communication tasks. If the communication task is κ_j precedence, the destination jobs to handle the data are indicated with solid lines. If the communication task is κ_j non-precedence, the destination jobs to handle the data are indicated with dashed lines.

C. Communication task model

A communication task $\kappa_j \in \mathbf{C}$ is defined by a traversal time W_{κ_j}, a period P_{κ_j}, a route R_j through the network, a source task τ_{SRC_j} for producing communication packets, and a destination task τ_{DST_j} for consuming communication packets, $j \in \{0, 1, \dots |\mathbf{C}|\} \subset \mathbb{Z}_0^+$. The communication task set $\mathbf{C}$ contains $|\mathbf{C}|$ communication tasks.

Each communication task κ_j releases an infinite number of packets $p_{j,l}$, $l \in \mathbb{Z}_0^+$, one packet in each period P_{κ_j}. Each packet $p_{j,l}$ generates m flow control units (flits), which will be *individually* passed through the network via a store-and-forward policy. Each flit has a fixed size F regardless of the traversal time W_{κ_j} of κ_j. The maximum amount of data transmitted by each packet is $m \cdot F$, which defines the traversal time

$$W_{\kappa_j} = \frac{m \cdot F}{b_{\mathcal{L}}} \tag{1}$$

through a link, where $b_{\mathcal{L}}$ is the upper-bounded link bandwidth. For defining a period P_{κ_j} of a communication task κ_j, the common approach is to send the data to the destination after the execution of each job. If the periods of the source and destination task are not equal $P_{\tau_{\text{SRC}_j}} \neq P_{\tau_{\text{DST}_j}}$, we define

$$P_{\kappa_j} = max(P_{\tau_{\text{SRC}_j}}, P_{\tau_{\text{DST}_j}}) \tag{2}$$

to avoid futile communication and allow to communicate more packets. In order to keep the communication predictable, the system designer can choose a certain sending and receiving job for communication. Otherwise the first jobs would be used for communicating, which is explained in the following. If always the same specific job sends data to its specific destination job, the communication is deterministic.

There exist two types of communication tasks, namely κ_j *precedence* and κ_j *non-precedence*, depending on whether τ_{SRC_j} is a predecessor of τ_{DST_j}. Only certain jobs of τ_{SRC_j} have data to be sent to τ_{DST_j}. Due to harmonic periods of $\mathbf{T}$ and the setting that the first job of each task arrives at the same time 0, the jobs of task τ_{SRC_j} are indexed as the h-th jobs of τ_{SRC_j} sending data to its destination with

$$h = \left\lceil \frac{P_{\tau_{\text{DST}_j}}}{P_{\tau_{\text{SRC}_j}}} \right\rceil g, \qquad g \in \mathbb{Z}_0^+. \tag{3}$$

The corresponding job $J_{\tau_{\text{DST}_j},k}$ to receive the packet of the sending job $J_{\tau_{\text{SRC}_j},h}$ is defined as the earliest job of task τ_{DST_j} that

- starts its execution at time $S_{\tau_{\text{DST}_j},k}$ no earlier than $a_{\tau_{\text{SRC}_j},h}$ if κ_j *precedence*, or
- starts its execution at time $S_{\tau_{\text{DST}_j},k}$ no earlier than $a_{\tau_{\text{SRC}_j},h} + P_{\tau_{\text{SRC}_j}}$ if κ_j *non-precedence*.

An example for the precedence definition and the corresponding timing in the destination jobs is shown in Figure 5.

For ensuring a feasible communication, each packet $p_{j,l}$ needs to be completely transmitted before its absolute deadline $d_{\kappa_{j,l}} = S_{\tau_{\text{DST}_j},k}$, which is the starting time of the job of τ_{DST_j} for processing the packet. If the source task τ_{SRC_j} and the destination task τ_{DST_j} are placed on the same core, we assume the communication can be done in zero time. Nevertheless, a precedence relation might even exist between these tasks.

The route R_j of a communication task κ_j is defined by a list of the nodes beginning with the first switch, some intermediate switches $\mathcal{S}_s$ and the destination core $\mathcal{C}_{c_{\text{DST}_j}}$. For the 2D-mesh structure of the NoC, the routes are calculated by an X-Y routing policy [12]. The number of links on the route is denoted as $|R_j|$. Under a X-Y routing policy, a packet is sent at first to the switch with the same X position like the destination and secondly it goes directly on the Y-axis to the destination.

D. Time-Triggered Constant Phase (TTCP) scheduling

This section presents the scheduling approach, which is used to schedule both the computational and communication tasks. We assume a time-triggered scheduler, which assigns a constant phase to each task τ_i and κ_j, which is implicitly presented in [10], [15] and explicitly in [4] without considering NoCs. A phase determines the *a priori* known time shift between the start of a task and their arrival time. The TTCP scheduler assigns a certain periodic time slot for each task such that the task is only allowed to be executed in this slot. For the definition of the phases, the TTCP scheduler assumes that all tasks arrive synchronized at time 0 for releasing their first job $J_{\tau_i,0}$.

The idea of the constant phase is to reduce the number of timing parameters to define the starting times of the jobs. The phases can be stored very efficiently and the schedulability analysis can be performed efficiently as well. Furthermore, due to the time-triggered scheduling principle, the starting times are known *a-priori*, which ensures a tight and reliable scheduling analysis. The TTCP scheduler is a contention-free scheduler, i.e. the phases are set such that a resource is requested by at most one task at any time.

Each computational task τ_i and communication task κ_j have a phase, Φ_{τ_i} and Φ_{κ_j}, respectively. For the feasibility analysis, additional constraints are defined based on the combination of τ_i and κ_j. The phase of a communication task can only start after the computational task finishes, thus $\Phi_{\kappa_j} \geq \Phi_{\tau_{\text{SRC}_j}} + W_{\tau_{\text{SRC}_j}}$ need to be fulfilled.

