Universität
Augsburg
University

Real-Time Analysis of MPI Programs for NoC-based Many-Cores using Time Division Multiplexing

Dissertation

for the degree of
Doctor of Engineering (Dr.-Ing.)

submitted to the
Department of Computer Science
University of Augsburg

presented by

Alexander Stegmeier

in 2019

REAL-TIME ANALYSIS OF MPI PROGRAMS FOR NOC-BASED MANY-CORES USING TIME DIVISION MULTIPLEXING

Alexander Stegmeier, M. Sc.

Examiner:	**Prof. Dr. Theo Ungerer**, Department of Computer Science University of Augsburg
Co-examiner:	**Prof. Dr. Sebastian Altmeyer**, Department of Computer Science University of Augsburg

Date of oral examination: March 9th, 2020

Abstract

Worst-case execution time (WCET) analysis is crucial for designing hard real-time systems. While the WCET of tasks in a single core system can be upper bounded in isolation, the tasks in a many-core system are subject to shared memory interferences which impose high overestimation of the WCET bounds. However, many-core-based massively parallel applications will enter the area of real-time systems in the years ahead. Explicit message-passing and a clear separation of computation and communication facilitates WCET analysis for those programs.

A standard programming model for message-based communication is the message passing interface (MPI). It provides an application independent interface for different standard communication operations (e.g. broadcast, gather, ...). Thereby, it uses efficient communication patterns with deterministic behaviour. In applying these known structures, we target to provide a WCET analysis for communication that is reusable for different applications if the communication is executed on the same underlying platform. Hence, the analysis must be performed once per hardware platform and can be reused afterwards with only adapting several parameters such as the number of nodes participating in that communication. Typically, the processing elements of many-core platforms are connected via a Network-on-Chip (NoC) and apply techniques such as time-division multiplexing (TDM) to provide guaranteed services for the network. Hence, the hardware and the applied technique for guaranteed service needs to facilitate this reusability of the analysis as well.

In this work we review different general-purpose TDM schedules that enable a WCET approximation independent of the placement of tasks on processing elements of a many-core which uses a NoC with torus topology. Furthermore, we provide two new schedules that show a similar performance as the state-of-the-art schedules but additionally serve situations where the presented state-of-the-art schedules perform poorly. Based on these schedules a procedure for the WCET analysis of the communication patterns used in MPI is proposed. Finally, we show how to apply the results of the analysis to calculate the WCET upper bound for a complete MPI program.

Detailed insights in the performance of the applied TDM schedules are provided by comparing the schedules to each other in terms of timing. Additionally, we discuss the exhibited timing of the general-purpose schedules compared to a state-of-the-art application specific TDM schedule to put in relation both types of schedules. We apply the proposed procedure to several standard types of communication provided in MPI and compare different patterns that are used to implement a specific communication. Our evaluation investigates the communications' building blocks of the timing bounds and shows the tremendous impact of choosing the appropriate communication pattern. Finally, a case study demonstrates the application of the presented procedure to a complete MPI program.

With the method proposed in this work it is possible to perform a reusable WCET timing analysis for the communication in a NoC that is independent of the placement of tasks on the chip. Moreover, as the applied schedules are not optimized for a specific application but can be used for all applications in the same way, there are only marginal changes in the timing of the communication when the software is adapted or updated. Thus, there is no need to perform the timing analysis from scratch in such cases.

Danksagung

An dieser Stelle bedanke ich mich in erster Linie bei meinem Doktorvater Prof. Dr. Theo Ungerer für die hervorragende Betreuung beim Verfassen der Dissertation. Die Unterstützung, die ich von ihm als Teil seines Lehrstuhls erfuhr, trug in großem Maße zur Findung des Themas sowie zur Entwicklung der Konzepte meiner Arbeit bei. Seine zahlreichen Hinweise und konstruktive Kritik haben wesentlich zum Gelingen meines Promotionsvorhabens beigetragen. Außerdem bedanke ich mich dafür, dass er es mir ermöglicht hat an verschiedenen nationalen und internationalen Konferenzen und Workshops teilzunehmen. Ebenfalls bedanke ich mich bei Prof. Dr. Sebastian Altmeyer für die Begutachtung dieser Arbeit.

Ich danke herzlich allen Kollegen des Lehrstuhls für Systemnahe Informatik und Kommunikationssysteme für die vielen fruchtbaren Diskussionen. Besonderer Dank geht dabei an Jörg Mische, dessen Idee zum Reduced Complexity Many-Core (RC/MC) die Grundlage für das in dieser Arbeit behandelte Thema bildet. Ihm sowie Martin Frieb sei nochmals gesondert für ihre langjährige enge Zusammenarbeit gedankt. Sie sind mir während meiner gesamten Promotionsphase in zahlreichen Situationen mit Rat zur Seite gestanden. Außerdem danke ich Christian Bradatsch für seine Unterstützung zu Beginn meiner Anstellung am Lehrstuhl.

Ich danke besonders meiner Familie für ihre Unterstützung. Meinen Eltern Renate und Manfred sowie meiner Schwester Kerstin danke ich, da sie mir auf meinem Weg jederzeit beigestanden und mich unterstützt haben. Ganz besonderer Dank gebührt meiner Frau Julia für ihre Geduld und ihrem Beistand, den ich während dem Verfassen der Dissertation von ihr erfahren durfte. Meinen beiden Kindern danke ich für die gute Laune, die sie tagtäglich verbreiten und damit mein Herz an jedem Tag mit Freude erfüllen.

Augsburg im Dezember 2019 *Alexander Stegmeier*

Contents

Abstract iv

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Overview . . . 3
- 1.3 Contribution . . . 5
- 1.4 Outline . . . 5

2 Background and System Model 7
- 2.1 Hardware Architecture and Routing Basics . . . 7
  - 2.1.1 Definition of Hardware Architecture . . . 10
  - 2.1.2 Definition of Routing Behaviour . . . 15
- 2.2 Time Division Multiple Access . . . 18
  - 2.2.1 Application Specific TDM Schedules . . . 26
  - 2.2.2 General-Purpose TDM Schedules . . . 27
- 2.3 Message Passing Interface . . . 31
  - 2.3.1 Point-to-Point Communication . . . 35
  - 2.3.2 Collective Communication . . . 38
- 2.4 Real-Time Analysis . . . 44
  - 2.4.1 Static Analysis . . . 46
  - 2.4.2 Measurement-based Analysis . . . 47

3 Related Work 49
- 3.1 Guaranteed Service for Network-on-Chips . . . 49
- 3.2 Message Passing Interface . . . 52

4 General-Purpose TDM Schedules for Message-Passing 55
- 4.1 Formalization and Construction of TDM Schedules . . . 56
  - 4.1.1 State-of-the-Art General-Purpose TDM Schedules . . . 58
  - 4.1.2 Composition of General-Purpose TDM Schedules . . . 90
- 4.2 Formal Timing Analysis . . . 99
  - 4.2.1 General Approach . . . 99
  - 4.2.2 Application to General-Purpose TDM Schedules . . . 101
- 4.3 Discussion . . . 105
  - 4.3.1 Comparison of General-Purpose TDM Schedules . . . 105

4.3.2 Comparison of Application Specific versus General-Purpose TDM Schedules . . . 109

5 Timing Analysis of MPI-based Communication Operations 113
5.1 Assumptions and Workflow for Communication Operations . . . 114
5.2 Assembling Phase and Formalism of Timing Analysis . . . 117
5.2.1 Point-to-Point Communication . . . 118
5.2.2 Collective Communication . . . 126
5.3 Evaluation . . . 135
5.3.1 Point-to-Point Communication . . . 138
5.3.2 Collective Communication based on a Central Process . . . 139
5.3.3 Collective Communication based on Uniform Data Exchange . . . 142
5.3.4 Influence of TDM Schedules . . . 145

6 Timing Analysis of MPI-based Parallel Programs 161
6.1 Assumptions and Workflow for Parallel Programs . . . 161
6.2 Assembling Phase . . . 164
6.3 Case Study: Parallel LU Calculation . . . 168
6.3.1 Algorithm Structure . . . 168
6.3.2 Real-Time Analysis . . . 172
6.3.3 Evaluation . . . 176

7 Conclusion and Future Work 189
7.1 Conclusion . . . 189
7.2 Future Work . . . 191

Bibliography 195

1 Introduction

1.1 Motivation

The demands in computational power of the applications for embedded systems increase more and more in recent years. This trend also influences the area of systems that require real-time capabilities in a broad range of different fields of applications, as e.g. industrial machinery, automotive and avionic systems. Industrial robots for example may target to work directly together with a human worker. Therefore, the system must recognize the position of the human and its complete body to prevent the human to be harmed by the movement or any other action of the machinery and also to effectively work together on the same component [LRS14]. In the avionics sector the complexity of the flight control mechanisms increase in order to save fuel or to guarantee a save flight path for modern designs of airplanes [Bie+12]. The development of autonomously driving cars seems to be the driving force for real-time systems with a high demand in performance in the automotive sector [Kuw+09; Wu+17]. In all domains the system must recognize different situations and set control commands accordingly within a given period of time.

Due to described increase in the amount of computational power the applications tend to be executed more and more on multi-cores rather than on single-core processors. Following the trend of applying more parallelism we even envision the utilization of many-core processors with a large number of cores for that kind of applications. However, this leads to a vast amount of issues to be solved, especially in *hard real-time (HRT)* systems.

Problems already arise in timing analysis of parallel applications running on a small number of cores [Roc+10; Fri+16a]. In particular shared memory multi-cores suffer from timing interference when real-time programs running on different cores access the shared memory. Thereby, the problems grow with the number of cores utilized due to the increasing possibility of interferences [Fri+16a]. Thus, solutions that alleviate the problems with inter-

ferences will be definitely in demand. A possible way to bound and reduce the amount of potential interferences is the restriction of communication to explicit message-passing since it limits the number of cores possibly entering a shared resource concurrently [Lis12]. Deterministically known communication partners as well as a clear separation of computation and communication may reduce interferences and facilitate the analysis likewise [Lis12].

We focus on many-cores that provide a certain number of nodes that are isolated against each other and are only connected via a dedicated interconnection network that is called *network-on-chip (NoC)*. With such a hardware the interaction between different nodes is carried out via explicit message passing. The standard programming model for this kind of communication is the *message passing interface (MPI)* [MPI15]. However, this interface is normally not used in embedded systems but for *high performance computing (HPC)*. Nevertheless, since it is the standard interface for explicit communication in parallel programs, it provides a set of different communications that are relevant for parallel programs. As the standard is built by an open community, new communication operations are only included after exhaustive discussions and when they have proven to be relevant for developing parallel software [MPI15, p. 1f.]. Hence, we see the communication included in the MPI standard as generally relevant for parallel software. According to the future applications for embedded systems like for example autonomous driving we envision that there will be a transition of typical HPC applications to the field of embedded computing. Therefore, the standard shows the relevant types of communication for future parallel applications in embedded systems and may also be used to implement the communication for embedded systems in future.

Several open source implementations exist that follow the MPI standard. These frameworks target an efficient and performant communication on arbitrary homogeneous and heterogeneous platforms. Internally they use sophisticated algorithms to provide communication as efficiently as possible in terms of bandwidth and latency. These algorithms may provide a good basis for communication in real-time systems as well.

Altogether the investigation of MPI programs in terms of HRT seems to provide relevant insights for future parallel embedded applications where worst-case timing constraints are an issue. Thus, in this thesis we focus on a *worst-case execution time (WCET)* analysis of MPI programs.

Many physical components (e.g. engines in the automotive sector) are controlled with legacy software that grows over years and is used to control multiple generations of components. In future applications the reuse of legacy software will be an issue as well [Ung+16]. Additionally, there is the objective to compose multiple applications to run on one processing chip to reduce costs (energy and purchase). Hence, it is obvious that in the embedded domain software is often adapted, updated or reused within its life cycle. Also, the placement of applications on a chip may change during their life cycles. Therefore, we target an analysis method that takes such needs into account. Thus, the concepts for our timing analysis attempt to comply the following characteristics.

Our main objective is to provide an efficient estimation of WCET bounds for parallel applications. For that we provide analysis components that can be (partly) reused for future analyses on the same target platform. Furthermore, changes in already analysed software shall cause only a small effort for re-running the analysis. Another characteristic

that facilitates the mentioned issues is a clear isolation of the traffic on the NoC in terms of timing. Thereby, we target a policy for data communication that does not cause a change in the timing behaviour of an application if the other workloads that are executed on the same chip change over time. Moreover, the timing of a parallel application shall not change even if the mapping of the application's tasks on the chip is changed.

To achieve all these aims we provide the analysis in several abstraction levels. Thereby, in each level the worst-case timing upper bounds of the according components are provided. The bounds of these components can be applied in the timing analysis of the next higher level as a single module. For example the end-to-end timing of one such module is returned depending on a set of parameters that are chosen according to the situation of the currently analysed abstraction level. The analysis of the highest abstraction level provides the end-to-end worst-case timing bound of the complete parallel program.

1.2 Overview

The end-to-end WCET upper bound of a message-passing based program comprises the timing of code executed and the timing for the delivery of the message. Hence, no trivial way reveals the overall timing neither of a MPI program nor a single MPI operation. Thus, a special procedure for the analysis must be applied.

In this section we present a short overview of our procedure of analysing a parallel MPI program. Hence, we give an overview of the abstraction levels that must be considered for the timing analysis of a complete parallel program. The abstraction levels are presented top down, starting at a parallel program and finishing at the mechanism to provide guaranteed services for the NoC in terms of timing. Altogether, the structure of a parallel message-passing based MPI program can be divided in three abstraction levels.

On the highest level it is sufficient for a timing analysis to look at a parallel program like it is done by Lisper [Lis12]. He envisions high-performance tasks as a parallel program running on several cores. Thereby, the program consists of core-local computation phases which are connected via parallel inter-process communication and synchronisation phases (see Figure 1.1). The communication phases can be seen as functions that are called by all participating processes and implement the concrete data exchange and synchronisation between the processes. The internal delivery of messages in those functions marks the second abstraction level. Typically, in one communication phase the delivery of multiple messages between the participating processes is conducted to perform the desired communication in an efficient way. Finally, the lowest abstraction level provides worst-case timing guarantees of single flits of a message the traversal through the NoC.

According to these abstraction levels we provide the results of the timing analysis of each level that can be used by the next higher level. The lowest level provides the time a flit must wait until it is released to the NoC and the time a flits needs to actually traverse the network. These two times are used by the level for analysing the communication phases of a program. This level in turn provides a formalism that returns the timing bounds for each communication operation. Thereby, some parameters must be supplied as input to the formalism to characterise the concrete instance of the communication. On application level these input parameters are used to exhibit the timing of a communication with the

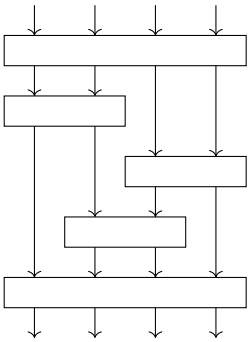


Figure 1.1: Schematic illustration of a parallel example program. The vertical lines indicate processes executing process local computations. The rectangles mark phases of communication or synchronisation between the processes.

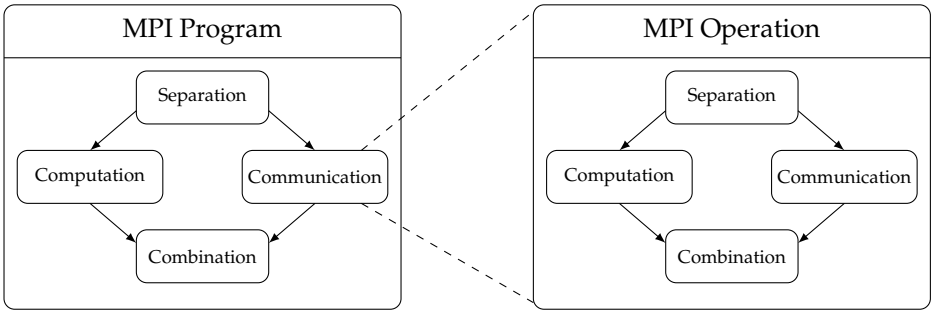


Figure 1.2: Workflow for the two higher abstraction levels of the timing analysis of parallel MPI programs. The lower level focuses on MPI operations while the higher one targets the analysis of the parallel program and relies on the results of the lower level.

specific characteristics for the given application. Thereby, the focus is on embedding the MPI operations' timing bounds in the timing of the application software to reveal an end-to-end timing bound for the overall program. On this highest level the end-to-end worst-case timing bound for a program is calculated according to a given input set and the program's parallel execution structure (cf. Figure 1.1).

Though the procedure for the analysis of the two higher levels differ in some issues in detail, the overall workflow is very similar (see Figure 1.2). In both parts there are four distinct processing stages [Fri+16b], which must be performed in correct order. At first, the code under analysis is examined to identify and separate parts which cause pure process-local computation and actual interaction with other processes regarding communication, respectively. The next two assignments may be performed concurrently. One is the estimation of WCET upper bounds for all parts that contain only pure process-local computation. The other piece of work focuses on the calculation of timing bounds for components where interaction between several cores take place. Finally, a combination of all revealed timing upper bounds lead to an end-to-end WCET upper bound. Please note that there is a precedence constraint between analysing operations and programs. To analyse a given MPI program, all applied MPI operations must be analysed beforehand.

1.3 Contribution

We provide a system model to describe the hardware and routing behaviour as well as issues about *time-division multiplexing (TDM)* and the MPI. Furthermore, we investigate techniques that allow us to state timing guarantees for the communication that is independent of task placement. Thereby, a formal description of existing general-purpose *time-division multiple access (TDMA)* schedules is provided including the transition from the hardware view of the original publication to the view on paths that is more common in the area of TDMA theory. The presentation of each presented TDM schedule also includes a formal proof of its conflict freedom, which was still missing in literature. Based on those concepts two new schedules are developed that overcome poor performance behaviour of the original schedules for some possible communication configurations. Both schedules are described in the same formal presentation as the former schedules and proofs about their conflict-freedom are provided. In addition to the presentation of the TDM schedules a formal timing analysis suitable for all TDM schedules is provided that transforms the schedule configuration to two time values for the timing analysis on a higher abstraction level. Afterwards, this general approach is applied to the presented TDM schedules to obtain the concrete timing of each schedule. Finally, the worst-case guarantees of all schedules are discussed and compared against each other and against state-of-the-art application specific schedules.

We provide an analysis of different MPI operations. At first, we state the assumptions made for this analysis and present the workflow of the analysis. Based on this knowledge and on the system model a formalism is developed that can be applied for the various communication operations supported by MPI. Thereby, we refer to two different types of point-to-point communication and various types of communications among a group of processes. We apply the formalism for the analysis of a concrete implementation of the MPI communications on a dedicated hardware platform. For that scripts are developed to automatically calculate timing upper bounds which depend on configurations for the communication (e.g. message length). We compare the guaranteed worst-case performance of different communication patterns that can be used for the same communication operation (see Section 2.3) to give a statement what fits best for the area of real-time. The building blocks of each communication are observed to exhibit the bottlenecks and the influences on the worst-case timing of the different components.

On application analysis level we present an algorithm to assemble all obtained WCETs in pseudocode. Similar to the analysis of MPI operations the workflow of this analysis is presented beforehand. Furthermore, we provide a case study that analyses a parallel benchmark with the methods developed in this work. Based on the revealed result we discuss the influences of the different components under analysis on the overall guaranteed worst-case performance.

1.4 Outline

In Chapter 2 we provide the background knowledge for the rest of the thesis and develop the system model to refer to the different aspects described. Thereby, we show our view on the hardware and routing behaviour that is targeted in this thesis. This is followed by the

description how the timing behaviour of communication can be guaranteed with a technique called time division multiplexing. Afterwards, we summarize the message passing interface that is investigated in this work. Finally, we give a short overview about real-time analysis.

In Chapter 3 the overview of the literature that is related to this work is provided. Thereby, on the one hand we focus on literature about providing *guaranteed service (GS)* to NoCs. On the other hand we target literature about the communication operations in MPI and the investigation for their performance.

The general-purpose TDM schedules for message-passing are presented in Chapter 4. At first, we provide a formal presentation and construction of various state-of-the-art schedules including proofs about their conflict-freedom. This is followed by the presentation of the two new schedules. Afterwards, the focus is put on the timing analysis of the schedules. Thereby, we firstly provide the general approach and show the application of this approach to the schedules at hand afterwards. Finally, the worst-case guarantees of all schedules are discussed and compared against each other and against representatives of state-of-the-art application specific schedules.

Chapter 5 presents the timing analysis of MPI communication operations. Thereby, we shortly refer to the workflow of the analysis and concentrate on the formalism to calculate to correct timing afterwards. We provide the calculation of point-to-point communication and further extend it to communication among groups of multiple processes. The evaluation of the formalism comprises the examination of timing upper bounds of different MPI operations. Thereby, we focus on the investigation of the building blocks of the communications and how they influence the timing behaviour and also compare different state-of-the-art communication patterns in order to provide a suitable pattern in the field of real-time computing. The chapter is concluded with the investigation of the influence of different schedules on the provided communication pattern and communication operations.

Afterwards, we focus on the timing analysis of a complete parallel program in Chapter 6. Similar to the analysis of MPI operations we firstly focus on the workflow of the analysis. This is followed by the presentation of the algorithm for assembling timing bounds to provide an end-to-end WCET upper bound for MPI programs. Afterwards, the algorithm is applied to calculate the guaranteed worst-case timing upper bound of a benchmark in a case study. The following evaluation provides insights on the influence of the different timing components regarded in this work.

Finally, Chapter 7 concludes the thesis and states some remarks on future work.

2

Background and System Model

In this chapter we provide background knowledge to our work on timing analysis of MPI programs targeting on-chip many-cores. Furthermore, the chapter specifies formal definitions that build the system model for the rest of the work.