We will define the corresponding timing parameters of κ_j based on the arrival time $a_{\tau_{\text{SRC}_j},h}$ of a job from τ_{SRC_j}, which triggers κ_j. For a defined phase Φ_{κ_j} of κ_j, the communication in the NoC is done through the X-Y routing protocol. Suppose the z-th node on the X-Y routing path R_j of κ_j, according to the NoC and the time-triggered schedule. Thus, the z-th node is used for this communication task in time interval $[a_{\tau_{\text{SRC}_j},h}+\Phi_{\kappa_j}+z(d_{\mathcal{S}}+d_{\mathcal{L}}), a_{\tau_{\text{SRC}_j},h}+\Phi_{\kappa_j}+z(d_{\mathcal{S}}+d_{\mathcal{L}})+W_{\kappa_j})$. Therefore, according to the above definitions, the communication task illegally transmits the packet to the destination later than $a_{\tau_{\text{SRC}_j},h} + \Phi_{\kappa_j} + |R_j|(d_{\mathcal{S}} + d_{\mathcal{L}}) + W_{\kappa_j}$, if there is no conflict in the path.

According to the definition of the two computational jobs that are communicating, represented by κ_j in Section II-C, we have the following three cases to set the absolute deadline of the communication job:

- $a_{\tau_{\text{SRC}_j},h} + \Phi_{\tau_{\text{DST}_j}}$ if κ_j *precedence*, or
- $a_{\tau_{\text{SRC}_j},h} + \Phi_{\tau_{\text{DST}_j}}$ if κ_j *non-precedence* and $P_{\tau_{\text{SRC}_j}} \leq P_{\tau_{\text{DST}_j}}$ and $P_{\tau_{\text{SRC}_j}} \leq \Phi_{\tau_{\text{DST}_j}}$, or
- $a_{\tau_{\text{SRC}_j},h}+\Phi_{\tau_{\text{DST}_j}}+P_{\tau_{\text{SRC}_j}}$ if κ_j *non-precedence* and $P_{\tau_{\text{SRC}_j}} > P_{\tau_{\text{DST}_j}}$ or $P_{\tau_{\text{SRC}_j}} > \Phi_{\tau_{\text{DST}_j}}$.

As a result, we can define the relative deadline of a communication task κ_j with respect to the arrival time of a job of its source computational task as follows:

$$D_{\kappa_j} = \begin{cases} \Phi_{\tau_{\text{DST}_j}} & \text{if } \kappa_j \textit{ precedence} \\ \Phi_{\tau_{\text{DST}_j}} + P_{\tau_{\text{SRC}_j}} \left\lceil \frac{P_{\tau_{\text{SRC}_j}} - \Phi_{\tau_{\text{DST}_j}}}{P_{\tau_{\text{DST}_j}}} \right\rceil & \text{otherwise.} \end{cases} \quad (4)$$

III. Feasibility Analysis and Problem Definition

In this section, we present the feasibility analysis for a TTCP scheduled system. We propose to use a TTCP scheduler on each core and to arbitrate the communication in a time-triggered manner, too. A time-triggered NoC implies that a packet of κ_j is delayed by the NA such that no contention occurs in the NoC. Furthermore, the schedule on each core is predictable and contention-free as well.

For determining conflicts in the TTCP scheduler the following theorem can detect a conflict between two tasks.

Theorem 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 a hypothetical phase Ψ with $\Psi_{\tau_i} = \Phi_{\tau_i}$ mod $gcd_{i,j}$, $\Psi_{\tau_j} = \Phi_{\tau_j}$ mod $gcd_{i,j}$ and $\Psi_{\tau_i} \geq \Psi_{\tau_j}$. These two tasks are feasibly scheduled by TTCP if, and only if,*

$$(\Psi_{\tau_i} \geq \Psi_{\tau_j} + W_{\tau_j}) \textit{and } (\Psi_{\tau_j} \geq \Psi_{\tau_i} + W_{\tau_i} - gcd_{i,j}) \quad (5)$$

Proof: This has been proven [4], [10]. ■

Focusing on the communication, a conflict between two communication tasks can be analyzed similar to computational tasks. Based on the X-Y routing protocol in the NoC, there exists at most one switch where two communication tasks can collide. Therefore, to detect the collision, we need to evaluate whether a link is used by two communicating tasks at the same time. Suppose two communicating tasks κ_i and κ_j. According to the X-Y routing, they collide on the switch $\mathcal{S}_C$ before the first shared link. Furthermore, suppose that $\mathcal{S}_C$ is the $z_{i,j}$-th node in the routing path R_i. The two communicating tasks do not collide with each other, if their intervals to use the switch $\mathcal{S}_C$ do not overlap with each other. For notational brevity, let the conflict matrix $\mathcal{M}_{i,j}$ be defined as follows:

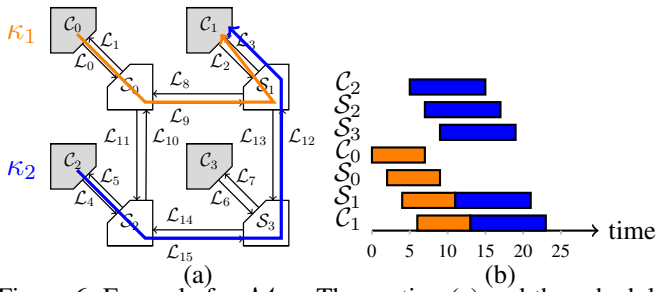

Figure 6: Example for $\mathcal{M}_{i,j}$: The routing (a) and the schedule (b) are presented for κ_1 and κ_2 colliding at switch $\mathcal{S}_1$, with $R_1 = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{C}_1\}$, $R_2 = \{\mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_1, \mathcal{C}_1\}$, $z_{1,2} = 2$, $z_{2,1} = 3$, $d_\mathcal{S} + d_\mathcal{L} = 2ms$, $\mathcal{M}_{1,2} = 4ms$, $\mathcal{M}_{2,1} = 6ms$, $W_{\kappa_1} = 7ms$, $W_{\kappa_2} = 10ms$, $\Phi_{\kappa_1} = 0$, $\Phi_{\kappa_2} = 5$.

$$\mathcal{M}_{i,j} = \begin{cases} z_{i,j}(d_\mathcal{L}+d_\mathcal{S}) & \text{if } R_i \text{ overlaps with } R_j \\ \emptyset & \text{otherwise.} \end{cases} \quad (6)$$

The example in Figure 6 demonstrates the usage of $\mathcal{M}_{i,j}$.