There are multiple disciplines that influence the topic described in Chapter 1. Hence, we provide the background of each field in separate sections. At first, a description of the targeted hardware and how the routing is implemented is given, to build a base for the remainder of the background. Thereby, formal definitions and assumptions restrict possible architectures and routing policies to cases assumed in this work. Next, we concentrate on timing analysis of HRT systems. Besides some basic knowledge, the techniques applied in this work are described here. Afterwards, the focus is put on providing real-time guarantees for NoCs. In this section the concept of TDMA is explained. Furthermore, several variations of TDMA are explored and hints are given how to construct instances of TDMA schedules. The next section takes a closer look on MPI. Besides presenting relevant communication operations, we show the communication patterns used internally in MPI. Finally, we address applications that are running on the defined hardware and the determination of the underlying model of execution. Apart from this we explain how MPI fits to the application model and to the area of embedded systems.

The definitions in this chapter are inspired by [Mar+09; BS12; Sør+14] and are adapted and extended to fit for the purposes of this work.

2.1 Hardware Architecture and Routing Basics

As stated in Chapter 1 we target embedded systems that are capable for safety-critical HRT applications. Furthermore, special focus is put on workloads that require high performance demands compared to traditional HRT applications. Thus, we assume architectures that

exhibit a large number of processing elements to satisfy the stated performance demands. These chips are often referred to as many-cores.

When multiple components (e.g. processing elements) are integrated on one chip, there must be at minimum one component that enables the interaction among the components. Traditionally, mostly one or multiple communication buses are used for the communication among different parts of a chip [DT04, p. 427] [HP19, p. F-23]. But when the number of elements increases the bandwidth of a bus may not be sufficient any more to serve all communication requests in reasonable time. Therefore, in recent years the communication resources of chips where extended from buses to NoCs[Kum+02]. These networks are typically hard-wired networks based on short wires directly etched on the chip.

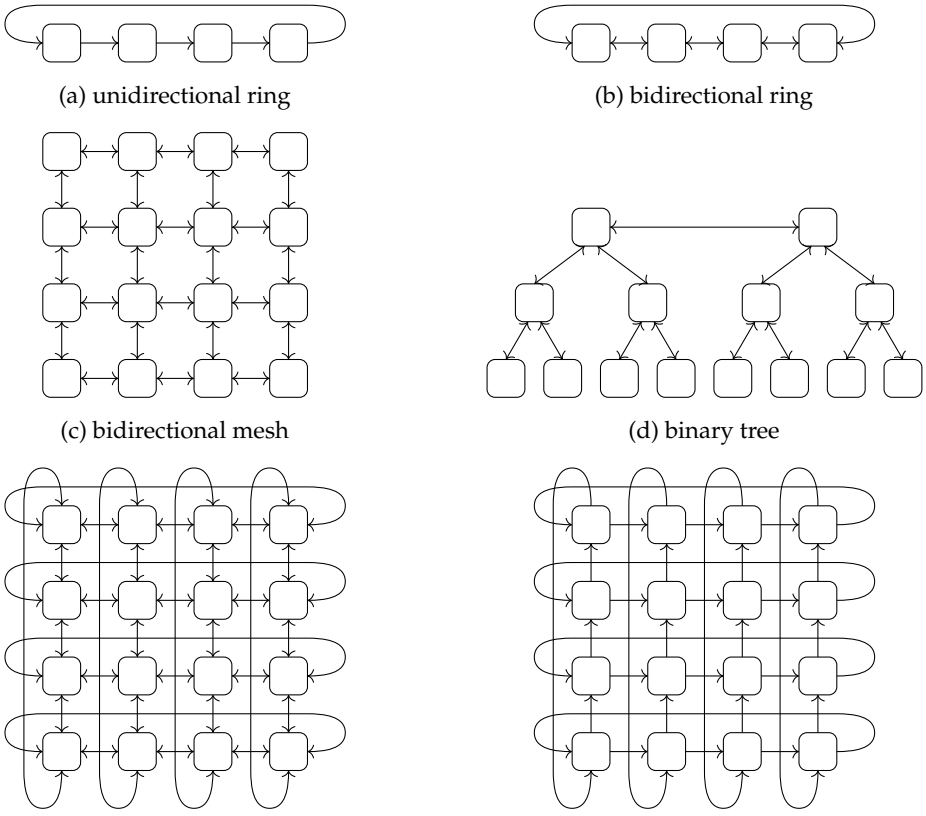
Regarding timing behaviour a NoC is often implemented as an asynchronous or a global asynchronous local synchronous network [Kum+02; Lud+11]. The main reason for favouring those design concepts instead of synchronous NoCs with a global time is the clock skew that impedes synchronisation in large designs [Lud+11]. Nevertheless, there are already efforts done to show that the design of a synchronous NoC is a valid option. These efforts concentrate on studies about low clock skew [Bin+03] or on avoiding the problem by utilizing a mesochronous clock on physical level but offering synchronous behaviour on logical level [HSG09; SMG14]. Subsequently, a synchronous NoC with global time is assumed.

The applied topology of those NoCs vary due to their area of applications. In the embedded area it is often the case that a chip is responsible for running a specific set of application software with a dedicated communication behaviour. In those cases the application's needs can be considered for the hardware design of the chip and the topology of the NoC can be designed to optimally meet the software's communication requirements [HP19, p. E-3]. The approach of a combined hardware/software solution is often used for *system-on-chips (SoCs)* and leads to highly specialized chips with good performance for the intended applications. However, their performance typically degrades significantly when executing other applications [DT04, p. 45f.].

Another approach is to design a multi- or many-core that fulfils general requirements for embedded systems (e.g. timing predictability) but is not optimized to a specific set of applications on hardware side [HP19, p. E-15] [DT04, p. 45f.]. Following this approach there are several standard topologies available to connect the components (mostly processing elements) [Sch+12] [DT04, p. 89ff.] [HP19, p. F-37]. Figure 2.1 displays a not exhaustive collection of different topologies.

Each network obtains different characteristics on e.g. bandwidth, latency and energy consumption [DT04, p. 51ff.]. In this work we focus on timing characteristics, which in fact mainly means latency and bandwidth.

The depicted topologies can be differentiated by the number of their dimensions. Typical one-dimensional topologies are the ring topologies, that are shown as a uni-directional (cf. Figure 2.1a) and a bi-directional ring (cf. Figure 2.1b). The two-dimensional NoCs can be distinguished in mesh and in tree like topologies. The meshes are expressed as a standard mesh (cf. Figure 2.1c), a uni-directional (cf. Figure 2.1f) and a bi-directional torus (cf. Figure 2.1e). An example of tree structures is the binary tree as it is shown in Figure 2.1d. Typically, trees obtain bi-directional links. In contrast to rings and meshes the trees often



(a) unidirectional ring (b) bidirectional ring

(c) bidirectional mesh (d) binary tree

(e) bidirectional torus (f) unidirectional torus

Figure 2.1: Several widely used topologies for NoCs. The rectangles with rounded corners indicate hardware nodes, while the arrows mark uni- and bi-directional links. Figures 2.1a and 2.1b show one-dimensional ring topologies, while 2.1c to 2.1f show two-dimensional topologies.

do not obtain processing elements at each nodes [Lei85]. Instead, only leaf nodes own processing elements, while the nodes of the rest of the tree consist of routers which are responsible for exchanging data between leaf nodes. Apart from one- and two-dimensional network topologies there exist a various number of higher dimensional topologies as for example cube topologies (three-dimensional) and hypercubes (n dimensional). Topology structures with more than two dimensions are not displayed in Figure 2.1.

As one can see the 2D mesh topologies with turn around links, aka the torus topologies, show links with different lengths, depending on whether a link is a turn around or not. This difference is not only an effect of drawing the topology, but is similarly present when building the actual hardware [DT04, p. 98f.]. The long wires suffer in latency of delivering data from one node to its desired neighbouring node compared to the shorter wires. As the network is driven by a global clock signal, this increased latency restricts the overall clock rate of the complete NoC. Hence, large parts of the NoC have to wait until the data is delivered along the long wires or the turn around links exhibit a latency of multiple clock cycles [DT04, p. 98f.]. This obvious disadvantage can be overcome with a sophisticated placement of the cores and wires on the chip. The placement leads to a topology called

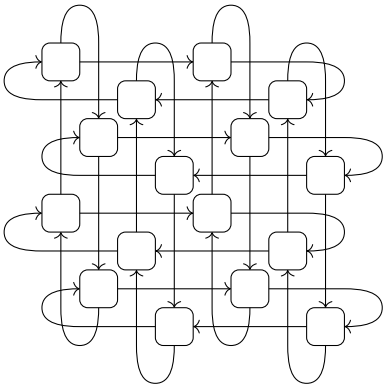


Figure 2.2: A folded torus of a uni-directional torus with 4×4 nodes. The topology is exactly the same as for a normal torus, but all wires exhibit equal length.

folded torus and shows exactly the same topology as a normal torus except that all wires exhibit the same length [DT04, p. 98f.]. An example of such a topology is depicted in Figure 2.2.

Subsequently, we consider many-cores that are not optimized to a small set of specific applications at hardware level, but focus on the possibility of executing different applications. Thus, we assume standard NoC topologies like displayed in Figure 2.1 and 2.2.

2.1.1 Definition of Hardware Architecture

Typically, a NoC architecture consists of several processing nodes that are connected via physical links. Additionally, there is a routing policy that defines the switching technique applied for transmitting data between nodes. Subsequently, we formally define our view on the hardware platform and some related characteristics we use throughout this work. The first definition provides a formal description of the nodes in a NoC.

Definition 2.1. *A node $n \in N$ of a NoC architecture is defined as a pair (r, c), where $r \in R$ is the node's router and $c \in C \cup \{\emptyset\}$ is the processing element (aka core) of that node. If a node only consists of a router, the missing core is marked by the symbol $\emptyset$. N is the set of available nodes of the NoC architecture, R is the set of routers and C the set of all cores of a NoC. The i-th node of a NoC is defined as $n_i = (r_i, c_i)$.*

Each core $c \in C$ comprises a *network interface (NI)* that connects the core to the local router of the node. We further assume that the routers in R do not support path-based routing like described in [LMN94], which means that all incoming data at a router $r \in R$ is exactly routed to one outgoing port. However, each router may contain buffers for temporally storing data until they are ejected to an outgoing port.

The routers of nodes that contain a processing element $c \in C$ obtain two links to the NI of c. One of the links serves for uploading data to the router, whereas the other link is used for download purposes. The NI comprises at minimum of a management logic to control the upload and download of data to and from the router. However, mostly NIs additionally include two buffers (send and receive buffer) that decouple the NoC from the core in terms of timing and memory space. Subsequently, we do not specify the structure of a core c. It

(a) intended logical execution order

(b) possible timing behaviour in reality

Figure 2.3: Conflicting communication. $ta0$ to $ta2$ indicate several tasks while $c1$ and $c2$ show different communications.

may be a single CPU connected to a local memory, a cluster of CPUs with shared memory and an additional *direct memory access (DMA)* unit or any other hardware component. The only restriction we assume to a processing element $c \in C$ is the ability to inject and eject data to and from the router r via a NI and that the component is timing predicable (cf. Section 2.2). Additionally, the timing within a single core shall not vary for communication over time. This assumption about timing shall hold for both directions (in and out of the core). Thus, each process of sending data to the NI takes the same time within a regarded node and the same holds for each receiving process of a node.

Data arriving at receiver nodes must be stored either in a dedicated receive buffer in the NI or directly in the node's local memory. If local memory is applied, the corresponding memory range must be prepared (e.g. establish exclusive memory access for NI). In those cases a problem is arising. Without flow control the receive buffers can overflow or the local memory may not actually be prepared if a task is participating in two communications, both in the role of a receiver (cf. $ta1$ in Figure 2.3). While it is receiving data from the first communication, a task participating in the second communication ($ta2$) is already sending data to the considered receiver. In those cases backpressure may occur to the NoC, which in turn may cause problems for the timing analysis. Thus, we assume a mechanism that ensures the absence of backpressure caused by this issue for a real world implementation. As there are several possibilities to implement an appropriate mechanism (e.g. in hardware or in software), we do not restrict our approach to a particular one.

After this view on the nodes of a NoC architecture, we will put the focus on the connections between the nodes. Besides the physical entity the subsequent description also comprises some performance metrics to characterize the defined components. We start with the definition of a physical link.

Definition 2.2. *A directed physical link $ph_{i,j} \in Ph$ is a pair (n_i, n_j) with $n_i, n_j \in N$. The symbol Ph denotes the set of directed physical links, each used to connect two nodes of the NoC. The direction of $ph_{i,j}$ is defined from n_i to n_j. Furthermore, we define $ph_{i,j}^{src}$ as the source node n_i of a physical link $ph_{i,j}$ and $ph_{i,j}^{dst}$ as its corresponding destination node n_j.*

The transportation of a portion of data from a node n_i to n_j over one physical link $ph_{i,j}$ is called a *hop*. The portions of data transported by a hop are called phits (physical transfer units). In contrast, a flit (flow control digit) is the basic unit of bandwidth and storage allocation used by flow control mechanisms. Subsequently, we assume that each flit consists

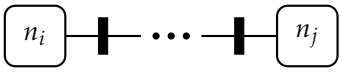


Figure 2.4: Usage of registers in a physical link. The white rectangles with rounded corners show two nodes connected via a physical link. Each black rectangle indicates a register that can store one flit at a time, if the link is not able to deliver a flit within one network cycle. Mostly, a link on NoC can be realized with at most one register between two nodes.

of one phit and thus, both terms are used in an interchangeable way. On hardware level each link $ph \in Ph$ consists of a number of parallel wires that are able to transport one flit at a time.

Phits and flits are the building blocks of packets and messages. A message is a certain amount of data that are delivered between two nodes. These nodes are not necessarily connected in a direct way. Hence, a message may traverse several physical links and routers until it reaches the destination node. As the size of messages can vary, they are typically split into packets which are basic units of data obtaining a maximal length. These packets are the basic units for routing the message in a network and include a header with information about routing and the data to be sent. Each packet consists of one or multiple flits, whereby flits are typically at minimum the size of the packet header and at maximum the flit size is equal to the size of its packet. The terms phit, flit, packet and message are described in more detail in [DT04, p. 223f.].

A physical link may consist of wires that are too long to transport a flit from one node to the next node within one network clock cycle. Such long distances can occur for example in turn around links of normal tori or in links of folded tori (cf. Figures 2.1 and 2.2). In those cases it is possible to build physical links that are able to pipeline the transportation of flits [BS12]. These links use additional registers placed on the physical link in a way that they split the link in parts of equal length (cf. Figure 2.4). A flit is then forwarded from one register to the next within one cycle. Typically, in chip development it is targeted to use no registers, aka to avoid this technique, as this is the best option from the performance point of view. Nevertheless, there are cases where at least one register is needed between two nodes, because otherwise the implementation in hardware would not be feasible. Thus, to be able to talk about the regions between two registers of a physical link, we define the term hardware slot as follows.

Definition 2.3. *A hardware slot $s \in S$ is the part of a physical link $ph \in Ph$ that is located between two consecutive registers or nodes on that link ph. S is the set of all slots in the NoC.*

In order to reason about timing we additionally define the time a flit needs to traverse one slot.

Definition 2.4. *The slot latency L^{slot} is defined as the time a flit needs to traverse from a particular physical link's register or source node to the next consecutive register or target node.*

In the remainder of this work we use the symbols $\langle$ and $\rangle$ to indicate the beginning and end of ordered lists of elements. When using the definition of a hardware slot it is possible to see a physical link also as an ordered list of slots $ph = \langle s_{first}, \ldots, s_{last} \rangle$. Typically, at most

two hardware slots (aka one register) are needed per physical link, but several registers on a link are possible with our definition. It is possible that in each cycle a register and slot is occupied by another flit. Thus, in the remainder of this work we assume that the slot latencies of all hardware slots in a NoC are equal. Furthermore, we define the bandwidth of a hardware slot as follows.

Definition 2.5. *The slot bandwidth B^{slot} is the number of flits a slot can forward within a given period of time.*

After defining the building blocks of a physical link it is possible to formulate some important characteristics. Those are the time needed to transport a flit over a hop and the number of flits that can be ejected to a specific link per unit of time.

Definition 2.6. *The hop latency $L_{i,j}^{hop}$ of a physical link $ph_{i,j}$ is defined as the time needed to transport one flit from a node n_i to a node n_j over the link $ph_{i,j}$. Furthermore, L_i^{hop} denotes the hop latency of the link from the NI of a core $c_i \in C$ to its corresponding router $r_i \in R$.*

Following Definition 2.6 the hop latency $L_{i,j}^{hop}$ of a physical link $ph_{i,j} = \langle s_1, \ldots, s_m \rangle$ can be calculated by the sum of all relevant slot latencies.

$$L_{i,j}^{hop} = \sum_{\{s|s \in ph_{i,j}\}} L_s^{slot} = |\{s \mid s \in ph_{i,j}\}| \cdot L^{slot} \tag{2.1}$$

Since an equal latency is assumed for all hardware slots, the hop latency $L_{i,j}^{hop}$ is the slot latency L^{slot} multiplied by the number of slots on the link $ph_{i,j}$.

In addition to latency the bandwidth is an important characteristic. Therefore, we also define the bandwidth of a hop.

Definition 2.7. *The hop bandwidth $B_{i,j}^{hop}$ of a physical link $ph_{i,j}$ is defined as the number of flits injectable to the link $ph_{i,j}$ within a given period of time. Furthermore, B_i^{hop} denotes the hop bandwidth of the link from the NI of a core $c_i \in C$ to its corresponding router $r_i \in R$.*

The hop bandwidth of a physical link $ph_{i,j}$ is directly related to the amount of data a hardware slot is able to forward within the regarded period of time. Hence, we can calculate the hop bandwidth as follows.

$$B_{i,j}^{hop} = \min\{B_s^{slot} \mid s \in ph_{i,j}\} \tag{2.2}$$

Since each physical link exhibits a direction, they map directly to the links used in uni-directional topologies depicted in Figure 2.1. A bi-directional link of the bi-directional topologies is created by assuming two links $ph_i, ph_j \in Ph$ between the dedicated nodes with opposed directions. Thus, a bi-directional connection for two nodes n_i and n_j is instantiated by using the physical links $ph_{i,j}$ and $ph_{j,i}$.

A further component of a NoC architecture is the routing policy that determines the routing behaviour of a router $r_i \in R$.

Definition 2.8. *The routing policy $R_i^P : N \times N \to \mathbb{N}_0 \times N, (n_{src}, n_{dst}) \mapsto (\Delta t, n_j)$ is a function that defines the forwarding strategy for flits that are sent between a source node n_{src} and a destination node n_{dst} when arriving at router r_i. Thereby, the regarded router stores the data for the given time interval Δt and afterwards forwards it to the physical link $ph \in Ph$ directly pointing to node n_j. We denote the set of all routing policies as R^P.*

The pairs $(\Delta t, n)$ determined by R_i^P are restricted to nodes $n \in N$ that are directly connected to the regarded node n_i via an existing physical link $ph \in Ph$. Thus, the following condition must hold.

$$\forall(\Delta t, n_j), (\Delta t, n_j) = R_i^P(n_{src}, n_{dst}) : \exists ph \in Ph : ph = (n_i, n_j) \tag{2.3}$$

Furthermore, the time interval Δt is equal for all data that are characterized by an equal pair (src, dst).

$$\exists j, k \in \mathbb{N}_0 : ((\Delta t_j, n_j) = R_i^P(src, dst) \wedge (\Delta t_k, n_k) = R_i^P(src, dst)) \Rightarrow \Delta t_j = \Delta t_k \tag{2.4}$$

There are routers available that obtain a dynamically determined storage time Δt (depending on the currently occurring traffic e.g. like in wormhole routing) for messages sent over the same path. Nevertheless, we restrict our approach to fully constant storage time for each element of a given message, which is valid for the usage of TDMA (cf. Section 2.2). After defining the basic components of the architecture, it is possible to give a formal description of a NoC architecture.

Definition 2.9. *A NoC architecture is described as a triple $Arch = (N, Ph, R^P)$. Thereby, N is the set of nodes of the architecture and Ph is the set of existing physical links. The third part of the architecture is the set of routing policies R^P which includes one policy for each router in the NoC. The two sets N and Ph form a pair $Topo = (N, Ph)$ that describes the topology of the NoC.*

Subsequently, it is necessary to refer to particular nodes and their position in the regarded topology. Thus, the following definitions allow a mapping of a particular node $n_i \in N$ to its corresponding position in the topology and vice versa.

Definition 2.10. *The function $Pos : N \to (\mathbb{N}_0, \mathbb{N}_0), n \mapsto (x, y)$ maps a node n to a pair (x, y) representing the node's coordinate in a two-dimensional topology.*

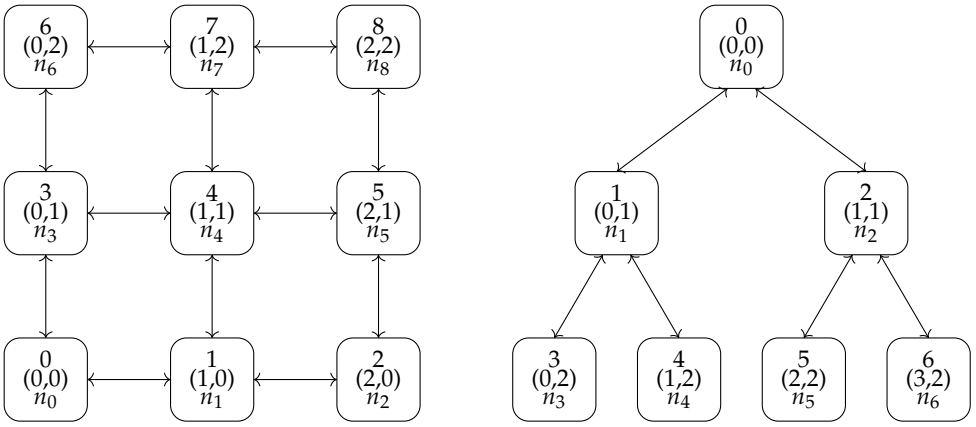
Regarding Definition 2.10 in mesh and torus like topologies x denotes the horizontal position offset from a dedicated node, while y indicates the vertical position offset from the dedicated node. Thereby, an offset of 1 indicates the position reachable when executing one hop. The horizontal offset x raises from left to right and the vertical offset goes from bottom to top. Furthermore, in tree like topologies y refers to level of depth on which the node is located and x shows the horizontal position on that level. Thus, y raises in terms of hops on the way from the root node to a leaf node and x behaves just as in the definition for mesh topologies.

Definition 2.11. *$PE : (\mathbb{N}_0, \mathbb{N}_0) \to N, (x, y) \mapsto n$ is a function the maps a topology's position $(x, y) \in \mathbb{N}_0 \times \mathbb{N}_0$ to the corresponding node $n \in N$ located at that position. The function is the direct inverse of Pos and the following condition holds: $PE = Pos^{-1}$.*

Please note that there is no need for position functions regarding one-dimensional topologies, as the position is trivially given by a node's index i (for a node $n_i \in N$). Figure 2.5 illustrates the usage of the functions Pos and PE.

Finally, we define several characteristics that describe topologies of NoCs.

Definition 2.12. *A network topology $Topo$ is direct if each node $n \in N$ includes a processing element $c \in C$. Thus, a direct topology must hold the condition: $\nexists n \in N : n = (r, \emptyset)$. If the condition is not met, the topology is indirect.*



(a) position in meshes

(b) position in trees

(c) legend

Figure 2.5: Illustration of functions Pos and PE. The rectangles mark nodes of topologies and arrows refer to physical links. The content of a node is described in 2.5c.

Definition 2.13. *A network topology is regular if each $n \in N$ is connected to the same number of physical links $ph \in Ph$.*

Definition 2.14. *A network topology is symmetric if the topology looks the same from each point of view of all topology's elements.*

The Definitions 2.13 and 2.14 are rather informal and tailored to the purposes of this work. The concrete mathematical description of regular and symmetric topologies can be found in [Bol02, p. 4], [DT04, p. 49f.] and [MSA08].

2.1.2 Definition of Routing Behaviour

After describing the hardware components, the focus changes to the characteristics of routing data through the previously described NoC architecture $Arch$. When talking about routing it is necessary to reason on the route a flit takes on its way through the NoC from its source to its target node. Thus, at first there is a need to associate a particular hop of a flit to a designated point in time.

Definition 2.15. *A scheduled hop $h \in H$ is triple (fl, ph, t) where $fl \in F \cup \{\emptyset\}$ is the transported flit, $ph \in Ph$ is a NoC's physical link and t denotes a particular point in time. F is the set of all flits that are communicated via the NoC and $fl = \emptyset$ denotes that h is not occupied by any data. The time is given as a discrete offset value, starting at a designated point in time ($t \in \mathbb{N}_0$).*

In addition to Definition 2.15, we allow the notation of a scheduled hop h as pair (ph, t) if there is no restriction to the data carried by the hop. Thus, it is not specified which flit is

transported by h or if h carries a flit at all ($fl = \emptyset$). In the remainder of this work we use the $\{.\}$-operator to refer to components that are included in surrounding components. If for example a term $h.t$ is applied, then it refers to the component t of the scheduled hop $h = (ph, t)$. Since in Definition 2.15 t is a discrete value, each instance of t represents the time interval $[t, t+1[$. Moreover, the mentioned time interval is assumed to be exactly the slot latency L^{slot}. As a hop designates the transport over a complete physical link, the length of a scheduled hop directly corresponds to the regarded link's hop latency L^{hop}. Please note that there is a difference between $h.t$, which denotes the point in time when the scheduled hop h starts, and the length of h, which indicates the duration to travel along the scheduled hop.

Based on Definition 2.15 we declare a communication path to indicate the route traversed by a flit on its way through the NoC and its corresponding timing.

Definition 2.16. *A communication path $\vec{p} = \langle h_{first}, h_{first+1}, \ldots, h_{last} \rangle \in P$ is a sorted list of scheduled hops $h \in H$. The Conditions 2.5 and 2.6 must hold for each path $\vec{p}$ to ensure temporal and spatial correctness. The symbol P denotes the set of all possible paths through a considered NoC.*

Essentially, a communication path is a sequence of consecutive physical links that must be traversed in order to reach the destination node. Additionally, each link is marked with a time offset which indicates the point in time when the first slot of the link is occupied by the flit. Hence, it is reasonable to use scheduled hops to describe a communication path. Furthermore, the following two conditions are used to ensure temporal and spatial correctness for communication paths.

$$\forall \vec{p} \in P : \forall h_i \in \vec{p} \setminus \{h_{last}\} : (h_i.t + L^{hop}_{h_i.ph} + R^P_{h_i.ph^{dst}}.\Delta t = h_{i+1}.t) \tag{2.5}$$

$$\forall \vec{p} \in P : \forall h_i \in \vec{p} \setminus \{h_{last}\} : (h_i.ph^{dst} = h_{i+1}.ph^{src}) \tag{2.6}$$

Condition 2.5 focuses on the temporal aspect of a correct communication path. It requires that the difference in time between two consecutive scheduled hops h_i and h_{i+1} is exactly determined by the hop latency of the traversed hop $L^{hop}_{h_i.ph}$ and the time the receiving router buffers the data before forwarding it along the next hop h_{i+1}. In contrast, Condition 2.6 concentrates on spatial correctness. Thus, it requires that the physical links of two consecutive hops are connected to the same node and that they point to the correct directions. Thus, it requires that the destination node of h_i is the same as the source nodes of h_{i+1}.

In the remainder of this work we refer to the position $POS(fl)$ of a flit fl that is transported on a path $\vec{p}_i$ and is at the moment in the scheduled hop h_l by using the following notation:

$$POS(fl^i_l) = POS(\vec{p}_i.h_l.ph^{src}) = (x^i_l, y^i_l) \tag{2.7}$$

Thereby, the symbol fl^i_l denotes a flit delivered along path $\vec{p}_i$ that is currently located in the path's scheduled hop h_l.

A communication path $\vec{p}$ describes the complete traversed way of a flit through the network, whereas a routing policy R^P_i focuses on a particular router and describes its routing behaviour. Nevertheless, the information provided by both perspectives must fit together. Their conjunction is described in the Conditions 2.8 and 2.9.

$$\forall h_i \in \vec{p}, \vec{p} \in P : h_i.ph = (n_l, n_m) \Rightarrow R^P_l(\vec{p}.h_{first}.ph^{src}, \vec{p}.h_{last}.ph^{dst}) = n_m \tag{2.8}$$

$$\forall \vec{p} \in P : \exists h_i \in \vec{p} : R_l^P(\vec{p}.h_{first}.ph^{src}, \vec{p}.h_{last}.ph^{dst}) = n_m \Rightarrow h_i.ph = (n_l, n_m) \tag{2.9}$$

Conditions 2.8 and 2.9 ensure that for each scheduled hop h_i there is an equivalent in the routing policy of the hop's source node $R^P_{h_i.ph^{src}}$ and vice versa.

In some situations there is a need to refer to a path through a NoC, but explicitly without specifying concrete instances of time. Thus, following Definition 2.16, we additionally define routes similar to communication paths but without explicit information about timing.

Definition 2.17. *A route $\vec{r} = \langle ph_{first}, ph_{first+1}, \ldots, ph_{last} \rangle \in Rt$ is a sorted list of physical links $ph \in Ph$. Thereby, Rt denotes the set of all routes. A route $\vec{r}$ is valid if Condition 2.10 holds.*

$$\begin{aligned} \exists \vec{p} = &\langle h_{first}, h_{first+1}, \ldots, h_{last} \rangle \in P : \\ &\forall ph_i \in \vec{r} = \langle ph_{first}, ph_{first+1}, \ldots, ph_{last} \rangle : ph_i = h_i.ph \end{aligned} \tag{2.10}$$

Condition 2.10 requires that there exists at minimum one possible communication path $\vec{p}$ that takes exactly the same way through the NoC as the regarded route $\vec{r}$. Hence, Condition 2.6 equally holds for the physical links of the considered route. Definition 2.17 and Condition 2.10 imply that a route $\vec{r}$ can map to multiple communication path $\vec{p}$ but each path refers to only one route. The route of a path is denoted as $\vec{r}_{\vec{p}}$.

Definition 2.18. *The set of possible routes between two nodes n_{src} and n_{dst} is denoted as $\vec{r}_{src,dst}$. $\vec{r}_{src,dst}$ is a subset of Rt ($\vec{r}_{i,j} \subset Rt$) and satisfies Condition 2.11.*

$$\vec{r}_{i,j} = \{\vec{r} \mid \vec{r}.ph^{src}_{first} = n_i \wedge \vec{r}.ph^{dst}_{last} = n_j\} \tag{2.11}$$

Definition 2.19. *The physical length $pl_{\vec{r}}$ of a route $\vec{r}$ is defined as the number of slots that must be used to completely traverse the route $\vec{r}$.*

The physical length of a route can be calculated as follows:

$$pl_{\vec{r}} = \sum_{ph \in \vec{r}} |\{s \mid s \in ph\}| \tag{2.12}$$

The routing in a NoC can be realized in different ways. On the one hand it is possible to utilize routing tables that must be implemented in each router of the network (e.g. described in [Bol+07]). With that technique besides some special variants typically each data packet carries its destination node in its header. Then for each incoming packet the router performs a lookup to the routing table and forwards the packet according to the packet's destination node and table entry. These tables are either implemented with reconfigurable routing entries or are fixed as electric circuits. Thus, it is possible to develop statically and dynamically routed NoCs depending on the targeted application area. Routing tables perfectly map to the routing policies R^P defined previously, if the lookup and forwarding is performed after storing the packet for the time Δt specified by R^P_i.

Another possibility to realize routing is called source routing (e.g. see [SKS13]). Here the route is completely located in the packet's header including all intermediate nodes on the path to the destination node. Thus, the routes of packets are typically determined at the

sender node, which may be the same route $\vec{r}$ for all packets that target the same node or adapted dynamically. However, after releasing a packet to the NoC the route is fixed. With message based routing the headers of packets must carry more information than when using routing tables. This may be a significant amount of additional data to be sent when the communication is based on a lot of small sized packets. Message based routing can be seen more like implementing the perspective of communication paths. Although our approach is equally applicable to routing table based routing and to source routing, in the remainder of this work we assume the routers to be implemented with routing tables.

It follows some performance metrics that are important for descriptions in the following sections and chapters. Since this work focuses on timing, we restrict the metrics to the fields of throughput and timing. At first, based on the latency and bandwidth of a hop, we define the latencies and bandwidths of complete paths.

Definition 2.20. *The path latency $L^{path}_{\vec{p}_i}$ of a path $\vec{p}_i \in P$ is defined as the time needed to transport one flit from the source node n_{src} of path $\vec{p}_i$ to its target node n_{dst} along the physical links and routers given by $\vec{p}_i$.*

Following Definition 2.20 the path latency can be calculated by considering the latencies of all hops and the times for buffering introduced by the routing policies.

$$L^{path}_{\vec{p}_i} = \sum_{\{h_i | h_i \in \vec{p}_i\}} [L^{hop}_{h_i.ph} + R^P_{h_i.ph^{src}}(\vec{p}_i.h_{first}.ph^{src}, \vec{p}_i.h_{last}.ph^{dst}).\Delta t] \tag{2.13}$$

The hop latency $L^{hop}_{h_i.ph}$ is only dependent on the physical link of h_i and is constant over time. Additionally, the buffering time for a routing policy $R^P_i(n_{src}, n_{dst})$ remains equal for all paths on the same route:

$$\vec{r}_{\vec{p}_i} = \vec{r}_{\vec{p}_j} \Rightarrow L^{path}_{\vec{p}_i} = L^{path}_{\vec{p}_j} \tag{2.14}$$

Thus, the path latency of all paths $\vec{p}_i$ with the same route $\vec{r} = \vec{r}_{\vec{p}_i}$ can also be denoted as $L^{path}_{\vec{r}}$.

Similar to the latency we define the bandwidth of a path as well.

Definition 2.21. *The path bandwidth $B^{path}_{\vec{p}}$ of a path $\vec{p} \in P$ is defined as the number of flits injectable from the source node n_{src} of $\vec{p}$ to the considered path $\vec{p}$ per given period of time without producing backpressure in the NoC.*

The path bandwidth can be calculated using the minimal bandwidths of all traversed hops:

$$B^{path}_{\vec{p}} = \min\{B^{hop}_{i,j} \mid \exists h \in \vec{p}, ph_{i,j} \in Ph : h.ph = ph_{i,j}\} \tag{2.15}$$

Similar to the latencies the bandwidth is constant over time and independent of the concrete data sent. For that reason the path bandwidths for all paths $\vec{p}_i$ that have the same route $\vec{r}$ are equal and can be indicated by $B^{path}_{\vec{r}}$ as well.

2.2 Time Division Multiple Access

TDMA [SP68] or TDM is a classical technique that manages the usage of a shared resource among multiple instances that require access to it. In TDMA each of the competing instances

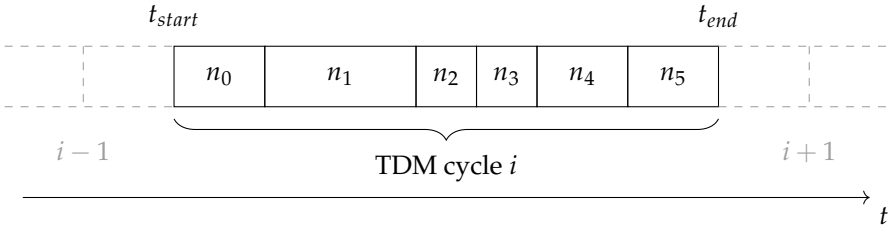


Figure 2.6: Concept of TDMA. One TDM cycle i is illustrated, which begins at t_{start} and ends at t_{end}. It is surrounded by the cycles $i-1$ and $i+1$ (indicated with dashed lines). Each rectangle indicates a time slot that grands access to the shared resource by the corresponding node (n_0 to n_5). The time goes from left to right and thus, the width of each time slot indicates the time a node is allowed to access the resource.

has dedicated periods of time when they are allowed to use the shared resource. These slots of time are arranged in a way that each participant has exclusive access to the resource or its required part of the resource. Hence, this technique is intended to be used to guarantee the absence of conflicts between two or more concurrently entering instances. Typically, the arranged time slots are reused periodically, in a way that each instance participating in the TDMA is permitted to access the protected resource at the dedicated time slots in each period. Thus, it is able to define a pattern of allowed accesses within a fixed period of time, that is repeated to manage the access to the resource over the complete time of execution. A concrete instance of those patterns is called a TDM schedule and one instance of the pattern's periodic execution is called a TDM cycle. Although there are dynamic variants [ROG06], mostly TDM schedules are defined at design time and are not changed during execution. If not stated otherwise, in the remainder of this work we consider TDM schedules to be static. Figure 2.6 illustrates the fundamental concept of TDMA.

TDM is a standard technique and can be utilized for different purposes. Some examples are wireless sensor networks [PC01; ROG06; EV10], memory access [DC98; Cil+15] or bus arbitration in general [WT06]. In this work we use TDMA to manage the access to the NoC, as it is a standard technique for that use case, too [Sch07; GH10; SMG14]. If not stated otherwise, we subsequently focus on the management of a NoC when we talk about TDM or TDMA. Hence, hereafter we describe the specifics of TDM for the usage of protecting a NoC from conflicts and unpredictable delays in transportation of data.

Typically, in a NoC the instances that compete for access to the resource are single portions of data that are intended to traverse the NoC. Most hardware and routing implementations see the packets of a message as these portions of data. Nevertheless, there is no formal convention about how many flits are included in one packet. Therefore, in this work flits are seen as the described portions of data and define a message as follows.

Definition 2.22. *A message* $msg = \langle fl_0, fl_1, \cdots, fl_{last} \rangle \in MSG$ *is a sorted list of flits* $fl \in FL^{msg}$*. The symbol* FL^{msg} *denotes the set of all Flits of the message* msg*, while MSG marks the set of all messages in an application.*

If the routing implementation requires that packets consisting of multiple flits must be transported as one instance, the TDM schedule is able to ensure this by granting access to

the required number of flits in successive time slots.

In contrast to some other resources like e.g. bus arbitration [Ros+07; SCT10] a NoC is not completely occupied when one instance is accessing it. Instead, each flit only occupies one hardware slot $s \in S$ at a dedicated instance of time. As stated in Section 2.1 each hardware slot belongs to a physical link $ph \in Ph$. Furthermore, the slots that are occupied by a flit at a given point in time are determined by the communication path $\vec{p}$ that is applied to communicate the flit between two nodes n_i and n_j. The path causes a flit to always occupy consecutive slots and physical links along its corresponding route at consecutive points in time. Thus, we can retrace the way of a flit through the NoC and also determine the concrete position of a flit at a given point in time. This traceability enables a prediction of the time a flit needs to traverse the NoC in the worst case.

Typically, in TDMA all paths needed for the communication among nodes within the considered application are arranged in a way that no hardware slot concurrently belongs to multiple paths at any point in time. Thereby, the paths are chosen in a way that they form a finite arrangement in terms of time and number of included path. This arrangement is repeated to provide access to the NoC over the entire time of an application's execution. Therefore, a TDM schedule can be formally defined as follows.

Definition 2.23. *A TDM schedule tdm^S is a triple (P^{tdm}, T, Co), where $P^{tdm} \subset P$ denotes a set of paths, T indicates the period of the schedule and Co is a set of constraints that restrict the usage of the given paths.*

The time components of the paths' scheduled hops $\vec{p}.h.t$ are typically given relative to the beginning of the period T, which is denoted as $t_0 = 0$. Furthermore, the time is stated modulo to the period T like described in Condition 2.16.

$$\forall \vec{p} \in P^{tdm} : \forall h \in \vec{p} : h.t = h.t \bmod T \tag{2.16}$$

Please note, that stating the time modulo to the period explicitly allows paths that cross the timing border of the period. Hence, paths are allowed that start near the end of the period cross the limit given by the period and finish shortly after t_0 during the execution of the next period (see Figure 2.7b).

Since the paths of a TDM schedule are repeated after passing the end of a period, a mechanism is needed to refer to a specific instance of the schedule's execution. Thus, we define the term TDM cycle to indicate a specific instance of a TDM schedule's execution.

Definition 2.24. *A TDM cycle tdm_i^C is defined as a concrete instance of a TDM schedule tdm^S The index i indicates the i-th cycle of the schedule which begins at $t_0 = i \cdot T$.*

Please note, that a path of a TDM schedule can act as representative for all corresponding paths of the TDM cycles belonging to the regarded schedule.

Each constraint $co \in Co$ stated in Definition 2.23 is a formal expression that can be evaluated either to *true* or to $false$. The set Co is not empty for instance when routing requires to send multiple flits successively as they belong to the same packet. The constraint ensuring this requirement for a packet of n flits is given in Condition 2.17.

$$\begin{aligned}\forall \vec{p} \in P^{tdm} : \exists P^{msg} \subset P^{tdm} : \vec{p} \in P^{msg} \wedge |P^{msg}| = n \wedge (\forall \vec{p}_i, \vec{p}_j \in P^{msg}, i \neq j : \\ \vec{r}_{\vec{p}_i} = \vec{r}_{\vec{p}_j} \wedge \forall \vec{p}_i.h_l, \vec{p}_j.h_l : L^{slot} \leq |\vec{p}_i.h_l.t - \vec{p}_j.h_l.t| \leq nL^{slot})\end{aligned} \tag{2.17}$$

The equation states that each path belongs to a subset P^{msg} of all schedule's paths P^{tdm} that includes in total n paths. All these paths share the same route and the difference in time between the scheduled hops h of the paths with the same index l is at minimum L^{slot} but not larger than nL^{slot}. Thus, this constraint forces the schedule to provide n consecutive paths that obtain the same route.

An important characteristic of TDM schedules is their freedom of conflicts which means that no two flits are ever intended to occupy the same hardware slot at the same point in time. Thus, we define the characteristic conflict-free as follows.

Definition 2.25. *A schedule is said to be conflict-free if there are no two paths that share the same hardware slot $s \in S$ of a physical link $ph \in Ph$ at the same point in time. Thus, Condition 2.18 holds.*

$$\begin{aligned} &\forall \vec{p}_i, \vec{p}_j \in P^{tdm}, \vec{p}_i \neq \vec{p}_j : \nexists (h_i, h_j) \in \vec{p}_i \times \vec{p}_j : \\ &\quad h_i.ph = h_j.ph \wedge h_i.t \bmod T = h_j.t \bmod T \end{aligned} \tag{2.18}$$

This Condition 2.18 requires for the complete TDM schedule that there are no two identically scheduled hops h_i and h_j no matter if or which data occupies those hops. Hence, it ensures that each hardware slot in the regarded NoC architecture is reserved by at most one path $\vec{p}$ of the TDM schedule at each point in time.

The restriction of a TDM schedule that there are no two hops h_i, h_j allowed to share the same physical link and time no matter if data is actually carried or not leads to possibly empty slots. These slots arise when paths of the schedule are not used for a communication in one or more TDM cycles of that schedule. Hence, though most TDM schedules rely on conflict-freedom as stated in Definition 2.25 (like in [Sør+14; Har+18]), we additionally give a weaker definition to enable the utilization of those slots.

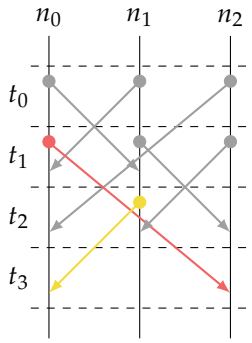
Definition 2.26. *A TDM schedule is said to be relaxed conflict-free if there are no two paths that share the same hardware slot $s \in S$ of a physical link $ph \in Ph$ at the same point in time and transport flits along the regarded paths in the same TDM cycle. Thus, it follows Condition 2.19.*

$$\begin{aligned} &\forall \vec{p}_i, \vec{p}_j \in P^{tdm}, \vec{p}_i \neq \vec{p}_j : \nexists (h_i, h_j) \in \vec{p}_i \times \vec{p}_j : \\ &\quad h_i.ph = h_j.ph \wedge h_i.t \bmod T = h_j.t \bmod T \wedge h_i.fl \neq \emptyset \wedge h_j.fl \neq \emptyset \end{aligned} \tag{2.19}$$

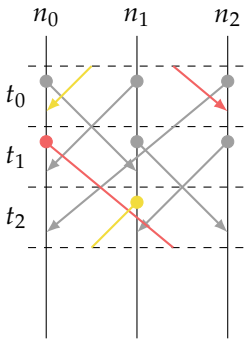
In contrast to Condition 2.18 the condition about relaxed conflict-freedom only requires that two actually present flits do not share the same hardware slot at the same point in time. Hence, a schedule is allowed to have multiple $\vec{p}_i \in P^{tdm}$ that share identical scheduled hops h. Please note that relaxed conflict-freedom can only be ensured by TDMA if suitable constraints are given. A TDM schedule is only valid, if it is (relaxed) conflict-free and all constraints $co \in Co$ can be evaluated to *true*.

$$\forall co \in Co : co = true \tag{2.20}$$

If no constraints are needed for a concrete TDM schedule, the schedule's set of constraints is empty ($Co = \emptyset$).