Theorem 2. *Suppose two periodic time-triggered constant phase scheduled communication tasks κ_i and κ_j with $\mathcal{M}_{i,j} \neq \emptyset$ and with known Φ_{κ_i} and Φ_{κ_j}. Let $\Psi_{\kappa_i} = (\Phi_{\kappa_i} + \mathcal{M}_{i,j})$ mod $gcd_{i,j}$, $\Psi_{\kappa_j} = (\Phi_{\kappa_j} + \mathcal{M}_{j,i})$ mod $gcd_{i,j}$ with $\Psi_{\kappa_i} \geq \Psi_{\kappa_j}$. These two tasks are feasibly scheduled by TTCP (in X-Y routing), if*

$$(\Psi_{\kappa_i} \geq \Psi_{\kappa_j} + W_{\kappa_j}) \textit{and } (\Psi_{\kappa_j} \geq \Psi_{\kappa_i} + W_{\kappa_i} - gcd_{i,j}) \quad (7)$$

Proof: With X-Y routing and no back pressure in the NoC, two packets can collide at most once at a particular switch. Otherwise, this particular switch would sequentialize the packets such that no further conflict can occur. The elapsed time getting to the critical switch is represented by $\mathcal{M}_{i,j}$ after the communicating task is started. Therefore, we can consider that this critical switch is the resource that may be used by more than one tasks in the time-triggered manner. The collision detection on this critical switch is hence the same as that in Theorem 1. ∎

Problem statement: For a given system, consisting of a platform with a NoC communication fabric, a computational task set $\mathbf{T}$ and communication task set $\mathbf{C}$, which are scheduled by a TTCP scheduler, the problem is to

- determine computational task phases Φ_{τ_i}, and
- determine communication task phases Φ_{κ_j},

such that all real-time and precedence constraints hold and the system is schedulable. Note that this is a satisfiability problem where each feasible solution has the same quality.

A feasible phase remains in a certain range,

$$0 \leq \Phi_{\tau_i} \leq P_{\tau_i} - W_{\tau_i} \quad (8)$$

$$\Phi_{\tau_{\text{SRC}_j}} \leq \Phi_{\kappa_j} \leq D_{\kappa_j} - W_{\kappa_j} - |R_j|(d_\mathcal{S} + d_\mathcal{L}), \quad (9)$$

where D_{κ_j} is the deadline for κ_j depending on κ_j is precedence or κ_j *non-precedence* (Equation (4)).

Additionally, each τ_i and κ_j task needs to be executed without any conflict with other tasks. All tasks are tested for conflicts to each other by using Theorems 1 and 2. If no conflict occurs and (8) and (9) hold, then our system can feasibly be scheduled under the given phases Φ_{τ_i} and Φ_{κ_j}. Note that the precedence constraints are not explicitly included in the equations. They are implicitly ensured by limiting the feasible range for each Φ_{κ_j}.

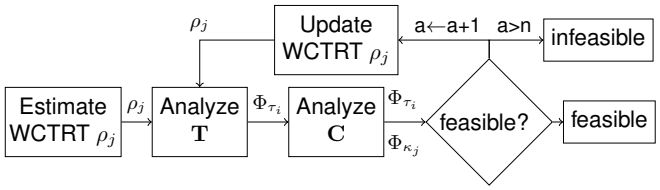

Figure 7: The workflow for solving the cyclic dependencies among each $\mathbf{T}$ and $\mathbf{C}$.

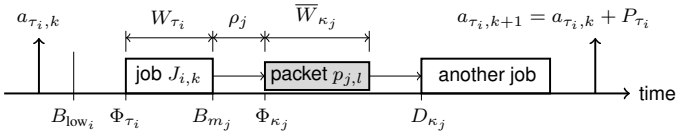

Figure 8: Additional parameters for TTCP scheduled tasks τ_i and κ_j, namely: B_{m_j}, B_{low_i}, ρ_j, $\overline{W}_{\kappa_j}$

IV. Phase Assignment

This section provides an overview of the method, we propose to solve the problem described above. Either a heuristic algorithm or the Satisfiability Modulo Theories (SMT) solver can determine the phases of our problem. Our approach is to perform the computational and communication phase assignment in a cyclic structure with a heuristic algorithm.

Figure 7 shows the iteration between the computation and communication analysis. Unfortunately, both analyses depend on each other. The idea is to analyze the computational and communication task sets separately. Preliminary experiments point out that a small number of iterations n is sufficient to get stable phases. Hence, we limited our heuristic to $n = 10$ iterations in the experiments. In the following subsections, the communication analysis and the computational analysis are presented. We introduce additional parameters, as shown in Figure 8, for assigning the phases to each task.

Each computational task τ_i gets a lower bound B_{low}, which limits the range of the phase. Similarly, each communication task κ_j gets a minimal communication phase $B_{m_j} = a_{\tau_{\text{SRC}_j,h}} + \Phi_{\tau_{\text{SRC}_j}} + W_{\tau_{\text{SRC}_j}}$. Considering the network delays, the *network traversal time* $\overline{W}_{\kappa_j} = W_{\kappa_j} + |R_j|(d_\mathcal{S} + d_\mathcal{L})$ is the maximum time between sending and receiving of a packet. As presented in Figure 8, the *packet delay* $\rho_j = \Phi_{\tau_{\text{SRC}_j}} - B_{m_j}$ determines the delay between the arrival of a communication task and the starting time of sending to the destination. For the first iteration, we estimate each $\rho_j = 0$, but further iterations can increase them.