(a) Schedule's paths without modulo timing or $T \geq 4$



(b) Schedule's paths with modulo timing for $T = 3$

Figure 2.7: Global view of an example for a TDM schedule. Each vertical line represents a node n_i and the time goes from top to bottom in terms of time slots (separated by dashed lines). Arrows mark the paths of the schedule. Thereby, a bold dot on a node's line illustrates the time slot when a flit can be inserted to the NoC along that path. The arrow tips indicate at which time slot a flit arrives at its target node. The red and yellow arrows mark paths that cross the timing border of a period when it is set to $T = 3$.

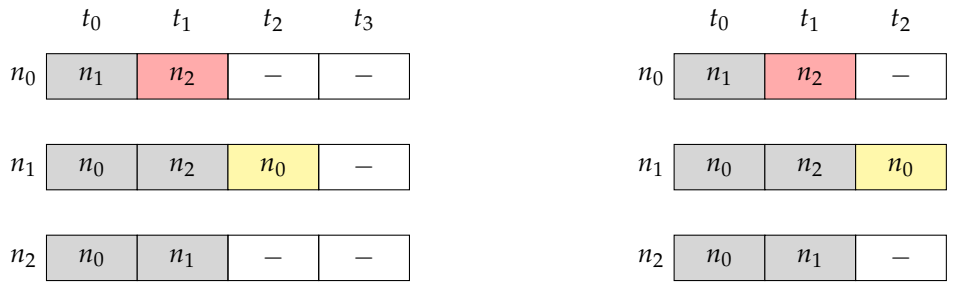
Since the main component of TDM schedules is the set of paths, a schedule can be illustrated as the arrangement of those paths. Figure 2.7 depicts an example of such an arrangement. The figure shows two schedules that define paths for the communication among three nodes n_0 to n_2. The nodes n_0 and n_2 are both allowed to send one flit per TDM cycle to each other node. Node n_1 is restricted to send one flit to n_2, too, but is allowed to send two flits to n_0. The two schedules differ in their periods which is 4 time slots for Figure 2.7a and 3 time slots for Figure 2.7b. Subsequently, we refer to this kind of illustration as the global view.

Besides the global view it is possible to look at a schedule from the perspective of a dedicated node. This kind of view is called the local view. Using the local perspective of a node a TDM schedule looks like an arrangement of time slots. These slots tell the regarded node if and when it is allowed to inject flits to the network. Furthermore, besides the determination of the timing, the allowed path is indicated by each slot as well. Figure 2.8 illustrates the local views of all nodes for the TDM schedule examples depicted in Figure 2.7.

Because of the importance of time slots for the local view of TDM schedules, we concretise the informal description of time slots by giving a formal definition.

Definition 2.27. *A time slot s^t is a pair $(\vec{r}, t)$ that indicates that the source node $\vec{r}.n^{src}$ of route $\vec{r}$ is allowed to inject one flit to the NoC along the route $\vec{r}$ at time t. The symbol $s_i^{t,j}$ denotes the i-th time slot from the schedule's local view of node n_j.*

Typically, the length of a time slot s^t is the slot latency L^{slot}, as this is the latency needed for injecting one flit to the NoC. Hence, we see the case that a node n_i is allowed to send f flits along the same route at consecutive points in time (e.g. when sending a complete packet of f flits) as f time slots. For simplicity we define the set of consecutive time slots for



(a) Local views without modulo timing or $T \geq 4$

(b) Local views with modulo timing for $T = 3$

Figure 2.8: The nodes' local views of the examples from Figure 2.7. Each row represents the local view of one node and each rectangle indicates a time slot. The points in time for the slots are marked with t_0 to t_3. The label of each time slot indicates the target node of the slot's corresponding route and the colours refer to the coloured paths of Figure 2.7. A dash means that a flit injection is not allowed in that slot.

the same route as block of time slots.

Definition 2.28. *A block of time slots b^t is a list $\langle s^{t,j}_{first}, \ldots, s^{t,j}_{last} \rangle$ of consecutive time slots $s^{t,j}_i$ which satisfies $s^{t,j}_i.t + L^{slot} = s^{t,j}_{i+1}.t$ and that share the same route $s^t.\vec{r}$.*

Furthermore, we define the sets of paths beginning in the same time slot s^t as follows.

Definition 2.29. *The set $P^s_i \subset P^{tdm}$ is the set of all paths of the schedule tdm^S that begin in time slot s^t_i. The set $P^s_{i,j} \subset P^s_i$ denotes the subset of all paths $\vec{p} \in P^s_i$ that exhibit the same source node n_j.*

Hence, the sets P^s_i and $P^s_{i,j}$ are determined by the following Equations 2.21 and 2.22.

$$P^s_i = \{\vec{p} \mid \vec{p} \in P^{tdm} \wedge \forall \vec{p}_j, \vec{p}_k \in P^s_i : \vec{p}_j.h_{first}.t = \vec{p}_k.h_{first}.t\} \tag{2.21}$$

$$P^s_{i,j} = \{\vec{p} \mid \vec{p} \in P^s_i \wedge \vec{p}.h_{first}.ph^{src} = n_j\} \tag{2.22}$$

In the remainder of this work, it is necessary to reason about the routes available in a TDM schedule. Thus, we define this set of routes as follows:

Definition 2.30. *The set of routes $tdm^R \subset Rt$ of a TDM schedule tdm^S is defined as a set of routes satisfying the Conditions 2.23 and 2.24.*

$$\forall \vec{p} \in tdm^S.P^{tdm} : \exists \vec{r} \in tdm^R : \vec{r}_{\vec{p}} = \vec{r} \tag{2.23}$$

$$\forall \vec{r} \in tdm^R : \exists \vec{p} \in tdm^S.P^{tdm} : \vec{r} = \vec{r}_{\vec{p}} \tag{2.24}$$

Following Definitions 2.23 to 2.30 each node is allowed to inject a flit to the NoC along a path $\vec{p}_i$ only at the points in time given by $\vec{p}_i.h_{first}.t$. These times mark the time slots for all nodes $\vec{p}_i.h_{first}.ph^{src}$ and no other injection of data to the NoC is allowed. If more than one flit shall be sent along a particular route within one TDM cycle, the schedule includes multiple paths with the same route. Typically, the applied TDM schedule is the dominating restriction for the bandwidths for each provided route.

Definition 2.31. *A TDM bandwidth $B_{\vec{r}}^{tdm}$ of a route $\vec{r}$ provided by a schedule tdm^S is the bandwidth that can be guaranteed for that given route $\vec{r}$ when applying the schedule tdm^S.*

Since the bandwidth generally is defined as the amount of data that can be handled within a fixed period of time, there are two characteristics that influence the bandwidth of a route in a TDM schedule. The first one is the period of the schedule. If leaving constant the handled amount of data per TDM cycle, the bandwidth raises if the period is decreased, which causes an indirect proportional relation to the period. The second characteristic is the handled amount of data per TDM cycle. Varying this value leads to a directly proportional variation of the bandwidths when the period T remains constant. Thus, the relationship of the bandwidth of a TDM schedule to the characteristics period T and handled amount of data per TDM cycle in terms of number of flits $f = |F|$ can be expressed like in Equations 2.25 and 2.26.

$$B^{tdm} \sim \frac{1}{T} \tag{2.25}$$

$$B^{tdm} \sim f \tag{2.26}$$

Since each path is suitable to send one flit per TDM cycle, the TDM bandwidth of a route $\vec{r}$ is calculated by considering the number of paths $\vec{p} \in P^{tdm}$ of the schedule that use the same route $\vec{r}$ and the schedule's period.

$$B_{\vec{r}}^{tdm} = \frac{|\{\vec{p} \mid \vec{p} \in P^{tdm} \wedge \forall h_i \in \vec{p} : h_i.ph = \vec{r}.ph_i\}|}{T} \tag{2.27}$$

Typically, the bandwidth is not bounded by the hardware limit $B_{\vec{r}}^{path}$ (see Section 2.1) when utilizing a concrete TDM schedule, since this bandwidth marks the maximum bandwidth the hardware can provide to a route $\vec{r}$. Hence, $B_{\vec{r}}^{path}$ is the upper bound that a schedule can grant to a route $\vec{r}$.

$$B_{\vec{r}}^{tdm} \leq B_{\vec{r}}^{path} \tag{2.28}$$

Additionally to B^{path} there is another upper bound available for TDM schedules. It is the hardware bandwidth B_i^{hop} of the link from the NI of node n_i to the NoC and thus describes the maximum bandwidth that a node n_i can serve to handle the schedule.

$$B_i^{hop} \geq \sum_{\{\vec{r} \mid \vec{r} \in tdm^R \wedge \vec{r}.ph_{first}^{src} = n_i\}} B_{\vec{r}}^{tdm} \tag{2.29}$$

In contrast to the stated upper bounds $B_{\vec{r}}^{path}$ and B_i^{hop}, Equation 2.27 describes the dominating factors of the TDM bandwidth of a route $B_{\vec{r}}^{tdm}$. Hence, a typical design goal for a route $\vec{r}$ of a TDM schedule is to maximize $B_{\vec{r}}^{tdm}$, while considering the requirements for the other routes of the schedule as well. Thus, an optimization of a TDM schedule typically targets to shorten the period T and to maximize the number of flits during one TDM cycle. Thereby, reducing the period increases the bandwidth of all routes $\vec{r} \in tdm^R$ of the schedule at the same time. On the other hand, varying the number of paths for the routes in the schedule additionally enables the possibility of weighting the routes in terms of granted bandwidth.

When looking at a concrete node, sending a message consists of multiple send operations (one operation per flit). Thereby, it is not specified how these operations are implemented.

Instead, it depends on the inner structure of each core $c \in C$. For example it may be implemented in hardware when using a DMA [Goo+17] or in software using an assembler instruction for sending each flit [Mis+17]. The same holds true for receiving a message and receive operations. If the send operation is executed for a flit, it is ejected to the NoC by the NI whenever an appropriate slot is available. When a flit reaches its targeted node, it is stored to a dedicated memory that is managed by the NI (e.g. a dedicated receive buffer). It remains there until the core $c \in C$ connected to the NI executes a corresponding receive operation. At that time the flit is taken from the receive buffer and is handled according to the executed operation (e.g. stored in local memory).

As mentioned above, except some special variations like the one described in [SKS13] (see below), the data of a flit is not directly inserted to the NoC when the send operation is executed. Instead, it remains locally at the sending node until the NI releases the data according to the applied TDM schedule. Subsequently, according to [MU14] and our work [Ste+18] this data is called to be in send state and the time a flit remains in this state is called its admission time t^a. The time a flit needs to travel through the NoC is called the transportation time t^t. This time starts when the flit leaves the sender's NI and ends at the point in time when the flit enters the NI of the receiver node. Similar to the send state there exists a receive state, too. Data in that state is already delivered to the receiver node, but the corresponding receive operation has not been executed yet.

In the special cases when the DMA is directly contained in the NI and it additionally covers the functionality of releasing flit at the correct time according to the TDM schedule (like in [SKS13]), the send operation and release to the NoC takes place at the same time. However, since the execution of those operations are triggered by the TDMA mechanism we see in those cases the admission time t^a as the period from releasing the previous flit to releasing the actual flit of a message. Nevertheless, the first flit of a message considers the core's operation that triggers the DMA as a send operation.

According to the applied TDM schedule there may be several paths with the same route in one TDM cycle. Thus, there may be multiple time slots $s^{t,i}$ available for sending flits from a source node n_i to its target node n_j along a route $\vec{r} \in \vec{r}_{i,j}$. Therefore, the admission times are defined as follows.

Definition 2.32. *An admission time $t^{a,x}_{i,j}$ denotes the admission time of the x-th time slot $s^{t,i}_x$ for a route $\vec{r} \in \vec{r}_{i,j}$ of the TDM schedule. If the mapping to a concrete time slot is not needed, the symbol $t^a_{i,j}$ can be used for the admission time.*

The value of admission time $t^{a,x}_{i,j}$ is determined by the time passing between the end of the previous time slot $s^{t,i}_{x-1}$ and the end of the current slot $s^{t,i}_x$. In addition to single instances of admission times we need to define a set for all admission times of the routes in $\vec{r}_{i,j} \subset tdm^R$ in a TDM schedule.

Definition 2.33. *The set of all admission times $t^{a,x}_{i,j}$ of a route $\vec{r} \in \vec{r}_{i,j}$ in a TDM schedule is denoted as $t^A_{i,j}$.*

In addition to the view on admission times based on time slots, there is the need for stating the admission time based on a specific flit of a message. Thus, we give the following notation of admission times, too.

Definition 2.34. *An admission time $t_{i,j}^{a,msg.x}$ denotes the admission time of the x-th flit of a given message which is sent along a route $\vec{r} \in \vec{r}_{i,j}$ of the TDM schedule.*

Similar to the admission times the transportation times are stated as follows.

Definition 2.35. *A transportation time $t_{i,j}^{t,x}$ denotes the transportation time of the x-th time slot $s_x^{t,i}$ for a route $\vec{r} \in \vec{r}_{i,j}$ of the TDM schedule.*

The value of $t_{i,j}^{t,x}$ is determined by the path latency $L_{\vec{r}}^{path}$ of the route used for the current time slot $s_x^{t,i}$. Furthermore, we define a set for all transportation times of the routes in $\vec{r}_{i,j} \subset tdm^R$ in a TDM schedule, too.

Definition 2.36. *The set of all transportation times $t_{i,j}^{t,x}$ of a route $\vec{r} \in \vec{r}_{i,j}$ in a TDM schedule is denoted as $t_{i,j}^{T}$.*

In the area of HRT systems we always need to consider the latencies exhibited in the worst case. This procedure also holds regarding the admission and transportation times. Since it is hard to determine the exact point in time when a flit is inserted to the NI, we are forced to assume the point in time that maximizes the admission time. Therefore, the point in time directly after a suitable time slot is assumed for that, as this point exhibits the longest period of time until a subsequent slot is suitable for the regarded flit. There is a similar consideration for the transportation time. If there are multiple durations possible for transportation (e.g. if there are multiple routes to a destination) and it is impossible to figure out the one that is currently used, we need to assume the largest one. Hence, if not stated otherwise in the remainder of this work we always refer to the admission and transportation times of the worst case.

We call the time a flit stays in a NI and the NoC the traversal time of that flit. This time is composed of the admission time t^a, which represents the time in the NI, and the transportation time t^t, which describes the time in the network. Furthermore, since we talk about the admission and transportation times in the worst case, the traversal time is applied for the worst cast, too. Hence, it is called worst-case traversal time *(WCTT)*.

$$WCTT = t^a + t^t \tag{2.30}$$

Subsequently, it follows an introduction to two different categories of static TDM schedules. Firstly, we present application specific schedules and afterwards general-purpose schedules are presented.

2.2.1 Application Specific TDM Schedules

Application specific TDM schedules are applied for chips that are designed to executed a specific set of applications during its complete live cycle. Since an application has different communication demands in terms of bandwidth for different communication routes, this can be respected by TDMA. If a higher bandwidth is needed for a given route, the TDM schedule exhibits more paths along that route within one TDM cycle than other scheduled routes do.

Typically, such schedules are produced by applying the following procedure, which is presented in [Jal+08; Mar+09; Sør+14]. Firstly, the target applications must be analysed to

identify the requirements of all the applications' communication among processes. Thereby, especially the bandwidth requirements are of interest. This analysis leads to the routes needed for communication and each one is associated with a required bandwidth. Next, the obtained bandwidths are used to calculate the amount of data to be send along a route over the NoC. The calculated amount of data directly corresponds to the number of time slots that a node is allowed to use per TDM cycle. This design step is marked by choosing the period to be large and determining the number of paths of a specific route of the TDM schedule. Thereby, the number of paths should be small but still reflecting the ratio of the desired bandwidths. At this point the communication requirements are mapped to the schedule and all the paths included in the schedule are known. The next challenge is the arrangement of all paths in a way that the schedule is conflict-free and a concrete and preferably low period T can be given. This arrangement of paths is an optimisation problem that is known to be NP-complete [EIS75]. Nevertheless, there are heuristics available that solve this problem in an appropriate way [Pop+04; HE05; Sør+14; Har+18]. At the end of the schedule's construction there is a conflict-free TDM schedule that offers a low period.

With the described procedure it is possible to highly optimize TDMA to a specific application and to maximize the NoC's possible utilization for the considered application or set of applications. The downside of that optimization is a loss in flexibility [GMC09; MU14; Ham+18]. If there is a change in the executed applications the TDM schedule is possibly not optimal any more. In contrast, there may arise cases where high communication demands meet low bandwidth guarantees.This is a problem especially if the system is already in field and the TDM schedule is implemented in hardware. In those cases the planned changes would lead to a significant loss in performance and in the worst case up to a situation that the system is not usable any more. Thus, the restrictions of the actual TDM schedule would hinder the adaptation of the software. This issue is a major drawback of static TDMA for hardware that is not restricted to the execution of the same application over its complete system live cycle.

Changes in applications that cause such imbalances in performance are of manifold reasons. One possibility is that an application running on the considered architecture either is adapted, or its functionality is extended or updated [BDM02; Fel+07]. With those changes the communication demands often change, too. Another reason could be a change in the placement of tasks to actual processing elements aka nodes. Obviously, the communication demands of nodes change likewise to the modification of the mapping from software tasks to the nodes of the hardware architecture. The reason for a change in placement could be an optimization of chip utilization or the avoidance of using defect parts of the hardware [Ren04]. Finally, a chip may be used to not only execute one application but a composition of multiple applications in order to build a multi-mode system [SPT09]. In that case the communication demands of parts of the system change if the application mode is changed, or an application is added or removed to or from the current execution.

2.2.2 General-Purpose TDM Schedules

As already mentioned the characteristics of application specific TDM schedules highly restrict the adaptability of applications due to bandwidth and latency requirements. Thus,

the question arises if there can be found a trade-off between performance (or efficiency) of a TDM based NoC and composability (or adaptability) for software running on that hardware. One promising approach in that direction is introduced by Brandner and Schoeberl [BS12] and is extended by Mische and Ungerer [MU14]. Both authors propose to use so-called generic or general-purpose TDM schedules that provide equal bandwidth and latency guaranties to all nodes within the NoC.

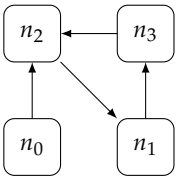
Generally, there are two ways to construct a general-purpose schedule. One solution is to in fact produce an application specific schedule but consider an equal number of paths from each node to each other node on the regarded NoC [BS12]. As the timing of the schedule shall not depend on task placement all routes are equally sized regarding to their guaranteed bandwidth. Typically, this way leads to a truly conflict-free TDM schedule.

The other option is to apply relaxed conflict-freedom and additionally restrict the incoming and outgoing number of flits of each node within one period [MU14]. To interpret this policy to a traditional TDM schedule, one can say that these schedules provide the same amount of paths in each TDM cycle as the schedule in [BS12] but only a subset of the paths are allowed to use in each TDM cycle to be able to reduce the period T. Thus, in this case additional constraints are needed to ensure conflict-freedom of the schedules.

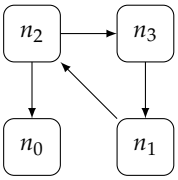
The variant of general-purpose schedules described first is introduced by Brandner and Schoeberl [BS12] and they propose a schedule where each node on a NoC is allowed to send and receive a message of equal size to and from each other node within one TDM cycle. Because of its characteristics this schedule is called an *AA (All-to-All)* schedule.

The second variant is presented by Mische and Ungerer [MU14]. They provide four different schedules constructed in the described way. The schedules are called *1A (One-to-All)*, *A1 (All-to-One)*, *11 (One-to-One)* and the fourth schedule is again the All-to-All schedule. The major characteristics of these schedules are their constraints about the number of flits sent and received per node in one TDM cycle. Mische and Ungerer describe implementation examples of the schedules using a uni-directional torus consisting of $n \times n$ nodes as basis. They assume strict dimension-ordered routing of flits starting in horizontal direction and a hop latency L^{hop} that is equal to the slot latency $L^{slot} = 1\ clock\ cycle$ for each physical link. In the remainder of this work for simplicity we call the routers where a flit performs the change in direction the corner router of that flit or path, respectively. Furthermore, the authors split each TDM cycle in a constant number of so called rounds with an equal size of $n \cdot L^{hop} = n\ clock\ cycles$ (except for the All-to-All schedule). The execution of paths is divided in two phases, also of equal size. Thereby, the first phase is executed during the round in that the regarded path begins and performs the horizontal delivery through the torus, while the second phase is executed in the subsequent round and is responsible for the vertical delivery. The characteristics of the schedules are presented below. The descriptions mainly follow the one of [MU14] and are rather informal. In Chapter 4 we extend those TDM schedules with formal definitions following our system model defined previously in this chapter. Figure 2.9 illustrates example communications of each of the described TDM schedules.

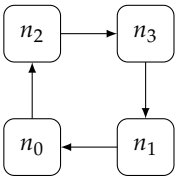
The One-to-All schedule [MU14] allows each node to send at most one flit per TDM cycle but each node may receive one flit from each other node (see Figure 2.9a). It consists of n rounds and in the r-th round the nodes are allowed to send a flit to a target node with a



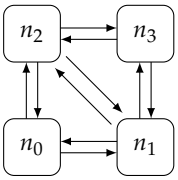
(a) Example communication of the One-to-All schedule. Each node is allowed to send at most one flit per TDM cycle and to receive one flit from every other node of the NoC.



(b) Example communication of the All-to-One schedule. Each node is allowed to send one flit to each other node but to receive only one flit within a TDM cycle.



(c) Example communication of the One-to-One schedule. Each node is allowed to send at most one flit per TDM cycle and equally receive only one flit within a TDM cycle.



(d) Example communication of the All-to-All schedule. Each node is allowed to send one flit to each other node and to receive one flit from every other node of the NoC within one TDM cycle.

Figure 2.9: Example communication for the presented general-purpose schedules. Each subfigure shows an example for a different schedule that is allowed to execute within one TDM cycle. The rectangles with rounded corners mark participating nodes and include their name (n_0 to n_3). The arrows indicate the delivery of one flit from a node n_i to a node n_j.

horizontal distance of r ($r \in [0, n-1]$). Since each node is allowed to send at most one flit per TDM cycle, after n rounds all flits are injected to the NoC. When a flit arrives at its target column it is stored in the corresponding router's buffer and waits until the next round starts. Afterwards they are delivered along the vertical ring. In each horizontal and vertical ring of the torus there are n nodes and the forwarding from one node to the next takes a hop latency L^{hop} of one clock cycle. Since all flits are injected to a ring at the beginning of a round, they are delivered to their target node or buffer within the same round. Hence, no conflict can occur.

The counterpart to One-to-All is the All-to-One schedule [MU14] which allows each node to send one flit to each other node but restricts the total number of sent flits by allowing each node to receive at most one flit in a TDM cycle (see Figure 2.9b). In each round r each node is allowed to send at most n flits that have destination nodes with a vertical distance of $n - r$. Due to the mentioned restriction there are at most n flits per horizontal ring and round and each one targets another corner router. Thus, the flits can be injected to the horizontal ring in a way that each flit reaches its target column exactly at the end of the paths' first phase. Since each node can receive only 1 flit per TDM cycle there is at most one flit stored in each router per round. At the beginning of the second phase the flits are injected to the vertical rings. Thus, they reach their target node not later than the end of the second phase. After

performing n rounds the schedule is executed completely.

The One-to-One schedule [MU14] restricts the communication by allowing each node to send at most one flit and equally receive at most one flit per TDM cycle (see Figure 2.9c). This schedule consists of only one round and the delivery is subdivided in two equally sized phases. At the beginning of a round in the first phase each node can send one flit along the horizontal ring to an arbitrary destination node as long as the constraint about receiving per TDM cycle holds. In the first phase the flits travel to their target column, where they are stored in the corner router's buffer. As all flits are inserted at the beginning of the phase no conflict can occur and all flits reach their target column no later than the end of the round. Hence, at the beginning of the second phase there are at most n flits stored in each router and vertical ring. These flits are already in the target column and have to travel at most n nodes along the vertical ring to their destination nodes. To ensure the absence of conflicts they are injected in a way that all flits reach their target node after exactly $n \cdot L^{hop}$. That means the time for injection to the vertical ring is defined by the corner router's distance to the target node. Thus, with a maximum distance of n nodes the time for injection is defined by $(n - \Delta y) \cdot L^{hop}$ where Δy marks the vertical distance from the corner router to the destination node. Following this procedure it is ensured that there is a free slot at the injection time for each flit in the considered ring and no conflict can occur.

Finally, the All-to-All schedule of [MU14] allows each node to send and receive one flit to and from each other node within one TDM cycle (see Figure 2.9d). Similar to the other schedules a TDM cycle consists of multiple rounds. However, with a length of $n^2 + n$ cycles the rounds of the All-to-All are longer than in the other schedules but it needs with $\frac{n}{2}$ rounds per TDM cycle a fewer number of rounds. Please note that the last round is only a half round if n is odd. Another difference is that the phase for vertical delivery already starts during the horizontal delivery instead of waiting for the begin of the next round. All nodes in the NoC send the flit with the same horizontal and vertical distance from the source to the destination node at the same point in time. Furthermore, the source nodes' flits with an equal horizontal distance are send successively by each node. That means that a node sends all n flits with the same horizontal distance before it performs the sending of flits with another distance. As all nodes send flits with the same horizontal distance Δx at a given point in time, after $\Delta x + 1$ cycles all those flits arrive at their corner routers. Thus, at that point in time the NoC is completely free to deliver the next set of flits and after $n \cdot (\Delta x + 1)$ clock cycles all flits with the same horizontal distance are delivered to their corner routers. For each round we combine the delivery to nodes with a horizontal distance of Δx and the one to nodes with distance $n - 1 - \Delta x$. That procedure leads to a length of the round that is independent of the distance Δx and results in $n(\Delta x + 1) + n(n - 1 - \Delta x + 1) = n(n + 1)$ clock cycles. Since there are $2n$ flits sent per round and node and the complete amount of flits per node is $n^2 - 1$, one TDM cycle consists of $\frac{n}{2}$ rounds ($\frac{n^2-1}{2n} \leq \frac{n}{2}$). The vertical delivery starts at the moment when the flits of the first part with horizontal distance Δx of the first round completely arrived at the corner routers. The procedure of vertical forwarding is analog to the horizontal case since it alternates between transporting one flit with a long distance $n - 1 - \Delta y$ and one flit with a short distance Δy. The vertical delivery of a flit takes the same amount of time as the horizontal delivery. Thus, it needs as many clock cycles as the length of the current distance plus one extra clock cycle. Therefore, each pair of flits with

a long and a short distance takes $(n-1-\Delta y+1)+\Delta y+1 = n+1$ clock cycles. Since there are $\frac{n}{2}$ pairs needed the vertical delivery of one round takes $\frac{n(n+1)}{2}$ clock cycles.

General-purpose schedules differ from application specific schedules in that they are not built for a specific application but provide equal guarantees for communication from each node in a NoC to each other node. Therefore, they are independent of the actual task placement in terms of bandwidths and latencies of applications running on the system [MU14; Ste+16]. This has several consequences to the characteristics of a system when applying it in field. One consequence is that an adaptation or update of an application does not lead to a recalculation of the TDM schedule, as all communication demands are handled in the same way. Furthermore, there is no need to reserve capacities over the whole system life cycle for communication demands of all planned applications or application modes in multi-mode systems. Finally, changing the placement of tasks during runtime does not lead to different timing behaviour for the worst case. However, these advantages typically come to the price of higher latencies and lower bandwidths.

In contrast to application specific schedules the schedules All-to-All, One-to-All, All-to-One and One-to-One limit the traffic of all nodes to an amount that enables a statement about the upper bound without considering the communication of other independent applications running on the same chip. When additionally assuming the same worst-case transportation time t^t for all pairs of source and target nodes, these schedules guarantee a *worst-case traversal time (WCTT)* which is independent of the position of participating nodes in the NoC. Hence, no complex method for an optimal placing of tasks is needed for optimizing latencies.

2.3 Message Passing Interface

We see MPI [MPI15] as de-facto standard for message-passing in distributed memory systems. Thus, we consider a basic subset of MPI that targets the execution of applications on NoCs to provide methods for the analysis of programs based on that standard. Thereby, we consider the network is free of hardware faults.

The MPI standard targets a parallel programming model which is based on message passing. With that model a parallel application consists of several processes where each process operates on its own address space. Thereby, moving of data from one process to another process is realized by executing operations on both processes that cooperate with each other.

Definition 2.37. *A process $v \in V$ is an execution instance of a program that runs on a single processing element $c \in C$. Each process runs with its own address space (memory context) and uses its own hardware context (registers, program counter and stack pointer). The symbol V denotes the set of all processes and the term v_i indicates the i-th process.*

The operations which are executed for the communication among the processes are standardized in the MPI specification and thus, MPI is constructed as a library interface specification [MPI15, p.1f.]. Please note that MPI is a specification and does not provide a concrete implementation for message passing. Moreover, the programming model considered for MPI does not require a specific hardware model except for executing multiple processes and the ability to communicate with each other. Hence, it is suitable for shared

memory systems as well as for distributed memory systems [MPI15, p.5]. As stated in Section 2.1 in the remainder of this work we focus on distributed memory systems connected via a NoC. The interface specification of MPI mainly targets the programming languages C and Fortran but is not restricted to them. In this work we concentrate on the binding to the C language. Some example implementations for C/C++ are MPICH [Tea19], OpenMPI [Gab+04; Tea16], MSMPI [Cor18] and others. Altogether, there are several implementations of the standard available for different programming languages [Haf+11]. All MPI operations can be implemented as functions which supports the portability of the standard and eases its use as a library [MPI15]. In the remainder of this section we address the specifications of MPI as well as some possible implementations that are considered in this work. Thereby, all implementations are based on state-of-the-art implementations of the MPI standard [Gab+04] and are adapted to fit for the purposes of this work.

On the one hand MPI defines basic point-to-point communication between two processes. Typically, in this communication one process sends a message to one other process by invoking a send operation, while the receiving process calls a receive operation. A variant of the point-to-point communication is so-called one-sided communication. Here the communication takes places between two processes as well. However, with that kind of communication only one of the processes is active and calls the according operation. The other process is passive when the communication takes place and may not even recognize that the communication is performed. Typical actions with that communication are to put data in the address space of another process or to get data from this process. In both cases the remote process does not know that this communication takes place, if no additional synchronization is performed. Since this communication implies the usage of the NI of the cores without the knowledge of the process running on that core, there would be additional hardware needed to prevent the receive buffer of the NI from overflowing. Moreover, if the passive process is additionally performing an active communication at the same time when the one-sided communication takes place, the NI of the according node must be seen as a shared resource between both communications. However, in this work we try to hinder communications to use such shared resources to provide timing bounds for the communications that are independent of the surrounding application and hence, from other communication that is performed on that application and chip. Therefore, we the one-sided communication out of scope of this work.

Besides simple point-to-point communications, there are also more sophisticated communication operations available. One class of those operations is called collective communication. These operations target groups of two or more processes and are typically used to collect, distribute or exchange data from, to or among all participating processes. Another class of sophisticated communication consists of operations that exchange data according to a given logical topology that is defined on the group of the participating processes. Thereby, each process only interacts with its direct neighbours of the topology. Thus, this kind of communication is also called sparse or neighbourhood collective communication [HLL08; HT09; Hoe+11; HS12]. In the remainder of this work we address point-to-point communication as well as collective communication and let neighbourhood collective communication to future work.

To enable the mentioned kinds of communication in a parallel application, each process

must be identifiable in an equal way for all other processes. Furthermore, there is a mechanism needed to group sets of processes that are intended to exchange data with each other and to provide topologies to those groups. In MPI the mechanism that encapsulates all these informations is called a communicator [Fei91; SL90; Skj+94]. This mechanism exists in the form of an intra-communicator and an inter-communicator. An intra-communicator handles the communication of processes within a particular group, while the inter-communicator is used for communication between different groups. In the remainder of this work we focus on intra-communicators and let the timing analysis of communication based on inter-communicators to future work.

Intra-communicators consist of different elements like a group, a process topology and attributes that may facilitate for example the mapping of processes to processing elements. Nevertheless, we only focus on the group element, as this is the relevant part of a communicator for the rest of this work.

Definition 2.38. *A group $V^g \subset V$ is a set of processes $v \in V$ that participate in the same communicator. The set of all groups of an application is denoted as V^G. The size of a group is given with $|V^g|$.*

As stated in Definition 2.38, a group is a collection of processes and defines the participants in the communication of a communicator. [MPI15, p.223ff.] Thereby, all processes of a group are subject to a total ordering in a way that each process gets a number that is unique within the group. This number is called the rank of the process and is used to identify a particular process within a group to provide point-to-point communication between two dedicated processes for example. For collective communication a group defines the participants of the communication. That means that when a collective communication is performed based on a communicator, all processes included in the communicator's group must participate in that communication and thus, call the according operation. All groups and communicators base on a special group and communicator in a way that they are subsets of this group and communicator. These special variants contain all processes of a MPI program and are typically called *group world* and *comm world*.

A communicator builds the context for a communication among processes and provides the information needed for a communication to each participating process. Each process of a communicator can communicate with each other process of the same communicator. Thereby, we refer to an actual communication that takes place in a communication operation as an communication edge that is defined as follows.

Definition 2.39. *A communication edge $e^c_{src,dst} \in E^c$ is a 4-tupel $(v_{src}, v_{dst}, o^c_{src}, msg)$ including the processes $v_{src}, v_{dst} \in V$ that mark a directed connection from v_{src} to v_{dst} and an ordering o^c_{src} which indicates the point in time relative to other edges of the process v_{src} when this edge is considered as active for sending a flit of the message msg. E^c denotes the set of all edges in a communication pattern. Furthermore, we define $e^{c,src}_{i,j}$ as the source process v_i of a communication edge $e^c_{i,j}$ and $e^{c,dst}_{i,j}$ as its corresponding destination process v_j.*

Building on the previous definitions we give the global view on the communication that actually takes place during a communication operation. We call that view the internal communication pattern and define it as follows.

Definition 2.40. *An internal communication pattern $cp \in CP$ is a pair (V^g, E^c) consisting of a set of processes V^g and a set of communication edges E^c. Thereby, each communication edge $e^c_{i,j} \in E^c$ directly connects two processes $v_i, v_j \in V^g$ and marks an actual communication when this pattern is executed.*

The communication patterns in CP form graphs in a way that each communication edge $e^c_{i,j}$ of the pattern symbolizes a point-to-point communication between the two processes v_i and v_j which are included in the pattern's set of processes V^g. Since each edge e^c is annotated with an ordering o^c the relative order of the according point-to-point communications is defined. Subsequently, it is necessary to reason about the internals of the communication patterns. To facilitate this reasoning we define two sets of processes that include all processes that act as children or parents of a regarded process v_i, respectively.

Definition 2.41. *The set $V_i^{cp,in} \subset cp.V^g$ denotes the set of all processes in a communication pattern cp that are parents of process $v_i \in cp.V^g$, aka the set is determined by Equation 2.31.*

$$V_i^{cp,in} = \{v_j \mid v_j \in cp.V^g : \exists e^c_{j,i} \in cp.E^c\} \tag{2.31}$$

Definition 2.42. *The set $V_i^{cp,out} \subset cp.V^g$ denotes the set of all processes in a communication pattern cp that are children of process $v_i \in cp.V^g$, aka the set is determined by Equation 2.32.*

$$V_i^{cp,out} = \{v_j \mid v_j \in cp.V^g : \exists e^c_{i,j} \in cp.E^c\} \tag{2.32}$$

For the same reason we define the sets of communication edges that connect a process with its parents and children as follows.

Definition 2.43. *The set $E_i^{cp,in} \subset cp.E^c$ denotes the set of all communication edges of a communication pattern cp that connect the regarded process v_i to a parent process $v_j \in cp.V^g$ and the communication follows the direction from v_j to v_i. Hence, the set is determined by Equation 2.33.*

$$E_i^{cp,in} = \{e_{j,i} \mid e_{j,i} \in cp.E^c : \exists v_j, v_i \in cp.V^g\} \tag{2.33}$$

Definition 2.44. *The set $E_i^{cp,out} \subset cp.E^c$ denotes the set of all communication edges in a communication pattern cp that connect the regarded process v_i to a child process $v_j \in cp.V^g$ and the communication follows the direction from v_i to v_j. Hence, the set is determined by Equation 2.34.*

$$E_i^{cp,out} = \{e_{i,j} \mid e_{i,j} \in cp.E^c : \exists v_j, v_i \in cp.V^g\} \tag{2.34}$$

In the remainder of this work we need to traverse the processes and communication edges of a communication pattern. Hence, we define a term pattern path to indicate a particular path along processes of edges in a communication pattern.

Definition 2.45. *A pattern path $pp \in PP^{cp}$ is a ordered list of edges $pp = \langle e^c_{m,i}, e^c_{i,j}, \cdots, e^c_{l,n} \rangle$ that define a connected path through a communication pattern $cp \in CP$. The symbol PP^{cp} denotes the set of all pattern paths of the communication pattern cp and the symbols $e^c.pre$ and $e^c.post$ are used to refer to the communication edge that is located directly before or after the edge e^c in pp. Each correct pattern path holds Condition 2.35.*

$$\begin{aligned} \forall e^c_{i,j} \in pp: \quad & (e^c_{i,j}.pre = \emptyset \vee (e^c_{i,j}.pre.v_{dst} = e^c_{i,j}.v_{src} \wedge e^c_{i,j}.pre.o^c < e^c_{i,j}.o^c)) \\ & \wedge (e^c_{i,j}.post = \emptyset \vee (e^c_{i,j}.post.v_{src} = e^c_{i,j}.v_{dst} \wedge e^c_{i,j}.o^c < e^c_{i,j}.post.o^c)) \end{aligned} \tag{2.35}$$

In addition to Definition 2.45 we provide the symbol pp^v that represents an ordered list of processes $pp^v = \langle v_m, v_i, \cdots, v_n \rangle$ that are connected via the pattern path pp. Thus, pp^v is ordered in the way the processes are connected by the communication edges $e^c \in pp$ of the communication pattern cp.

As a process v_i may send a message to multiple processes at the same time ($|V_i^{cp,out}| > 1$), we must respect that there are flits sent to multiple processes. The same holds true for receiving flits from multiple processes if $|V_i^{cp,in}|$ is larger than 1. Thus, we need a possibility to refer to sent or received flits in more detail than just referring to a specific flit in a message ($msg.fl$). The following functions are defined for that reason.

Definition 2.46. *The function $snt_i^{cp} : (MSG, cp.V^g) \rightarrow \mathbb{N}, (msg, v_j) \mapsto n$ provides the number of flits of a message $msg \in MSG$ that are already sent to a process $v_j \in cp.V^g$ when the last flit of the message $msg.fl_{last}$ is intended to be sent next. Thereby, for v_j it must hold that $\exists e^c_{i,j} \in cp.E^c$.*

According to Definition 2.46 the function $snt_i^{cp}(msg, v_j)$ returns the number of flits that are already sent to the process v_j when the last flit of the message msg is about to send. In contrast to referring to the length of the message $|msg|$, it reflects the situations when there are different numbers of flits sent to different child processes or when the message is distributed among all children which causes the regarded process v_i to send only a fraction of the complete message to each child process v_j.

Similar to sending to multiple processes the number of flits received from multiple processes may differ. Hence, we provide an according function for this case, too.

Definition 2.47. *The function $rcd_i^{cp} : (MSG, cp.V^g) \rightarrow \mathbb{N}, (msg, v_j) \mapsto n$ provides the number of flits of a message $msg \in MSG$ that are already received from a process $v_j \in cp.V^g$ when the last flit of the message $msg.fl_{lst}$ is intended to be received next. Thereby, for v_j it must hold that $\exists e^c_{j,i} \in cp.E^c$.*

Please note that the functions *snt* and *rcd* can target any intermediate flit of a message if the according part of the message $msg^{part} \subset msg$ is used as parameter.

2.3.1 Point-to-Point Communication

The group of point-to-point communications basically consists of two kinds of communications. The one communication is in fact sending data from one process to another process and the other communication is about sending data to a process while receiving other data

(a) explicit *send* and *receive* operation (b) combined *send-receive* operation

Figure 2.10: Communication patterns $cp \in CP^{P2P}$ that deliver data according to point-to-point communications. The rectangles indicate processes $v \in cp.V^g$ that participate in the communication and the arrows mark communication edges $e^c \in cp.E^c$ that are used as the basic blocks of the communication pattern.

at the same time. The communication pattern used for those communications are grouped in the set CP^{P2P} and are displayed in Figure 2.10.

The point-to-point communication where one process sends data and one other process receives this data is a basic mechanism provided with MPI. This kind of communication is based on a communicator which provides the communication context for the data exchange. Thereby, both involved processes must be included in the applied communicator but there are also further processes that do not participate in the communication allowed to be in that communicator.

The processes trigger the communication by calling a *send* or a *receive* operation (naming convention in MPI: `MPI_Send`, `MPI_Recv`), depending if the process acts as sending or receiving process (see Figure 2.10a). A *send* operation gets an area in memory as parameter that is used as send buffer within the operation. Hence, the data stored in that memory contains the data that is intended to be delivered to the receive process. On the other side of the communication the *receive* operation that is called by the receiving process gets an area of memory that is used to locally store the received data and is therefore called the receive buffer. Furthermore, both operations get the rank of the other process according to the used communicator to send the message to the correct target process or to listen for the correct message, respectively. Moreover, stating a tag value is possible to distinguish between different messages of the same sender and receiver processes.

Basically, it is possible to trigger the communication as blocking or as non-blocking communication. Thereby, blocking communication causes the according *send* and *receive* operations to not return until the send or receive buffer can safely be modified by the according send or receive process. In concrete this causes the send operation to copy away or directly send the content of the send buffer and do not return before all data is copied or sent. On the other hand the receive operation does not return until all received data are stored in the given receive buffer.

In contrast to the blocking version the non-blocking communication does not wait until the send and receive buffers are handled appropriately and thus are usable again. A non-blocking *send start* operation (naming in MPI: `MPI_Isend`) executes the send operation and than immediately returns to enable further execution of the program. At the point of returning to the program the content of the send buffer is not copied away. Thus, the program is not allowed to modify this buffer until a separate *send complete* operation (naming in MPI: `MPI_Wait`) is called, which verifies that the data is completely copied away from the send buffer. On receiver side there is a *receive start* operation (naming in MPI: `MPI_Irecv`) that initiates the receive operation. Again this operation returns immediately after initialization and the data is not stored in the receive buffer at that time. Only a call of a *receive complete*

operation (naming in MPI: `MPI_Wait`) verifies that the receive operation has completed safely and the data is fully stored in the receive buffer. The non-blocking communication variants are especially of interest if there is additional hardware that supports a concurrent execution of code and delivery of data.

Altogether, there are four different communication modes provided by the MPI standard for blocking and non-blocking communication except for the *send-receive* operation. These are called *standard*, *buffered*, *synchronous* and *ready* communication. In *standard* mode it is up to the implementation of MPI to decide if a outgoing message is copied to an extra local buffer in the sender process or not before it is released to the communication channel. When buffering is performed, the *send* operation (or *send complete* operation if non-blocking) can complete before an appropriate *receive* operation is invoked. On the other hand, in case of not using the buffering mechanism the *send* or *send complete* must wait for its completion until the according *receive* operation is called and the data is completely delivered to the receive process. Since the *send* operation may or may not use the buffering mechanism, a successful completion of the operation may depend on the occurrence of a matching *receive*. Thus, these operations must be considered as non-local for worst-case timing considerations.