A. Phase assignment Φ_{τ_i} under given ρ_j

This subsection explains the computational phase assignment Φ_{τ_i} for each computational task τ_i under given packet delays ρ_j. Since we have to satisfy the precedence constraints, we first perform topological ordering Ω_τ of the given task

graph (by ignoring all the non-precedence communication tasks in this step). Note that Ω_τ is an order of all τ_i.

Suppose that task τ_i is the i-th task in Ω_τ. Our heuristic assigns each Φ_{τ_i} of τ_i one by one following Ω_τ. For the i-th task τ_i, we first check the lower bound B_{low_i} for executing τ_i. For verification, we have to consider κ_j *precedence* and κ_j *non-precedence* that may affect the phase assignment of τ_i.

For defining this lower bound, let E_{prec,τ_i} be the set of κ_j, consisting of κ_j *precedence*, in which τ_i is τ_{DST_j} of κ_j, i.e., $E_{prec,\tau_i} = \{\kappa_j \mid \kappa_j \text{ precedence and } \tau_{\text{DST}_j} = \tau_i\}$. For the precedence constraint, we cannot assign the phase Φ_i of τ_i before the predecessors finish and their corresponding packets are received. Therefore, Φ_i cannot start before

$$b_{prec,i} = \max_{\kappa_j \in E_{prec,\tau_i}} (\Phi_{\tau_{\text{SRC}_j}} + W_{\tau_{\text{SRC}_j}} + \rho_j + \overline{W}_{\kappa_j}). \quad (10)$$

Similarly, E_{np,τ_i} only considers κ_j *non-precedence*. So, let E_{np,τ_i} be the set of communication tasks κ_j, consisting of κ_j *non-precedence*, in which τ_i is τ_{DST_j} of κ_j, i.e., $E_{np,\tau_i} = \{\kappa_j \mid \kappa_j \text{ non-precedence and } \tau_{\text{DST}_j} = \tau_i\}$. For the non-precedence constraint, we know that we cannot assign the phase Φ_i of task τ_i before the required packets from the sources have received. Therefore, Φ_i cannot start before

$$b_{np,i} = \max_{\kappa_j \in E_{np,\tau_i}} (B_{m_j} - P_{\tau_{\text{SRC}_j}} + \rho_j + \overline{W}_{\kappa_j}). \quad (11)$$

Hence, by respecting both precedence and non-precedence communication tasks, we cannot legally assign the phase of task τ_i less than B_{low_i}, where

$$B_{low_i} = \max(0, b_{np,i}, b_{prec,i}). \quad (12)$$

After B_{low_i} is calculated, we search chronologically through the time to get a feasible phase Φ_{τ_i}, starting from B_{low_i}. We verify whether a phase assignment for computational tasks is feasible or not by using Theorem 1.

B. Phase assignment Φ_{κ_j} for given Φ_{τ_i}

In this subsection, we present our approach to assign the communication phases Φ_{κ_j} for each κ_j scheduled in a NoC under given Φ_{τ_i}. Similar to the task order Ω_τ, we define a communication phase assignment order Ω_κ for the communication tasks. Since the phases of the computational tasks are already specified, we can define the urgency of the communication tasks by their relative deadlines defined in (4), i.e., Ω_κ starting with the tasks with the shortest relative deadline.

Our greedy approach (Algorithm 1) assigns the communication phases Φ_{κ_j} according to the defined order Ω_κ. Therefore, we first try to set the phase Φ_{κ_j} to B_{m_j}, which represents the completion of the source task τ_{SRC_j}.

By using Theorem 2, if the current assignment of Φ_{κ_j} is not feasible, we resolve the conflict by delaying this communication task for a certain amount of time such that this conflict is resolved, shown in Algorithm 2.

C. Time complexity of our heuristic

We now analyze the complexity of the workflow presented in Figure 7 for one iteration of phase assignments for $\mathbf{C}$ and $\mathbf{T}$. The time complexity of determining the conflict matrix $\mathcal{M}$ is $O(|\mathbf{C}|^2 \cdot |\mathcal{S}|)$. Thus, the time complexity for Algorithm 2 is $O(1)$. The time complexity of the phase assignment ordering is $O(|\mathbf{C}|^2)$, because for each κ_j, we need to find the lowest D_{κ_j}. Each Ω_{κ_a} assignment takes at most $|\mathbf{C}|$ iterations.

With regard to Algorithm 1, one while-loop has time complexity $O(|\mathbf{C}|)$ for detecting a conflict with another communication task. The while-loop is executed at most ν times, with $\nu = |\mathbf{C}|(\lceil \frac{P_{\kappa_\ell}}{P_{\kappa_0}} \rceil + \lceil \frac{P_{\kappa_\ell}}{P_{\kappa_1}} \rceil + \ldots + \lceil \frac{P_{\kappa_\ell}}{P_{\kappa_{(\ell-1)}}} \rceil)$, because each κ_ℓ collides at most once with another job, in which $\lceil \frac{P_{\kappa_\ell}}{P_{\kappa_j}} \rceil$ jobs for each κ_j exists. Thus, $\nu \leq \lceil \frac{P_{\max}}{P_{\min}} \rceil |\mathbf{C}|^2$. Therefore, the time complexity of Algorithm 1 is $O(\lceil \frac{P_{max}}{P_{min}} \rceil |\mathbf{C}|^3 |\mathcal{S}|)$. Similarly, the time complexity for the computational phase assignment is $O(\lceil \frac{P_{max}}{P_{min}} \rceil |\mathbf{T}|^3)$.