The buffered communication mode forces the *send* operation to use the described buffering mechanism. Therefore, this operation can be started and also may complete before a matching *receive* operation is invoked. Hence, its completion does not depend on the occurrence of operations called on other processes and can be considered as local.

Like the previously described modes the synchronous communication mode provides a *send* operation that can be started before a matching *receive* operation was posted. However, the mode forces the *send* (or *send complete*) operation to only complete if the according *receive* operation has started to receive the message, which causes the send operation to be non-local. Thus, a completion of a *send* or *send complete* with this mode additionally guarantees that the program execution of the receiving process has reached the receive operation. Moreover, when the *send* and *receive* operations are both in blocking mode, the program execution on both processes does not proceed until both processes rendezvous at the communication.[MPI15, p.37ff.]

Finally, the ready communication mode allows a call of the *send* (or *send start*) operation only after a matching *receive* (or *receive start*) operation has been posted. If this requirement is not fulfilled, the operation is considered erroneous and its behaviour is undefined. The intention of this communication mode is to provide a mechanism that saves some overhead by skipping a hand-shake operation in certain systems. In concrete programs a *send* operation in ready mode behaves similar to the ones in standard or synchronous mode. Thus, it is considered to be non-local.

On a first glance the buffered communication mode seems to be convenient mode when targeting timing analysis, as it is a local operation that reduces effects on the timing of other processes and vice versa. However, we do not consider the buffered communication mode in the remainder of this work, as in this mode enough memory must be provided for each performed send operation. If there are several send operations each with a long message, the amount of memory that must be provided at the same time increases. Furthermore, if local memory is limited, the timing of these send operations depend on the performed delivery of previously sent data. Thus, in those cases the operations are not local any more.

Moreover, in those cases the availability of memory for a particular *send* operation depends on the context of the program execution which contradicts the goal of providing a timing behaviour for the communication that is independent of the concrete application.

Since the standard communication has the ability of applying an additional buffer at the sender process, it struggles with the same problems as the buffered communication mode. Moreover, as it is allowed the decide whether to use that mechanism or not according to the current situation, there is another uncertainty that complicates the timing analysis.

The ready communication mode requires that the receive operation is already invoked at the time when a send operation is called. Thus, there is an inherent dependency of the send operation from the timing of the complete execution of the receive process to run correctly. As this communication mode causes high effort during programming and possibly leads to undefined behaviour, we omit this communication mode.

According to the previously described advantages and disadvantages we focus on synchronous *send* and *receive* operations in the remainder of this work. Though the send operations in this communication mode are non-local and therefore, depend on the timing of the receive process, they do not need any additional buffering. Hence, there is no need to know the memory usage of communication performed previously to a particular *send* or *receive* operation. Thus, the timing and memory characteristics of such an operation can be determined without the knowledge of the program context in that the operation is executed.

Furthermore, we focus on blocked communication and leave non-blocking communication to future work. Our system model enables the usage of a DMA. Thus, the implementation of non-blocking communication as described in the MPI standard is principally possible with the assumed hardware system model. Our approach can also be used for the timing of a DMA and the procedure of sending and receiving within the DMA is similar to blocked synchronous communication. Thus, both types of communication can be mapped directly to the timing analysis of blocked synchronous communication. Nevertheless, compared to blocked synchronous communication there are a lot of additional circumstances to consider for non-blocking communication. An example of such issues is the presence of possibly unknown multiple communication partners at the same time. Hence, we let the analysis of non-blocking MPI communication for HRT systems to future work.

In addition to the normal send and receive operations MPI provides another point-to-point communication. This operation is called *send-receive* (naming in MPI: `MPI_Sendrecv`) and combines one *send* operation to one process with one *receive* operation from another process in one function call (see Figure 2.10b). Thereby, the source and destination processes of this operation are possibly the same but may also be different processes. Semantically, the operation causes the concurrent execution of one blocking *send* and one blocking *receive* operation. To be consistent to that semantic the messages sent or received with this operation can be handled at the partner processes with another *send-receive* operation or with regular *send* or *receive* operations.

2.3.2 Collective Communication

Collectives are communication operations that target the communication among a group of two or more processes. Therefore, each collective communication is based on a commu-

nicator that defines the group of participating processes. During its execution the data is distributed across or collected from the processes according to the type of the applied collective. The execution is only correct if each participant calls the intended collective operation. Thereby, the communication is organised using multiple point-to-point communications within that operation.

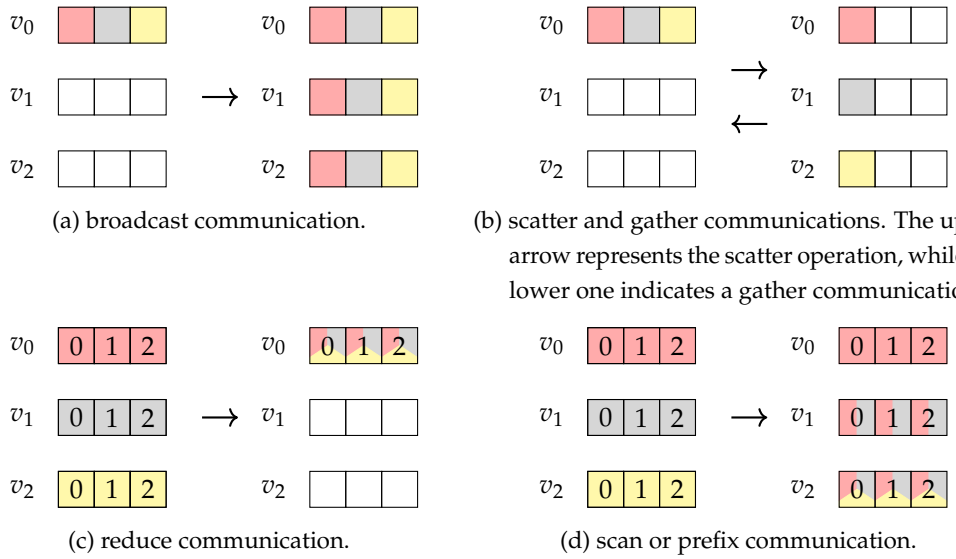
From the view of a single process the collective communication completes as soon as it has finished its participation to the communication. According to the standard the completion of a collective operation does indicate that the communication buffers used are free to modify again. It does not indicate any synchronization effects like all participating processes have finished or even started the communication as well [MPI15, p.142]. Similar to the point-to-point communication all collective communications can be executed as blocking or as non-blocking communication. Subsequently, we focus on blocking communication and leave the non-blocking versions to future work for the same reasons as with point-to-point communication.

Generally the collectives standardized in MPI can be classified in two groups with respect to the targeted communication. [MPI15, p.145] The first group aims to send or receive data to or from a single participant. Therefore, data of all participants are collected and sent to this particular attendee or vice versa. This group of operations includes MPI_Broadcast, MPI_Gather, MPI_Scatter, MPI_Reduce and their different variants like the non-blocking versions.

Figure 2.11 illustrates the performed data exchanges among the participating processes for each of those operations at an example with three participating processes. At the beginning of the broadcast communication only the root process holds data. During the communication these data are copied to the local memories of all participating processes (see Figure 2.11a). In the scatter communication at the beginning the root process holds all data that will be distributed over all processes during the communication. The gather operation behaves exactly vice versa to the scatter operation (see Figure 2.11b). At the beginning of the reduce communication each process holds the same amount of data. During the operation these data are collected and reduced by performing an element-wise mathematical operation. At the end the results are stored at the root process (see Figure 2.11c).

Finally, there is one collective communication that neither fits perfectly to the class with a special attendee nor to the second class described below. This collective is called MPI_Scan (also called *prefix*) and performs a communication like it is illustrated in Figure 2.11d. In that communication at the beginning each process holds the same amount of data. During the operation there is an element-wise mathematical reduction. At the end each process gets the results of the reduction of the data of all processes with a lower or equal rank than itself. Since all internally proceeded communications rely on the first process, this communication typically is implemented using a point-to-point communication pattern that is typical for the first class which exhibits a central process[Tea16]. Hence, though this kind of collective communication does not perfectly fit to the classification, we see this collective in the class with the special attendee.

In the second group there is no special attendee on which the communication is focused on. Rather, all nodes are equal with regard to the performed communication. This group of operations includes MPI_Barrier, MPI_Allgather, MPI_Alltoall (also called *complete exchange*),



(a) broadcast communication.

(b) scatter and gather communications. The upper arrow represents the scatter operation, while the lower one indicates a gather communication.

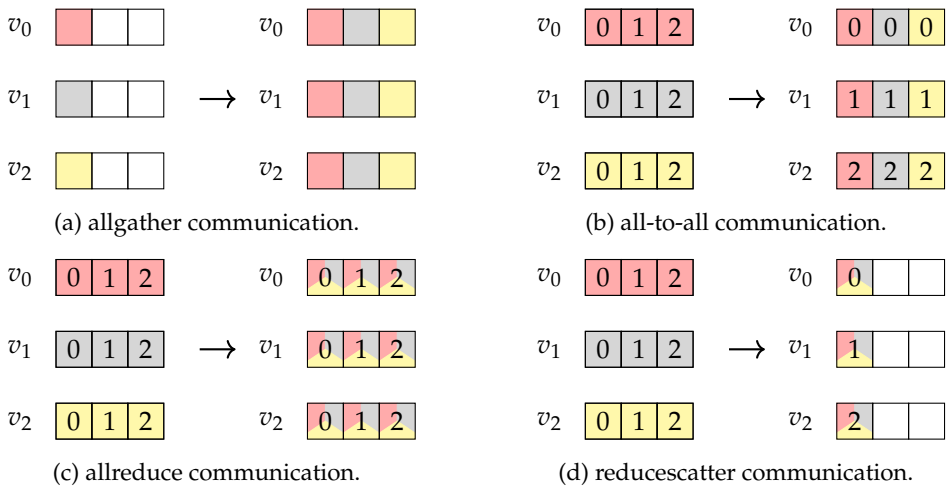
(c) reduce communication.

(d) scan or prefix communication.

Figure 2.11: The functionality of collective communications that are based on a central process. Thereby, each operation is illustrated by depicting the participating processes once before the communication is executed and once afterwards. The arrows indicate the transition of the situation beforehand to the situation afterwards. Each process is displayed with its symbol v_i and the area in local memory that is used to send or receive data, respectively. Theses memories are illustrated as arrays of rectangles containing three elements of data. The distinction of the elements is given by marking them with different colours and additionally assigning indices if needed. Elements that consist of multiple colours indicate that they are build by summarizing several elements with a mathematical operation. The original elements are determined by combining one of the colours and the given index. White elements mark that there is no data included in that element. The depictions are based on [MPI15, p.143].

MPI_Allreduce, MPI_ReduceScatter and their different variants like their non-blocking versions.

Figure 2.12 illustrates the performed data exchanges among the participating processes for each of those operations at an example with three processes. At the beginning of the allgather communication each process owns the some data. These data are collected similar to a gather operation and are then equally provided to all participating processes (see Figure 2.12a). Before an all-to-all communication is performed each process has the same amount of data. Then, during the execution the data of each process are distributed among all processes. Thereby, the ordering of data in memory is according to the ordering of the ranks of the processes (see Figure 2.12b). At the beginning of the allreduce collective communication each process has the same amount of data. During the operation the data are reduced with a mathematical operation like it happens in a reduce collective operation. However, instead of providing the result to only one process, it is provided to all participating processes (see Figure 2.12c). In the reducescatter collective communication each process holds the



(a) allgather communication.

(b) all-to-all communication.

(c) allreduce communication.

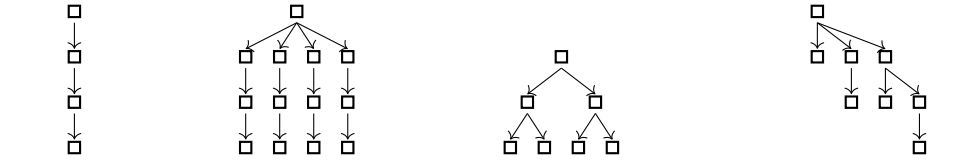
(d) reducescatter communication.

Figure 2.12: The functionality of collective communications that are based on uniform data exchange. Thereby, each operation is illustrated by depicting the participating processes once before the communication is executed and once afterwards. The arrows indicate the transition of the situation beforehand to the situation afterwards. Each process is displayed with its symbol v_i and the area in local memory that is used to send or receive data, respectively. Theses memories are illustrated as arrays of rectangles containing three elements of data. The distinction of the elements is given by marking them with different colours and additionally assigning indices if needed. Elements that consist of multiple colours indicate that they are build by summarizing several elements with a mathematical operation. The original elements are determined by combining one of the colours and the given index. White elements mark that there is no data included in that element. The depictions are based on [MPI15, p.143].

same amount of data before the execution starts. These data are reduced like in a reduce collective operation and afterwards distributed among all processes like in a scatter collective operation (see Figure 2.12d).

The implementation of collective communications can algorithmically take place in different ways, classified as unicast-based and path-based [McK+94; TPL96]. Oral and George [OG04] give an overview of several different unicast-based and path-based algorithms that are described in detail in [MTR95; RMC95; RMC97].

Path-based solutions rely on sending one flit to multiple destinations keeping a copy of the flit at each passed target node. These solutions require hardware extension for sending the flit to multiple destinations and copying it at the destination node. Since the system model considered in this work does not support such hardware (see Section 2.1), path-based routing algorithms are not considered subsequently. Unicast-based algorithms follow another approach. They compose several unicast messages to form a communication pattern cp and provide a broadcast communication for example. As they are based on unicast messages no additional hardware is needed and therefore, they are appropriate for the usage in the remainder of this work. Thereby, several algorithms exist for unicast-based collective



(a) pipeline (b) chains (c) binary tree (d) binomial tree

Figure 2.13: Communication patterns which deliver data in a tree like way. The illustrations show example patterns cp that are included in $CP^{tree}_{\downarrow}$ (e.g. for broadcast). The rectangles indicate processes $v \in cp.V^g$ that participate in the communication and the arrows mark communication edges $e^c \in cp.E^c$ that are used as the basic blocks of the communication pattern. Typically, they are implemented as a send or receive operation, depending if the process acts as a sender or a receiver.

communication based on a central process [RMC95; TG03; Hoe+04] and for uniform data exchange [Jin+05; Bru+97; TG03; TRG05].

According to the classification described above different communication patterns are used to effectively perform the distribution of data. The group obtaining a single central node forwards data in a tree like communication pattern, while the other group performs multiple steps of *send-receive* communication [TRG05]. To facilitate the referring the communication patterns of one class of communication we provide two sets of communication patterns. Thereby, the set CP^{tree} includes the pattern using a central process, while CP^{exch} consists of the patterns that support uniform data exchange. According to the regarded communication operation the tree-like patterns need to be constructed with different communication direction. In one direction the data is transported to the root process, while the other direction provides data movement to the leaf nodes. Regarding the direction of the communication edges $e^c \in cp.E^c$ must be defined accordingly. Hence, we further subdivided the set CP^{tree} in the subsets $CP^{tree}_{\uparrow}$ and $CP^{tree}_{\downarrow}$ ($CP^{tree}_{\uparrow} \cup CP^{tree}_{\downarrow} = CP^{tree}$). Thereby, $CP^{tree}_{\uparrow}$ includes all communication pattern that move data to the root process, while in $CP^{tree}_{\downarrow}$ the patterns see the leaf processes as the destination for data transportation.

In this work we examine five patterns of each group of collectives. All these communication structures are used in a state-of-the-art MPI implementation [Gab+04; Tea16] and hence, can be seen as relevant for providing an efficient data distribution. Tree like communication is performed as *pipeline* (cp^{pipe}), *chains* (cp^{chain}), a *binary tree* ($cp^{bintree}$) or a *binomial tree* (cp^{bmtree}). Conversely, the other group utilizes the *ring* (cp^{ring}), *neighbour exchange* (cp^{ne}), *recursive doubling* (cp^{rd}) and *bruck* (cp^{bruck}) patterns. Furthermore, both groups use the *basic linear* (cp^{bl}) pattern. Hence, we can state the following conditions.

$$cp^{pipe}, cp^{chain}, cp^{bintree}, cp^{bmtree}, cp^{bl} \in CP^{tree} \tag{2.36}$$

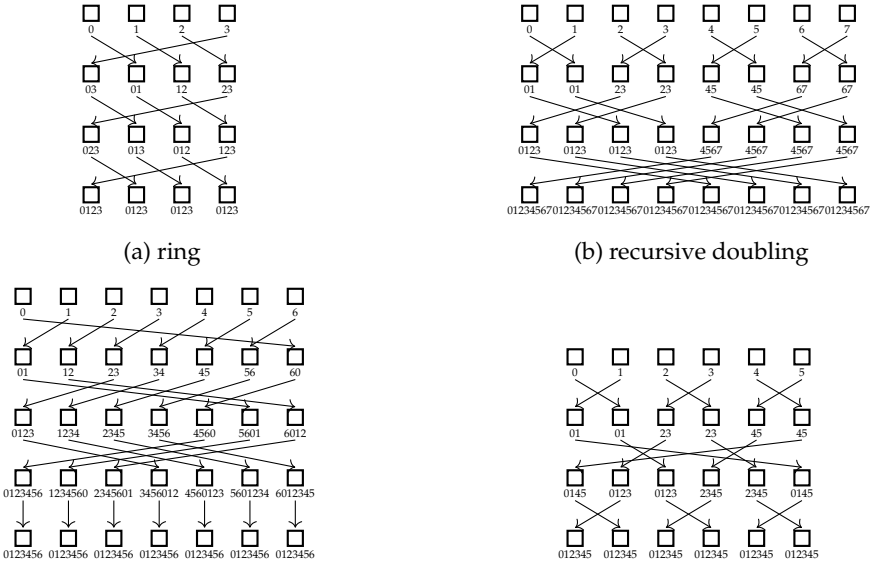
$$cp^{ring}, cp^{ne}, cp^{rd}, cp^{bruck}, cp^{bl} \in CP^{exch} \tag{2.37}$$

The MPI_Scan operation is a special case that does not fit to both classes. However, this communication can be implemented with the *pipeline* communication pattern.

In *basic linear* each sender node sends its data to all intended receiver nodes separately. This strategy often causes high latencies caused by the bottleneck at the root nodes. The

alternatives for tree based communication are displayed in Figure 2.13. The pipeline pattern (see Figure 2.13a) is characterized by forwarding data from one node to only one other node. If messages with multiple flits are send, it causes a transport of the flits in a pipelined fashion. The chains communication (see Figure 2.13b) behaves similar except that there are multiple pipelines in parallel which converge at the central sending node. This causes shorter pipelines than in the pipeline pattern but leads to a bottleneck at the root process. In the binary tree pattern (see Figure 2.13c) a node is connected to one parent and two children. Sending data along a binary tree means that each node send its received data to two children or sends the data received from its two children to its parent node, respectively. This distributes the bottleneck of the chains pattern to all participating processes and further reduces the length of the transportation. The binomial tree communication (see Figure 2.13d) traverses the data along a binomial tree. With that pattern the number of children of a process varies according to its position in the tree. Intuitively one can say that in this pattern the number of children raises with the distance to the farthest leaf process. Please note that according to state-of-the-art implementations [Gab+04] all tree patterns are constructed to be balanced at the beginning of a communication.

The communication patterns of the second group of collectives (except for basic linear) are shown in Figure 2.14. All of these communication patterns are based on multiple steps and per step each process delivers data to one other process. In the ring pattern (see Figure 2.14a) each process is sending data to its right neighbour until each data element is forwarded to each participating process. With neighbour exchange (see Figure 2.14d) the processes are forming pairs of two for communication in each step. Thereby, they alternate between their right and left neighbour. This communication is similar to the one of recursive doubling (see Figure 2.14b). However, a different policy is used for forming pairs, since in recursive doubling the distance between two communication partners doubles with each communication step. In doing so the number of needed communication steps can be reduced. There exists a variant of this pattern that is not displayed in Figure 2.14. Instead of starting with a low distance and doubling it for each step, at the beginning of that pattern the distances are maximal with half the number of participating processes and are halved in each step. This special variant is used in MPI_ReduceScatter for example [Tea16]. The bruck pattern (see Figure 2.14c) is some way similar to ring but changes the communication partners in each step by incrementing the communication distance. Bruck obtains a reduced number of communication steps, too.



(a) ring

(b) recursive doubling

(c) bruck

(d) neighbour exchange

Figure 2.14: Communication patterns for performing uniform data exchange. The illustration show example patterns cp that are included in CP^{exch} (e.g. for allgather). The figures representatively illustrate an allgather communication. The rectangles of one line indicate the processes $v \in cp.V^g$ that participate in the communication and each line stands for the status of the communication between two logical steps of the communication pattern. The arrows mark communication edges $e^c \in cp.E^c$. Since in each step all processes concurrently perform one send and one receive, there is one send-receive operation per process and logical step. The numbers assigned to each process in each step show the data and their ordering that the processes hold at each step. The presentation of the illustrations are inspired by [Jin+05].

2.4 Real-Time Analysis

In the field of embedded computing the systems are often required to be real-time systems. The characteristics of such systems are presented in [SR90]. The overall correctness of a real-time system requires not only the functional correctness but it also must fulfil a certain timing behaviour. Typically, this correctness in time is not characterized by the average time for executing software but the time of each execution of the code under consideration matters [SR90]. Subsequently, we refer to the term task for the code that requires real-time capabilities and to the term job for a concrete execution of the code.