Hence, the time complexity of our workflow for assigning the phases of $\mathbf{C}$ and $\mathbf{T}$ is $O(n \lceil \frac{P_{max}}{P_{min}} \rceil (|\mathbf{T}|^3 + |\mathbf{C}|^3 |\mathcal{S}|))$, where n is the number of iterations defined in the workflow (Figure 7).

D. Phase assignment by an SMT solver

Another approach is to solve the feasibility test with the Satisfiability Modulo Theories (SMT) solver, which is described

Algorithm 1 Heuristic algorithm for assigning the phases of the communication task set $\mathbf{C}$

Input: $\mathbf{C}$, $\mathbf{T}$, Ω_κ, and platform;
Output: Phases Φ_{κ_j}, packet delay ρ_j and feasibility;
1: $\forall \kappa_j \; B_{m_j} \leftarrow \Phi_{\tau_{\text{SRC}_j}} + W_{\tau_{\text{SRC}_j}}$;
2: Calculate a route conflict matrix $\mathcal{M}$ for given platform;
3: **for** $\ell = 1, \cdots, |\mathbf{C}|$ stepped by 1 according to Ω_κ **do**
4: $\quad \Phi_{\kappa_\ell} \leftarrow B_{m_\ell}$;
5: $\quad$ feasible $\leftarrow$ **false**;
6: $\quad$ **while** $(\Phi_{\kappa_\ell} < P_{\kappa_\ell})$ **and** (feasible = **false**) **do**
7: $\quad\quad$ feasible $\leftarrow$ **true**;
8: $\quad\quad$ **for** each κ_j with $j < \ell$ **do**
9: $\quad\quad\quad \delta_\ell \leftarrow$ ResolveCommConflict(κ_ℓ,κ_j,$\mathcal{M}$);
10: $\quad\quad\quad$ **if** $\delta_\ell \neq 0$ **then**
11: $\quad\quad\quad\quad \Phi_{\kappa_\ell} \leftarrow \Phi_{\kappa_\ell} + \delta_\ell$;
12: $\quad\quad\quad\quad$ feasible $\leftarrow$ **false**;
13: $\quad$ **if** (feasible = **false**) **then**
14: $\quad\quad$ **return** "not feasible";
15: $\quad \rho_\ell \leftarrow \Phi_{\kappa_\ell} - B_{m_j}$;
16: **return** "feasible";

Algorithm 2 ResolveCommConflict() - algorithm

Input: κ_ℓ, κ_j, $\mathcal{M}$;
Output: time shift t;
1: $t \leftarrow 0$;
2: **if** $\exists$ conflict between route R_ℓ and R_j via $\mathcal{M}$ **then**
3: $\quad$ Calculate $gcd_{\ell,j}$ of P_{κ_ℓ} and P_{κ_j};
4: $\quad \Psi_a \leftarrow (\Phi_{\kappa_\ell} + \mathcal{M}(\ell, j) \cdot (d_\mathcal{L} + d_R)) \bmod gcd_{\ell,j}$;
5: $\quad \Psi_b \leftarrow (\Phi_{\kappa_j} + \mathcal{M}(j, \ell) \cdot (d_\mathcal{L} + d_R)) \bmod gcd_{\ell,j}$;
6: $\quad$ **if** $(\Psi_a < \Psi_b)$ **then**
7: $\quad\quad$ **if** $\Psi_b < \Psi_a + \overline{W}_{\kappa_a}$ **then**
8: $\quad\quad\quad t \leftarrow \Psi_a + \overline{W}_{\kappa_a} - \Psi_b$
9: $\quad\quad$ **else if** $\Psi_a + gcd_{\ell,j} < \Psi_b + \overline{W}_{\kappa_b}$ **then**
10: $\quad\quad\quad t \leftarrow \Psi_a + gcd_{\ell,j} + \overline{W}_{\kappa_a} - \Psi_b$;
11: $\quad$ **else**
12: $\quad\quad$ **if** $\Psi_a < \Psi_b + \overline{W}_{\kappa_b}$ **then**
13: $\quad\quad\quad t \leftarrow \Psi_a + \overline{W}_{\kappa_a} - \Psi_b$;
14: $\quad\quad$ **else if** $\Psi_b + gcd_{\ell,j} < \Psi_a + \overline{W}_{\kappa_a}$ **then**
15: $\quad\quad\quad t \leftarrow \Psi_a + \overline{W}_{\kappa_a} - \Psi_b - gcd_{\ell,j}$;
16: **return** t;

in this section. The problem to determine a feasible set of computational and communication phases can be formulated into a set of equations, called SMT problem. In the Appendix is the detailed Algorithm 3, which represents the formulated SMT problem for our task model.

In general, there exist two different ways to formulate the SMT problem. On one hand, the problem can be formulated on the task level by using all equations from the feasibility test, see Section III. On the other hand, the schedule can be unfolded to the job level. Preliminary experiments point out, that the job level formulation of the SMT problem requires significally less runtime than the task-level formulation. Therefore, Algorithm 3 in Appendix shows only the job level formulation, as a base-line for comparison, though both formulations were implemented in the experiments.

V. Experiments

In this section, we present experiments to explore the capabilities of our phase assignment algorithm getting feasible results.

A. Experimental set up

We generate randomized synthetic system sets based on our system model with typical industrial application characteristics, like an engine control system to get reliable results with harmonic periods and a large number of tasks, i.e. 100–1000 tasks with appropriate communications. Note that arbitrary periods would lead to a lower system utilization and the sending and receiving jobs becomes inconsistent.

The generator performs the following steps for one system including a platform with computational and communication task sets.

1) Platform generator: Build a 3×3 platform like in Figure 2 with all cores, switches and links.
2) Random computational task set: Generate 100–1000 task with harmonic periods and no heavy tasks.
3) Random communication task set: Generate $2 \times |\mathbf{T}|$ to $4 \times |\mathbf{T}|$ communication tasks that connects τ_i with each other. Each κ_j has a chance of 20% to get a precedence constraint, if not specified.
4) Task to core mapping: Worst-fit bin packing, minimizing the communication utilization by placing tasks to the same core.