In concrete in a real-time system each job of a task must hold a certain deadline. Thereby, a deadline denotes the point in time at which each job of a task must have finished its execution. According to [Kop11, p. 3] there are several categories of real-time, which are called soft real-time, firm real-time and hard real-time. In soft real-time the miss of a deadline can be tolerated to a certain amount. The miss of a deadline only causes that the result is outdated and thus, less valuable for the execution of the overall program. Firm real-time

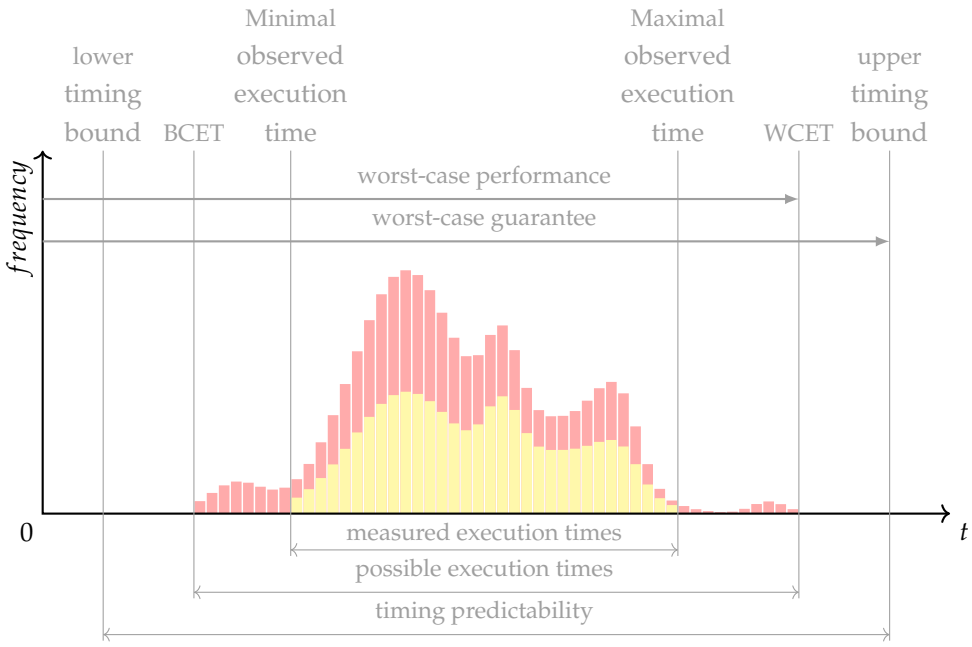


Figure 2.15: Different aspects of the execution times of a task. The execution time of a particular job is indicated on the horizontal axis, while the vertical axis denotes the percentage of the occurrence of the time. The red bars mark the real distribution of execution times of a job and the yellow bars display the distribution of the measured execution times after a significant number of executions. The grey lines and arrows highlight different aspects of the timing. The figure is based on [Wil+08].

means that a result becomes useless as soon as the job misses the deadline. In both cases a certain amount of deadline misses can be tolerated. However, the allowed amount of misses strongly depends on the executed application. On the other hand in hard real-time systems all the deadlines of all jobs must hold. Otherwise situations may occur that cause harm to people or lead to significant damage of the system of its environment. Each task of a real-time system can be classified to one of the presented categories, depending on the arising consequences of a miss of the deadline. Tasks that do not need to fulfil any real-time constraints are called to be non-real-time tasks. In this work we focus on hard real-time systems.

There are different aspects of the timing of the execution of a task. They are presented in [Wil+08] and displayed in Figure 2.15 which show the distribution of the execution time of a task for multiple executions. Thereby, the variations of the execution times are caused by different input data and code coverages for a particular execution. Both displayed distributions (the real and the actual measured execution times) are limited with a maximal and a minimal execution time. For the measured execution times (yellow bars in the Figure) we refer to those limits to the minimal observed execution time and the maximal observed execution time. There are no lower or higher execution times measured for the regarded number of runs of the program. Hence, the range between both limits is denoted as the

interval of measured execution time.

In reality there may be times possible for the execution that are not measured during the runs (cf. red bars in Figure 2.15). Thus, the limit for the minimal possible execution time typically is shorter than the minimal observed execution time. This time is called the *best-case execution time (BCET)*. On the other hand the maximal possible execution time typically exceeds the maximal measured time and is called *worst-case execution time (WCET)*. These limits (BCET and WCET) form the range of the possible execution times. Typically, the determination of the real limits is not possible as the possible variations of input data, input states and uncertainties during execution are to manifold to cover all possibilities. Therefore, approximations and estimations are used to obtain timing bounds that are close to the real limits. These bounds are denoted as save if they are provably lower than the BCET or provably higher than the WCET. Hence, they are called the lower timing bound for the approximation of the BCET and the upper timing bound for the approximation of the WCET.

Typically, in hard real-time systems the WCET and its upper bound are the relevant aspects of the timing behaviour. Hence, there are two terms that refer to the performance of a real-time task. The term worst-case performance denotes the actual performance of a task in the worst case, while a worst-case guarantee states the guaranteed performance of a task according to the obtained upper timing bound.

To provide the WCET guarantees a special analysis must be performed, since measuring the time of an execution always underestimates the actual WCET. There are two types of analyses that found on different principles. These types are the static WCET analysis and the measurement-based WCET analysis. Both techniques are shortly presented, subsequently. If not stated otherwise the content of the following sections is based on Metzlaff [Met13, p. 41ff.] which in turn is mainly based on Wilhelm et al. [Wil+08]. In this thesis we use the static WCET analysis technique to reveal the results for the evaluation sections.

2.4.1 Static Analysis

This analysis statically analyses the code of a task and obtains the upper bound for the WCET by always assuming the worst possible case for an execution based on a model of the underlying hardware and the knowledge of the current state as a result of the previously executed code. According to [Eng+03] the analysis can be divided in three different phases. These are the analysis of the program's control flow, a low-level analysis that considers details of the hardware architecture and the actual calculation of the WCET bound.

In the analysis of the program control flow [The02] the application's control flow graph is constructed in order to determine the possible paths of execution. Based on this graph it is possible to determine the longest execution path in terms of worst-case timing in the WCET calculation phase. Typical results of the control flow analysis include the functions that are called, the number of iterations executed in a loop or the dependencies between conditional execution parts. [Eng+03]

The low-level analysis targets the timing behaviour of the underlying hardware. Thereby, it considers the behaviour of the processor pipeline as well as the utilized memory system [Met13, p.42]. The low-level analysis must regard to the correct type of the pipeline

and to additional possibly speculative components. There is a lot of work done for the analysis of the different types of hardware. Metzlaff [Met13, p.42] provides a short overview of analysis examples for different types of pipelines. The types reach from very simple pipelines (e.g. in [ZBN93; Sun+95]) to superscalar in-order pipelines (e.g. in [JII94; Sun+98]) or out-of-order pipelines (e.g. in [RS09; LRM06]) and speculative components such as a branch predictor (e.g. in [CP00; BR05]). Next to the variations of pipelines there are different memory systems, too. Here a variation can be for example the usage of caches (analysis are e.g. in [FW99; HP11]) or scratchpad memories [Met13].

The responsibility of the WCET calculation phase is the combination of the previously presented analyses to calculate the timing upper bound. According to [Eng+03] there are three different approaches for the calculation. The first one is called path-based calculation (cf. [SEE01]) and examines all possible execution paths and compares their timing bounds to obtain the paths with the maximal execution time. The tree-based approach (cf. [CP01]) considers the tree representation of a program. It follows the tree bottom up to find the maximum execution time. The third approach is the IPET approach [LM95] and uses *integer linear programming (ILP)* for calculating the WCET. For that it represents the results of the previous analysis phases as a set of constraints and objective functions.

There is a large number of tools available that use static analysis. Some examples are the commercial tool aiT from AbsInt [1] and OTAWA [Bal+11]. A more exhaustive survey with detailed descriptions is provided by Wilhelm et al. [Wil+08].

2.4.2 Measurement-based Analysis

On the other hand the measurement based WCET analysis relies on the measurement of execution times of the program or of parts of it. These measurements are either taken from end-to-end of a program or from segments of the program. Thereby, the segments are typically the basic blocks of the code under analysis. Altogether, with this approach the measurements replace the low-level analysis of the static analysis [Wil+08].

With measurement based analysis it must be ensured that the measurements of each measured part include the worst case of the execution to provide a safe upper timing bound. According to Wilhelm et al. [Wil+08] holding this assurance is a problem for measurement based methods. A possible solution to consider the worst-case execution path of the program is to use a control flow analysis. It is applied to find all possible paths which are then combined with the measured times in order to reveal an end-to-end timing bound. This approach leads to a safe upper bound if it is guaranteed that all measures of the basic block are safely upper bounded. However, providing safe upper bounds to basic blocks need to ensure the consideration of the worst-case initial processor state for each measurement. Especially for complex processors this is a hard or even impossible task [Wil+08].

Wilhelm et al. [Wil+08] states that there are multiple ways to perform measurements. The simplest option is to provide an additional instrumentation of the code that collects timestamps or cycle counters. Another method is a mixed HW/SW instrumentation. It requires additional hardware and is able to collect timings with only a lightweight instrumentation.

[1] AbsInt Angewandte Informatik GmbH - `www.absint.com/ait/` [Online, last accessed on 4th November 2019]

There are non-intrusive methods available, too. Typically, these methods rely on techniques like logic analysers. Other options are the usage of VHDL simulators for example.

An example for a tool that utilizes measurement-based analysis is RapiTime from Rapita Systems Ltd. [Rap08]. It relies on the measurement of execution times of basic blocks and their combination according to the structure of the program under analysis. A more detailed description and a survey of other measurement based tools is provided by Wilhelm et al. [Wil+08].

3
Related Work

In this chapter we present previous work that is related to our work. It complements the literature that is already refered to in Chapter 2, where we provide the background knowledge for this work. To the best of our knowledge we provide descriptions to the work that exhibits most similarties to our work and mention the most relevant related literature.

We separate the presented literature in two sections whereby each handles a different topic. The first Section 3.1 provides an overview over *Quality of Service (QoS)* and GS for NoCs, while the second Section 3.2 targets the implementions and evaluation of MPI based communication.

3.1 Guaranteed Service for Network-on-Chips

There are several methods for GS. The most common ones are rate control for routers or virtual circuit switching applying TDM [GH10].

The usage of TDMA for the communication in HRT systems is introduced by Kopetz et al. [KG94; KB03]. While Kopetz targets off-chip networks, the principles of this technique are also suitable for NoCs. There are several approaches to apply TDM on a specific platform. One option is utilizing an application specific schedule, as applied in [GH10; SMG14; Sch07] in different variants. Such schedules are statically defined for the communication of a specific application or set of applications and require detailed knowledge about the internal communication at design time. Sørensen et al. [Sør+14] developed a methaheuristic scheduler for inter-processor communication in multi-processor platforms using TDM. Other approaches for TDM do not target application specific schedules but apply a general purpose static schedule independent of the actual communication. Schoeberl et al. [Sch+12] propose the use of an All-to-All schedule. This schedule enables each node to send one flit to each other node within one period and Brandner and Schoeberl [BS12] present a heuristic to

generate static schedules that provide all-to-all communication. Mische and Ungerer [MU14] provide several alternatives to the all-to-all schedule. These general-purpose schedules of Mische and Ungerer [MU14] and Schoeberl et al. [Sch+12; BS12] provide latency guarantees that are not influenced on the placement of tasks on a processor platform. Details of this work are described in Section 2.2.2. This thesis mainly bases on the previous work of [MU14].

A mathematical framework called Network Calculus is often used for rate control. Its purpose is to reason about dynamically scheduled packet-switched networks. This framework is introduced by Cruz [Cru91a; Cru91b] and later simplified and extended by Le Boudec [Le 96; Le 98]. Network Calculus models the behaviour of each router by reasoning about the incoming and outgoing traffic. Thereby, the data that arrives at or is forwarded by the router is plotted in curves (arrival and departure curve) and according to these curves it is possible to give a statement about the time a packet may be delayed and the backlog that occurs in the router. Originally, Network Calculus targets off-chip networks but there are also applications to on-chip networks [Bak+09; QLD09; Sub+10]. Furthermore, there are various variants for different types of networks. One example is a variant for the Kalray MPPA-256 processor [Din+14; de +14], which is based on [Hui95]. Another variant is suitable for mixed-criticality off-chip networks [Boy+13].

Another approach to reason about latencies in NoCs that are dynamically scheduled is based on the idea of response time analysis [JP86]. Shi and Burns [SB08; SB10] consider the maximal interference of communication flows to reason about the maximal latency of that message. Thereby, the flows are assigned with priorities and the rate for inserting packets of a flow is bounded. There is also an extension that combines response times for tasks with response times for NoC traffic [Ind14]. Altogether, Network Calculus and response time analysis are alternate approaches to TDM for GS in NoCs. Puffitsch et al. [PSS15] shortly summarizes the three approaches and compares Network Calculus and TDM for the usages in NoCs.

Saha et al. [SMT94] provide a non-work-conserving TDM-based multi-rate control mechanism for a network processor. Thereby, the main idea is to grant time slots of a TDM cycle to connections that are not satisfied in previous TDM cycles if a single TDM cycle is not able to fulfil the guaranteed rate of all served connections. Hansson et al. [HCG07] provide channel trees to reduce the latency of data communication when using TDM for QoS in NoCs. These trees are sets of point-to-point channels that share the same source or destination node. Thereby, channels are established end-to-end connections between to nodes of a NoC. For each channel there are time slots allocated to the source node for sending data. With channel trees several channels which share the same source or destination node can be aggregated to use the same time slots for the physical links at the root of the tree. This leads to an over-allocation of the TDM slots and hence, must be handled by an additional scheduling level that multiplexes the channels of a tree to prevent conflicts at data delivery. The channel to use are determined by the application executed on the NoC. The work in [Gan08] is very similar to [HCG07] but does not cover the allocation of time-slots and the evaluation of the approach. Similar to this work Guerin and Orda [GO00] and Stuijk et al. [Stu+06] provide multiplexing of independent channels in a TDM-based NoC, too. Guérin and Orda [GO00] provide initial work on the effects of advance reservation for path selection in a network and analyse the impact on computational complexity. Thereby, for particular service models

it enables the preemption of lower priority connections for granting bandwidth to higher priority connections. The idea in [Stu+06] is to reserve bandwidth guarantees to connections not permanently but only during certain intervals. Thereby, the paper provides several heuristics to find feasible schedules.

As seen in the previous paragraph the idea of multiplexing several communication channels on a single time slot in a TDM-based interconnect is presented in various variants [HCG07; Gan08; SMT94; GO00; Stu+06]. In Chapter 4 the presented TDM schedules apply similar ideas for the composition of the communication paths. However, while the related approaches optimise application specific schedules, our approach targets to form general-purpose schedules with latency guarantees that are independent of task placement. Furthermore, our focus in the evaluation is different. The experimental evaluation of Hansson et al. [HCG07] for example mainly focusses on a fixed number of four channels on a 2×3 mesh NoC and on three embedded case studies of SoCs. The other related work [Gan08; SMT94; GO00; Stu+06] targets only small NoCs as well and focuses on computational complexity or on networks rather than on NoCs. In contrast, we target the systematic evaluation of group communication in NoCs and vary the number of participating nodes, the size of the NoC and the size of the messages sent.

There is a vast number of NoCs that use TDMA to provide QoS. Most of them use slightly different variants of TDM. Since our approach is not exclusively restricted to one specific NoC[1], we give a non-exhaustive survey of such NoCs and exemplarily provide a short description of the functionality of some presented NoCs. The Intel Labs presented the Intel *Single-chip Cloud Computer (SCC)* [Int10] and Scheller [Sch12] investigated real-time programming for this chip. However, the SCC cannot guarantee bounded timing behaviour for each case, as [PNP13] reveals. CompSOC [GAC+13] is another platform which focuses on timing-predictability. Other often referred TDM NoCs are the Nostrum NoC [Mil+04], Aethereal [GDR05], Aelite [HSG09] and the Argo NoC [Kas+16; Kas15]. Nostrum [Mil+04] implements the concept of so called looped containers. These containers are constantly routed through the NoC. For sending a message the source node waits for an appropriate container. At its arrival the message is put in that container which then transports the message to the target node. The Aethereal network [GDR05] uses switching tables in each router that determine the current connection of an input port to an output port. These connections change for each cycle within a fixed period. After injecting a message to the NoC it is forwarded without buffering until is reaches the targeted node. The DSPIN micronetwork [PGS06] combines rate control and TDMA. It provides one virtual channel in each router for serving QoS needs. Within this channel TDM is used to aviod conflicts between message streams. The Kilo-NoC [Gro+11] targets to reduce overhead for providing QoS. Therefore, it only provides GS to parts of the NoC and uses simple routers elsewhere.

Another possibility for providing QoS is to apply TDM on virtual channel level rather than multiplexing time slots. This technique is implemented in PhaseNoC [Psa+15]. The pipeline of the router in this NoC serves different virtual channels in each pipeline stage. To avoid waiting times the handling of the channels is arranged in a way that the first stage of a router handles a channel right after the finishing of the last pipeline stage for the channel

[1] Though we partly describe our work at the example of a particular platform (RCMC) [Mis+17]

in the previous router. This leads to a wave-like activation of the virtual channels through the NoC.

A lot of research has been carried out on analysing real-time performance of off-chip and on-chip networks. Most of the work targets the analysis of best-effort networks. One example is the work of Diemer et al. [Die+11] that provides a formal real-time communication analysis for latency and throughput. Thereby, they focus on wormhole-switched networks with a multi-stage arbitration of the shared resources in a router to incoming data streams. Diemer et al. focus on router architectures without special support for real-time traffic. They base their analysis on the scheduling algorithm iSLIP [McK99] which is also investigated by Gopalakrishnan et al. [GCL06]. It arbitrates the real-time streams in multiple steps to the components of a router. The approach considers individual traffic injection patterns for the analysis. They are characterized by arbitrary worst-case arrival functions. These functions describe the maximal number of flits that arrive during a given period of time. Other real-time analyses of best-effort NoCs are provided for example in [QLD09; QLD10; GCL06]. Additionally, Rambo et al. [RE15] and Tobuschat et al. [TE17b] target a worst-case communication time analysis of wormhole-switched best-effort NoCs, too. Furthermore, [TE17a] focuses on latency guarantees in mixed-criticality NoCs. There has been also done some work that supports backpressure in its analyses [KP16; LT01; TE17b]. Other work especially focuses on traditional off-chip networks [HKR14; Lee03; Byu+98]. However, according to [BD01] these technique cannot be directly applied to NoCs. Other approaches use priorities to provide QoS in their NoCs. Techniques for the analysis of priority-based wormhole switching NoCs were investigated in [LJS05; SB08; KGP15; SB09].

We presented a non exhaustive set of investigations about real-time in NoCs that are using best-effort NoCs. In contrast to these papers we apply TDM to prevent the occurrence of contention instead of modelling it in our analysis. Furthermore, the presented work does not target the analysis of specific standard communication patterns. Instead, it regards to the complete system including the consideration of all applications running on the chip at the same time. This causes a high analysis effort if the applied set of applications changes. In contrast, we target an analysis that causes only small efforts for recalculating the worst-case timing behaviour of a changed set of applications.

3.2 Message Passing Interface

MPI is a widespread standard for developing parallel programs on distributed and shared memory architectures. Therefore, a lot of work has been done that targets the efficient and performant implementation of the communication operations in MPI. In [BPG91; JC97] several tree-based broadcast and multicast algorithms are presented. Also, McKinley et al. [MTR94] survey several algorithms. Sack and Gropp [SG12] consider the hardware topology to implement performant allgather and reduce-scatter algorithms. Thereby, they target supercomputers like the BlueGene/P with 32-k nodes. Algorithms for barriers have been investigated exhaustively, too. Examples for the work on this topic are [LMN94; Pan95; JC98; San+01]. Research was also carried out for other collective communication operations like all-to-all communication [TC94; Alm+05]. While some of the work especially concentrates on a specific collective like the allgather [Ben+03] or target specific use cases

like the delivery of short messages [Bru+97], other work focuses on the improvement of the performance of the complete group of collective communication [TRG05; TG03; Cha+04]. The list of work about developing implementation variants of different communication operations can be continued. Altogether, this literature forms the basis for the different implementations of the MPI standard [MPI15] like for example OpenMPI [Tea16], which we used for porting the communication operations to our targeted platform. Hence, though this topic in literature mainly targets HPC it also forms the basis for our work as it provides performant state-of-the-art implementations of the communication investigated in this thesis.

In addition to the development of different implementations a lot of work deals with performance evaluation of the standard's communication primitives. However, most effort targets the area of HPC and often considers the LogP model [Cul+93] to assume a specific kind of network [Pje+07; OG04; Hoe+05]. Besides LogP other computation models can be used to analyse collective communication in HPC, too [Tu+12]. There is a comparison of the usage of different models for analysing the performance of collective communication provided in [Pje+07]. It compares the Hockney model [Hoc94], LogP [Cul+93], LogGP [Ale+97] and PLogP [KBV00]. Besides the usage of computation models another method for analysing the performance of collective communication is based on measurements and among others can be found in [HLR07; Liu+03; Ben+03]. The complexity of collective communication on NoCs is investigated by Jaros et al. [JOD06]. Chen et al. [Jin+05] evaluates allgather algorithms on a terascale linux cluster with fast ethernet. However, all this work does not target real-time behaviour and often focuses on the differences in applying various networks.

Other work that focuses on the performance of communication operations in NoCs addresses real-time behaviour as well. Sørensen et al. [Sør+15] examine a barrier and a broadcast implementation with respect to their WCET bounds. Thereby, they use a slightly different technique to ensure upper bounds for traversal times in the NoC than this thesis and focus only on relatively small numbers of cores. In contrast, Stegmeier et al. [Ste+16] investigates NoC sizes up to 64 cores, but it only considers WCTTs. Frieb et al. [Fri+16b] is based on [Ste+16] and complements the missing end-to-end WCETs. However, instead of systematically providing timing bounds for different collectives it mainly focuses on the issue of analysing a complete MPI program. Additionally, all those papers only consider one communication pattern (basic linear, see Section 2.3) for the collectives. Frieb [Fri19] investigates hardware extensions for a timing-predictable many-core processor. These extensions are mainly applied to the NI and utilize the characteristics of one of the general purpose schedules (One-to-All, see Section 2.2.2). Though Frieb also provides case studies that perform a WCET analysis of MPI programs, his work mainly focuses on the hardware extensions.