In each experiment and particular parameter setting, 100 randomized system sets are generated and the feasibility is evaluated for different algorithms. We calculate an upper bound for the feasibility, which is done based on utilization tests on an individual core under the task mapping.

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. In order to get a solution from the SMT solver, the size of a task set in the experiments is set to $|\mathbf{T}| = 100$ tasks with $|\mathbf{C}| = 300$, whereas other experiments use $|\mathbf{T}| = 250$ tasks with $|\mathbf{C}| = 750$. Note that the SMT solver failed for larger system sets due to its memory requirement. Our heuristic algorithm returns also a solution in larger sets, shown in Section V-B5. The complexity of the SMT solver based solution takes exponential time complexity in terms of tasks, whereas our heuristic is executed in pseudo-polynomial time complexity.

B. Experimental results

This section presents several experimental results, showing the capabilities of our approach.

1) Reachable utilization: In this experiment, we present the reachable system utilization of the TTCP approach in conjunction with our heuristic algorithm. The results are shown in Figure 9a and 9b with $|\mathbf{T}| = 100$ and $|\mathbf{C}| = 300$.

On the abscissa is the core utilization U_τ or communication utilization U_κ with $U_\tau = \sum_{\forall \tau_i} (\frac{W_{\tau_i}}{P_{\tau_i}})$ and $U_\kappa = \sum_{\forall \kappa_j} (\frac{W_{\kappa_j}}{P_{\kappa_j}})$, which represents the utilization summation of the computational or communication tasks, respectively. We compare our heuristic against the SMT solver based approach, which takes much more time to compute. As both figures show, the TTCP scheduling approach is capable to effectively utilize the cores and the NoC for larger applications. Note that the single core utilization bound is $U_\tau \leq 1$ and the single shared bus utilization bound is $U_\kappa \leq 1$. In the experiment *run time and solver feasibility* (Section V-B5), we focus more on the comparison between the SMT solver and our heuristic.

Additionally, we run another experiment with $|\mathbf{T}| = 1000$ and $|\mathbf{C}| = 3000$, where the solver was unable to provide a solution. Here, the heuristic algorithm returns a feasible solution with $U_\kappa = 60\%$ for 50% of the tested task sets in around $12s$. A communication utilization of $U_\kappa = 60\%$ means that a core sends on average in 60% of its time a packet to other cores without interfering other packets or violating the real-time constraints.

2) NoC scalability: This experiment examines different NoC platform sizes. The structure and the policies of the NoC are unchanged except the platform size from 4 to 64 cores. We evaluate the computation and the communication utilization for different NoCs, which are shown in Figure 10a, 10b. We can effectively utilize larger NoCs with our method. In the experiments the number of computational tasks is set to $|\mathbf{T}| = 250$. With a platform of 36 cores the task set reaches its limits, so more cores do not enable more computation nor more communication resources. Additionally, the upper bound drops down $U_\tau > 25$ under less than 30%. Hence, no feasible schedule could be found regardless of the NoC size. The reason is that the precedence constraints reduce the parallelizability, which is examined in Section V-B4.

3) Communication phase assignment order: For the communication phase assignment, we define an order Ω_κ based on deadline monotonic (DM) in our heuristic algorithm. There exist other possibilities to define the Ω_κ, which are evaluated in this experiment. The lower periods first (LPF) heuristic assigns the order based on the period. If the periods are equal, then the lower B_{m_j} is assigned first.

The DAG heuristic analyzes the DAG to define Ω_κ. Each path in the DAG gets a score such that κ_j on a higher scored path is assigned before in Ω_κ. For the understanding of the importance of the communication ordering, we add

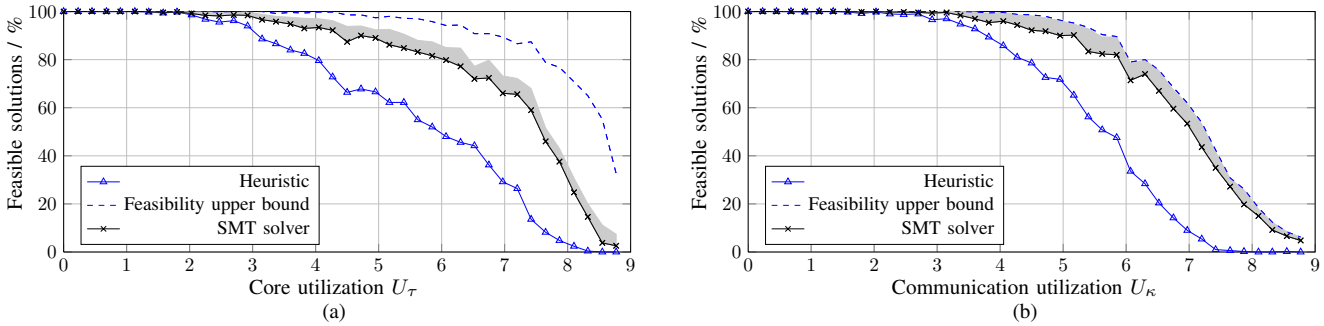

Figure 9: Reachable utilization: Different computational (a) and communication (b) utilization on a 3×3 NoC

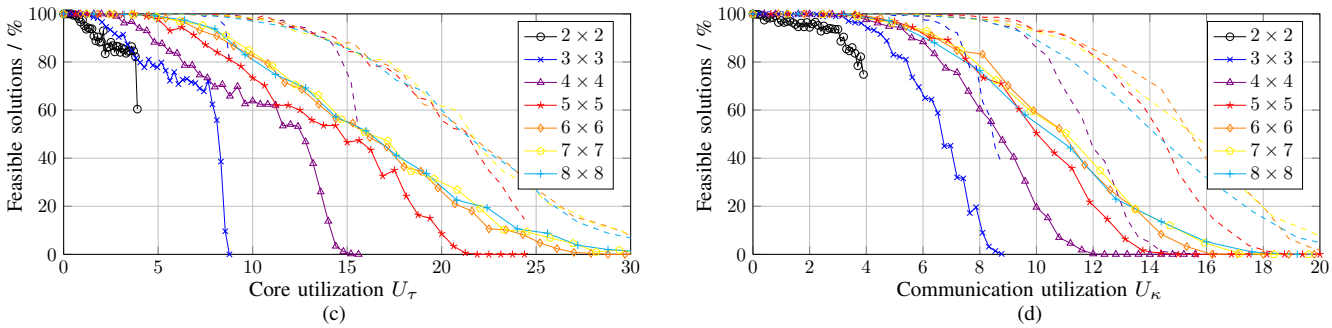

Figure 10: NoC scalability: Computational (a) or communication (b) utilization for different sized mesh NoCs are presented here. The dashed lines represent corresponding upper bound.

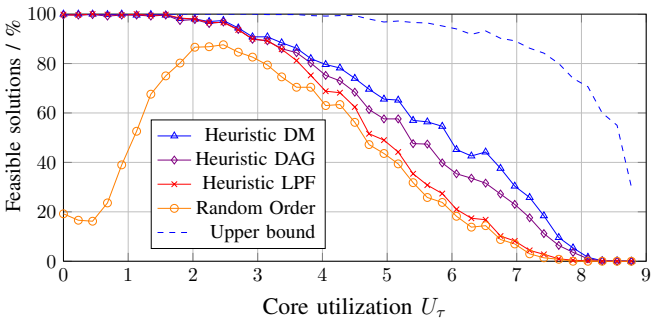

Figure 11: Communication phase assignment orders: Different methods to define the order Ω_κ are compared to each other.

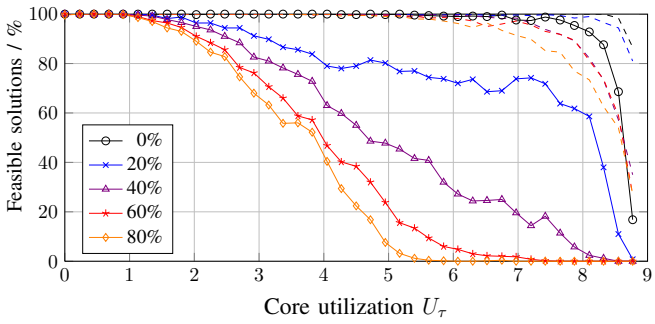

Figure 12: Precedence rate influence: Different precedence rates significally determine the reachable core utilization U_τ.

pure randomized assignments. Figure 11 shows that the DM heuristic for the phase assignment ordering outperforms the other approaches.

4) Precedence rate influence: In this experiment, we evaluate the influence of the precedence constraints given by the application. Figure 12 shows that the reachable utilization strongly depends on the precedence rate. Therefore, an application full of precedence constraints is hard to be parallelized. Furthermore, applications with no precedence relations, but communication relations can be parallelized quite well.

5) Run time and solver feasibility: Based on Figure 9a, it seems obvious that the SMT solver is better to determine the phases for the computational and communication tasks. On one hand, the maximum feasibility of the SMT solver is higher, but not much higher than our heuristic algorithm. On the other hand, the time complexity of our heuristic is more affordable, which makes it capable for handling large-scaled applications. In Figure 13 and 14, we evaluate the SMT solver and our heuristic under different sized sets. The experiment runs on a 3×3 NoC with $|\mathbf{C}| = 3|\mathbf{T}|$ such that both task sets increase their size. The computational utilization is set to $U_\tau = 4.5 = 50\% U_{max}$ and the communication utilization is set to $U_\kappa = 1$. With $|\mathbf{T}| = 100$ the SMT solver was able to solve the given problem, but later the solver returns no feasible solution, because the problem size exceeds its limits. For instance, with $|\mathbf{T}| = 1000$, the SMT problem would result in approximately $8 \cdot 10^9$ *ASSERT* statements for describing the problem. Considering typical industrial cases, of 100–1000 tasks only our heuristic is able to provide a feasible solution.

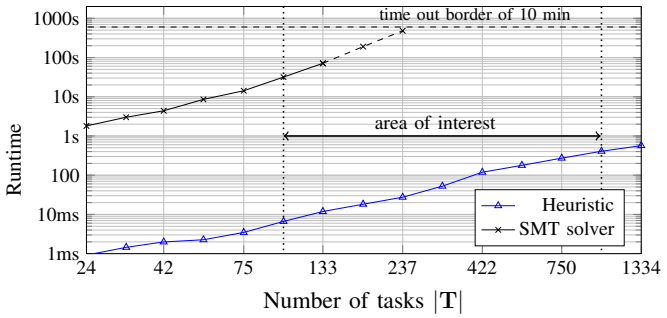

Figure 13: Run time and solver feasibility: Double logarithmic plot of the runtime for different phase assignment methods.

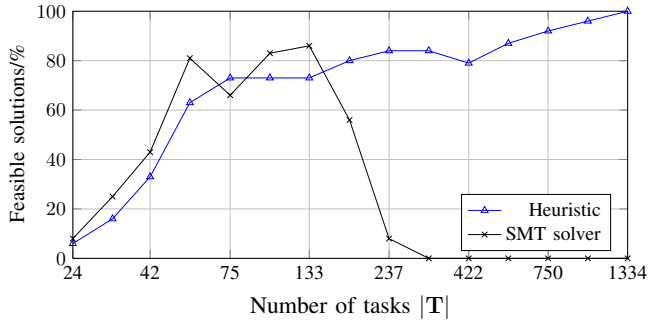

Figure 14: Run time and solver feasibility: Semi-logarithmic plot of the feasibility of assignment methods. With more than $|T| > 133$ tasks, the solver reached its limits depending on the hyper-period, such that no solution can be found.

VI. Conclusion

In this paper, we show that a tangled task model, which is typical for industrial applications, can be scheduled by a time-triggered constant phase (TTCP) scheduler on a multicore system with NoCs. This approach is capable to highly utilize both the cores and the NoC for typical industrial applications with a large amount of tasks. We provide a feasibility analysis to verify the real-time constraints by a given set of time-triggered parameters. Furthermore, we present an iterative algorithm to determine a feasible set of the TTCP scheduling parameters running in pseudo-polynomial time complexity for our tangled task model. Experiments show the maximum possible utilization of large and tangled applications. Additionally, we confirm that the task precedence constraints limit the parallelizability of typical industrial applications.

References

[1] A. Biewer, J. Gladigau, and C. Haubelt. A novel model for system-level decision making with combined ASP and SMT solving. In *DATE*, 2014.

[2] S. S. Craciunas and R. S. Oliver. Smt-based task-and network-level static schedule generation for time-triggered networked systems. In *RTNS*, page 45, 2014.

[3] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, 2008.

[4] M. Freier and J.-J. Chen. Time triggered scheduling analysis for real-time applications on multicore platforms. In *RTSS workshop on REACTION*, pages 48–53, 2014.

[5] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE D & T*, 2005.

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

[7] T. Kuroda. Low-power, high-speed cmos vlsi design. In *Computer Design: VLSI in Computers and Processors*, pages 310–315, 2002.

[8] Z. Lu and A. Jantsch. TDM virtual-circuit configuration for network-on-chip. *VLSI Systems*, '08.

[9] 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.

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

[11] I. Miro Panades, A. Greiner, and A. Sheibanyrad. A low cost network-on-chip with guaranteed service well suited to the gals approach. In *NanoNet*, 2006.

[12] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integration, the VLSI Journal*, 2004.

[13] P. Munk, M. Freier, J. Richling, and J.-J. Chen. Dynamic guaranteed service communication on best-effort networks-on-chip. In *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 2015.

[14] C. Paukovits and H. Kopetz. Concepts of switching in the time-triggered network-on-chip. In *RTCSA*, 2008.

[15] F. Reimann, M. Lukasiewycz, M. Glass, C. Haubelt, and J. Teich. Symbolic system synthesis in the presence of stringent real-time constraints. DAC, 2011.

[16] F. Sagstetter, S. Andalam, P. Waszecki, M. Lukasiewycz, H. Stähle, S. Chakraborty, and A. Knoll. Schedule integration framework for time-triggered automotive architectures. DAC, 2014.

[17] K. Schild and J. Würtz. Scheduling of time-triggered real-time systems. *Constraints*, 2000.

Appendix

Algorithm 3 SMT formulating of a TTCP scheduling problem

Input: computational task set $\mathbf{T}$, communication task set $\mathbf{C}$;
Output: SMT problem;

for each $\tau_i \in \mathbf{T}$ **do**
  DEFINE t(i,0);
  ASSERT t(i,0)$\geq$ 0;
  for each predecessor τ_p of τ_i **do**
    if $c_i = c_p$ **then**
      ASSERT t(i,0)$\geq$t(p,0);
    else
      $\forall\ \kappa_\ell \mid \kappa_{\text{DST}_\ell} = \tau_i$;
      ASSERT t(i,0)$\geq$c(ℓ)+$\overline{W}_{\kappa_\ell}$;
  ASSERT t(i,0)$<$ $D_{\tau_i} - W_{\tau_i}$;
  for each job $J_{\tau_i,k}$, $k = 1, \cdots \frac{\text{h}}{P_{\tau_i}}$ stepped by 1 **do**
    DEFINE t(i,k);
    ASSERT t(i,k) = t(i,0)$\cdot$k$\cdot P_{\tau_i}$;
for each $\kappa_j \in \mathbf{C}$ **do**
  DEFINE c(j,0)
  ASSERT c(j,0)$\geq$t(SRC$_j$,0);
  if κ_j precedence **then**
    ASSERT c(j,0)$\leq$t(DST$_j$,0)$-\overline{W}_{\kappa_j}$;
  else
    if $P_{\tau_{\text{SRC}_j}} \geq P_{\tau_{\text{DST}_j}}$ **then**
      ASSERT c(j,0)$\geq$t(DST$_j$,0)+$P_{\tau_{\text{SRC}_j}} - \overline{W}_{\kappa_j}$;
    else
      ASSERT (c(j,0)$\leq$ t(DST$_j$,0) $-\overline{W}_{\kappa_j}$ **and** t(DST$_j$,0) $\geq P_{\tau_{\text{SRC}_j}}$) **or** (c(j,0)$\leq$ t(DST$_j$,0)+$P_{\tau_{\text{SRC}_j}} - \overline{W}_{\kappa_j}$ **and** t(DST$_j$,0) $< P_{\tau_{\text{SRC}_j}}$);
    ASSERT c(j,0)$\geq$t(SRC$_j$,0);
  for each packet $p_{j,l}$, $l = 1, \cdots \frac{\text{h}}{P_{\kappa_j}}$ stepped by 1 **do**
    DEFINE c(j,l);
    ASSERT c(j,l) = c(j,0)$\cdot$l$\cdot P_{\kappa_j}$;
for $i = 1, \cdots, |t|$ stepped by 1 **do**
  for $k = 1, \cdots, |t|$ stepped by 1 ($k \neq i$) **do**
    ASSERT (t(i) $\geq$ t(k) + W_{τ_k}) **or** (t(k) $\geq$ t(i) + W_{τ_i});
for $j = 1, \cdots, |c|$ stepped by 1 **do**
  for $l = 1, \cdots, |c|$ stepped by 1 ($l \neq j$) **do**
    ASSERT (c(j) $\geq$ c(l) + W_{κ_l}) **or** (c(l) $\geq$ c(j) + W_{κ_j});