Yagna et al. [YPM16] focus on efficient and predictable implementation of group communication. They provide communication patterns for the most common collective communication operations. Thereby, the patterns are arranged with the target in mind to reduce possible contention between the single point-to-point communications of a pattern. They evaluated the average execution times and their variance on a 64-core Tilera TilePro processor and the Intel SCC. The experimental results show an improved performance and a reduction of the

variance of the execution times. In contrast to our approach the applied communication patterns depend on the concrete mapping of the processes to hardware nodes. A further difference is that they applied a best-effort NoC for their evaluation and only target the reduction of the variance of the average execution time but do not apply GS for the NoC or provide a safe upper bound for the WCET.

Kanevsky et al. [KSR98] and Skjellum et al. [Skj+04] present a standard for MPI which is capable for real-time systems. The standard provides an object-oriented view on the communication and suggests concepts for constructing a complete middleware. However, as it generally focuses on processor networks, it is not appropriate for on chip communication.

4

General-Purpose TDM Schedules for Message-Passing

Often an application specific TDM schedule is used to provide an upper bound for the communication time of an application [GH10; SMG14; Sch07]. However, those schedules are difficult to calculate and depend on the application and the mapping of tasks to cores. Thus, when an application specific TDM schedule is used, either the analysis of the communications must be performed from scratch for each application or the worst case in terms of placement must be assumed. Both options suffer in practibility, as the effort of multiple analyses is very high and the provable identification of the worst mapping for a specific communication pattern is complicated. Moreover, implementations of the MPI standard provide library functions that implement the communication defined in the standard. Typically, these functions are independed from the placement of the tasks to specific processing elements. As MPI provides standardized communication patterns independent of a specific application, this independence should hold true for the timing bounds as well. The general-purpose TDM schedules address these issues in an appropriate way as they allow each core to send and receive the same amount of data within one period of time. As Mische and Ungerer [MU14] show, their timing behaviour is independed from placement of tasks to nodes.

The downside of such general-purpose schedules is that they cannot be optimized for a specific application and therefore, may lead to higher latencies and lower bandwidths than application specific schedules. But on the other hand the worst-case timing of a MPI collective is not influenced by any other communication which is executed additionally to the regarded collective. Furthermore, when assuming the transmission time of the longest possible path through the NoC for all communication between any cores, the timing becomes independent of the task placement, too. Thus, there is no need to concern about

the actual path distance between communicating cores for timing analysis. Consequently, the calculated timing of a collective is generally valid for a given hardware configuration. Hence, an analysis performed based on these assumptions results in a timing behaviour that holds true for all possible surrounding software influences and task placements on chip.

This chapter extends the largely informal descriptions of state-of-the-art general-purpose TDM schedules of [MU14] to formal definitions and shows the construction for all TDM schedules. Thereby, the expressions follow the system model described in Chapter 2. Furthermore, it presents two new general-purpose schedules. The new schedules are compositions of the previously presented state-of-the-art schedules and target the combination of the positive aspects of their basic schedules. Afterwards, we present a formal analysis of the schdedules' timing behaviour. After describing the procedure of this analysis it is applied to the schedules to obtain their influence of TDM schedules on the timing for message passing and a discussion provides insights to the advantages and disadvantages of the investigated schedules in terms of timing. Besides evaluating the different variants of general-purpose schedules, this chapter also discusses a comparison to application specific schedules to ensure an encompassing consideration.

4.1 Formalization and Construction of TDM Schedules

In order to investigate the influence on timing of the TDM schedules, we subsequently provide a formal definition of the schedules provided by Mische and Ungerer [MU14] (see also Section 2.2.2). As concrete implementations of schedules are dependent on the applied hardware, a concrete platform is needed as basis for our schedule definitions. We assume the same NoC topology and fundamental routing policy as in [MU14]. Thus, the subsequent schedule definitions are designed for a quadratic two-dimensional torus with $n \times n$ nodes with the following set of nodes.

$$N = \{n_i \mid i \in [0, n^2 - 1]\} \tag{4.1}$$

As we consider a uni-directional torus, the position of the nodes follow the example of Figure 2.5a which is formally stated as follows.

$$POS(n_i) = (x, y) = \left(i \bmod n, \left\lfloor \frac{i}{n} \right\rfloor \right) \tag{4.2}$$

The nodes are connected via directed physical links $ph \in Ph$ consisting of one hardware slot $s \in S$ in a way that all horizontal rings and all vertical rings route flits in the same direction. Hence, each node can only send flits to the east and north neighbours and receive flits from the south and west neighbours. Please note that with this kind of connection each horizontal row and vertical column of a node can also be seen as a uni-directional ring (cf. Figure 4.1). Equation 4.3 formally defines the described characteristics of the applied physical links.

$$\begin{aligned} Ph = \{ph \mid\ & |ph.S| = 1 \\ & \wedge ((POS(ph^{dst}).x = POS(ph^{src}).x + 1 \bmod n \\ & \quad \wedge POS(ph^{dst}).y = POS(ph^{src}).y) \\ & \quad \vee (POS(ph^{dst}).x = POS(ph^{src}).x \\ & \quad \wedge POS(ph^{dst}).y = POS(ph^{src}).y + 1 \bmod n))\} \end{aligned} \tag{4.3}$$

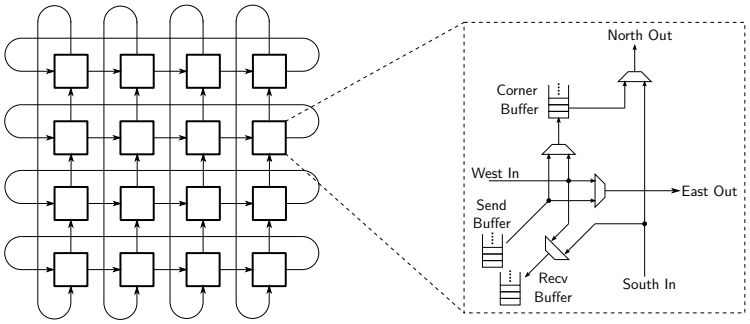


Figure 4.1: Logical network structure of the assumed topology. The left part shows the connection of physical links and nodes including the direction of the links. The right part indicates the minimal assumed components of a router in a node. [Ste+16]

Furthermore, the forwarding of flits along the horizontal and vertical ring can be handled concurrently by a router and therefore, cannot conflict each other.

Corollary 4.1. *It follows from the description of the assumed hardware that no pure horizontal delivery of multiple flits and paths can conflict with any other pure vertical delivery of multiple flits and paths even if they are executed concurrently.*

A strict dimension-ordered routing is applied which starts in horizontal direction. After inserting a flit in the horizontal ring it is forwarded bufferless to its target column and stored in the corresponding corner router's buffer. Here the flits wait until they are inserted in the vertical ring which applies bufferless forwarding as well. Hence, the minimal set of rules for the routing policies R_i^P is given by the following function.

$$R^P_{PE(x,y)}(n^{src}, n^{dst}) = \begin{cases} (0, PE(x+1 \bmod n, y)) & \text{if } x \neq n^{dst}.x \wedge y = n^{src}.y \\ (k, PE(x, y+1 \bmod n)) & \text{if } x = n^{dst}.x \wedge y = n^{src}.y \\ (0, PE(x, y+1 \bmod n)) & \text{if } x = n^{dst}.x \wedge y \neq n^{src}.y \end{cases} \tag{4.4}$$

Equation 4.4 shows the minimal needed the routing policies for a router at a position (x, y) of the NoC. The first line describes the policy for horizontal delivery, the second line is responsible for the case when a router acts as corner router and the last line defines the forwarding in vertical direction. It follows from the equation that the transported data is stored in the corner router for a fixed period of time $\Delta t = k$ which is constant for the given node $PE(x, y)$, source node n^{src} and target node n^{dst}. Thus, the set of all routing policies R_i^P is given as follows.

$$R^P = \bigcup_{x,y \in [0,n-1]} R^P_{PE(x,y)} \tag{4.5}$$

Please note that a larger set of routing policies would be possible, but the presented schedules require at minimum those R^P as all paths of the schedules follow these policies with various k depending on the current path and schedule. Figure 4.1 illustrates the assumed topology and a minimal schematic structure for a router of the nodes.

According to the timing of the routing the latencies for a hop and slot are assumed to be equal with

$$L^{hop} = L^{slot} = 1 \; clock \; cycle. \tag{4.6}$$

Furthermore, as we assume buffering in the corner router, it takes at minimum 1 clock cycle to change from horizontal to vertical delivery (cf. Equation 4.4: $k \geq 1$). Please note that this also holds for pure vertical delivery when $n^{src}.x = n^{dst}.x$. Finally, there is one clock cycle needed to forward a flit from the destination router to the local NI but the latency for forwarding in the contrary direction from NI to router at a sender node can be omitted due to the routers' horizontal bufferless forwarding techniques. Mische et al. [Mis+17] present a hardware prototype that follows the definitions presented above.

Please note that schedules for uni-directional tori can be used for bi-directional tori without adaption, because a uni-directional torus is a sub-topology of a bi-directional one (see Section 2.1). Nevertheless, for bi-directional tori there may be schedule implementations available with higher bandwidths and lower latencies, as the additional available physical links lead to a higher link capacity bound B^{Cap} and may provide an extended choice of routing possibilities without conflicts. Hence, an adaption of the schedules is an adequate way for optimization purposes. Additionally, there are only small adaptions needed to apply the schedules presented here to bi-directional mesh topologies. Nevertheless, we leave construction for meshes and trees to future work.

4.1.1 State-of-the-Art General-Purpose TDM Schedules

To the best of our knowledge until now implementations of general-purpose schedules are only presented by Brandner et al. [BS12] and Mische and Ungerer [MU14]. Hence, we see the general-purpose TDM schedules presented in this literature as the state-of-the-art for general-purpose schedules. While Brandner et al. introduce the All-to-All schedule by using construction techniques for application specific schedules, Mische and Ungerer propose the One-to-All, All-to-One, One-to-One and All-to-All schedules that exhibit a manually arrangement of the paths. Thereby, they describe all schedules but the All-to-One in detail and only mention the All-to-One as a further possibility. Section 2.2.2 reviews the descriptions of [MU14] in order to build the basis for this section.

Subsequently, we describe all four schedules in terms of our system model presented in Chapter 2 and extend the original literature with formal definitions. Furthermore, an example for a 3×3 NoC is provided to illustrate the application of the schedules. Finally, we review the proofs and informal explanations about conflict-freedom from the original literature to provide proofs that fit to the system model as well. As there are some characteristics that share all described state-of-the-art schedules, we firstly present the definitions that the schedules have in common before presenting the specifics of each schedule by its own.

Common Characteristics of State-of-the-Art Schedules

All schedules (One-to-All, All-to-One, One-to-One and All-to-All) share the target to provide an equal bandwidth for all connections from each node to each other node. Hence, the number of routes and paths per TDM cycle tdm^C is equal for each pair of nodes (n^{src}, n^{dst}). In the schedules presented below in each TDM cycle there is exactly one route and path from each node to each other node of the NoC. Thus, we can define the set of routes Rt^{tdm}

for the schedules as follows.

$$
\begin{aligned}
Rt^{tdm}_{common} = \{\ \vec{r} \mid \vec{r} \in Rt,\ & \\
& (\forall x, y, u, w \in [0, n-1], x \neq u \vee y \neq w : \exists \vec{r} : \\
& POS(\vec{r}.ph^{src}_{first}) = (x, y) \wedge POS(\vec{r}.ph^{dst}_{last}) = (u, w)) \\
\wedge\ & (\forall \vec{r}_i \in Rt^{tdm}_{common} : \nexists \vec{r}_j \in Rt^{tdm}_{common}, i \neq j : \\
& \vec{r}_i.ph^{src}_{first} = \vec{r}_j.ph^{src}_{first} \wedge \vec{r}_i.ph^{dst}_{last} = \vec{r}_j.ph^{dst}_{last})\}
\end{aligned}
\tag{4.7}
$$

In Equation 4.7 the schedules' set of routes is constructed by two conditions. The first condition, which is located in the second and third line, states that there must be a route between each source node with coordinate (x, y) and each destination node with the coordinate (u, w) except when both coordinates are equal. As all components of the coordinates (x, y, u, w) go from 0 to $n - 1$, the source as well as the target nodes cover all nodes of the NoC. The second condition in lines four and five ensures that there is only one route that exhibits the same source and target node. Please note that this condition implicitly requires that the set $\vec{r}_{i,j}$ of a source node n_i and a destination node n_j has a size of $|\vec{r}_{i,j}| = 1$. Thus, we use the symbol $\vec{r}_{i,j}$ as synonym for this only route contained in the according set $\vec{r}_{i,j}$.

Additionally, as each node is connected to each other node, Equation 4.7 implies that there are $n^2 \cdot (n^2 - 1) = n^4 - n^2$ routes $\vec{r}$ per TDM cycle tdm^C that can be used by the nodes and we already mentioned that one route shall be represented by exactly one path.

$$
|Rt^{tdm}_{common}| = |P^{tdm}_{common}| = n^4 - n^2 \tag{4.8}
$$

The beginning of Section 4.1 states the application of strict dimension-ordered routing and the potential buffering of flits in the routers when they change from horizontal to vertical direction (see Equation 4.4). Therefore, each path exhibits a similar route through the NoC. According to the given routing policy each path goes from its source node at a dedicated point in time in horizontal direction without buffering until it reaches its target column. At this time the path meets its corner router where it obtains a specific storage time Δt. Afterwards, each path follows the nodes and physical links in vertical direction until it reaches the destination node. Equally to the horizontal case, the forwarding in vertical direction is performed without any additional buffering in the routers. Following this description each path $\vec{p}^{tdm}_{i,j}$ from any node n_i with $POS(n_i) = (x, y)$ to any other node n_j with $POS(n_j) = (u, w)$ is defined in Equation 4.9.

$$
\begin{aligned}
\vec{p}^{tdm}_{i,j} = \langle h_{first}, ..., h_{cin}, h_{cout}, ..., h_{last} \rangle & : h_{first}.ph^{src} = n_i \wedge h_{last}.ph^{dst} = n_j \\
\wedge\ & (\exists n_k : n_k = h_{cin}.ph^{dst} = h_{cout}.ph^{src} \\
& \wedge POS(n_k).x = POS(n_i).x \wedge POS(n_k).y = POS(n_j).y) \\
\wedge\ & (\forall l \in [first, last - 1] : h_l.ph^{dst} = h_{l+1}.ph^{src}) \\
\wedge\ & (\forall m \in [first, cin - 1] \cup [cout, last - 1] : h_m.t + L^{hop}_{h_m.ph} = h_{m+1}.t) \\
\wedge\ & (h_{cin}.t + L^{hop}_{h_{cin}.ph} + t_{buf} = h_{cout}.t)
\end{aligned}
\tag{4.9}
$$

In Equation 4.9 the first line shows that the paths are composed of several scheduled hops h and states the source and destination node. The second and third lines identify the corner

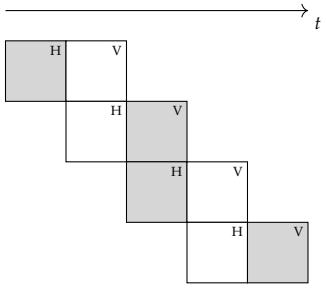


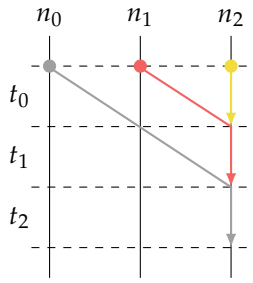
Figure 4.2: Overlapped execution of a schedule's rounds. Each white or gray column marks one round, while each row indicates a set of paths that contain the paths starting in the same round. Typically, the paths start in one round with horizontal delivery and perform their vertical delivery in the subsequent round. The phases of horizontal data transfer and vertical data transfer are marked with the symbols H and V.

router of the path and its connected physical links. The fourth line restricts the scheduled hops to a successive order following connected physical links and nodes as it is also stated in Condition 2.6 for all possible valid paths of a NoC. The bufferless routing within a horizontal or vertical ring is forced in line five and the last line allows the corner router to buffer flits on that path for a given time t_{buf}. Altogether Equation 4.9 forces a fix and similar behaviour to the forwarding of the flits for all paths of a schedule. Besides the source and target nodes there remain only two degrees of freedom for influencing a path's behaviour. These degrees are the point in time when a path begins at its source node $\vec{p}.h_{first}.t$ and the time a flit is stored in the corner router t_{buf}. Thus, the paths of the schedules mainly differ in those two points and in constraints about which paths are allowed to be applied concurrently. The set of all paths possibly included in a schedule's set of paths P^{tdm} is given as follows.

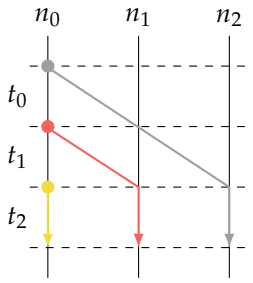
$$P^{tdm}_{total} = \bigcup_{\substack{i,j \in [0,n-1] \\ i \neq j}} \vec{p}^{\,tdm}_{i,j} \tag{4.10}$$

All state-of-the-art schedules share a similar structure of a TDM cycle. They are composed of one or multiple rounds which are further subdivided in an equal number of time slots for each round. Thereby, the presented schedules assume that the slot latency L^{slot} is equal to the hop latency L^{hop}. Typically, paths that begin in one round reach their destination node in the next successive round. Furthermore, in most described schedules (except All-to-All) after each arrival of a flit in its corner router it remains there until the current round finishes. In contrast, the All-to-All schedule starts with the vertical delivery nearly as soon as there are flits available in the corner router. According to the distinguishing of the horizontal and vertical transport, the delivery of the set of paths that begin in the same round can be subdivided in two phases. The first phase is responsible for the horizontal delivery, while the second phase is characterized by vertical data transfers. Therefore, following Corollary 4.1 it is possible to overlap the vertical phase of one round with the horizontal phase of the next round to reduce the schedule's period T. Figure 4.2 indicates this possibility.

There are two different procedures that are mainly used for delivering flits along one direction in a horizontal or vertical ring. Typically, the first procedure is applied when each



(a) The procedure used when there is a restriction for the sender node to send at most one flit per TDM cycle. The main characteristic of this procedure is to send all flits directly at the beginning of a round.



(b) The procedure used when there is a restriction for the receiver node to receive at most one flit per TDM cycle. The main characteristic of this procedure is to send the flits at a time that all flits reach their target node concurrently at the end of a round.

Figure 4.3: Procedures for delivering flits along one direction in a horizontal or vertical ring of a NoC. Each vertical line represents a node n_i and the time goes from top to bottom in terms of time slots (separated by dashed lines). Arrows mark the paths of the schedule. Thereby, a bold dot on a node's line illustrates the time when a flit can be inserted to the NoC along that path. The arrow tips indicate at which time a flit arrives at its target node. The paths are illustrated in different colours to support readability.

node is allowed to send at most one flit per TDM cycle. With that restriction it is possible to release the flits of each node to the NoC at the same time. Thus, they start at the beginning of a round and reach their targeted node after travelling all hops from the source to the destination node plus one hop for inserting the flit to a buffer (at the corner router or the NI). In the second procedure there is no restriction about sending but each receiving node is restricted to receive at most one flit per TDM cycle. In those cases it must be possible for each node to send as many flits as there are nodes reachable with the delivery in the regarded direction and round. On the other hand all flits can arrive at their target node at the same time as each node is allowed to only receive at most one flit. Thus, the sender node releases the flits as many hops as they need to travel to the target node plus one hop before the end of the current round. With this release time all flits reach their target nodes concurrently at the end of the round. Figure 4.3 depicts both options for delivery at an example with 3 nodes.

Subsequently, we provide a lemma that is applied in the remainder of this chapter to proof relaxed conflict-freedom for various schedules.

Lemma 4.1. *We assume that all flits of a NoC follow a path which satisfies Equation 4.9 and regard to either pure horizontal or pure vertical delivery. Then, if each path starts its horizontal or vertical delivery on a different node at the same time slot, all flits can be transported without conflicts along that direction as long as they do not change their direction of transportation. Thus, the following equation holds for a considered set of paths P.*

$$\forall l \in \mathbb{N} : \forall \vec{p}_i, \vec{p}_j \in P, i \neq j : POS(fl_l^i) \neq POS(fl_l^j)$$

Proof. Base case: $\forall l \in \mathbb{N} : \forall \vec{p}_i, \vec{p}_j \in P, i \neq j : POS(fl_1^i) \overset{!}{\neq} POS(fl_1^j)$
The delivery of a flit along a path $\vec{p}$ either in horizontal or in vertical direction starts at a position $POS(f_1)$ which is at node $\vec{p}.h_1.ph^{src}$ (see Equation 2.7). As we assume that all regarded paths start from different nodes, we see

$$\forall \vec{p}_i, \vec{p}_j \in P, i \neq j : POS(fl_1^i) = POS(\vec{p}_i.h_1.ph^{src}) \neq POS(\vec{p}_j.h_1.ph^{src}) = POS(fl_1^j)$$

and the lemma hold for the base case.

Inductive hypothesis: Suppose the lemma holds for all scheduled hops $\vec{p}_i.h_l$ and $\vec{p}_j.h_l$ up to scheduled hops with index k ($\forall l \in [1,k] : \forall \vec{p}_i, \vec{p}_j \in P, i \neq j : POS(fl_l^i) \neq POS(fl_l^j)$).

Inductive step: Let $l = k + 1$. Then from Equation 4.9 it holds

$$\forall \vec{p} \in P : \vec{p}.h_l.t = \vec{p}.h_{k+1}.t = \vec{p}.h_k.t + L^{hop}$$

Subsequently, we assume the notation $POS(fl_i^a) = (x_i^a, y_i^a)$. The two possible cases of pure horizontal or pure vertical delivery are considered separately. For both cases Equation 4.9 ensures that the duration of transporting a flit along a physical link is equal for all links in the NoC and no buffering is performed in the routers.

Case 1: For the horizontal delivery it holds

$$\begin{aligned} &\forall \vec{p}_a \in P : \forall m \in [0, n-1] : POS(fl_{m+1}^a) = (x_{m+1}, y_{m+1}) = ((x_m + 1) \bmod n, y_m) \\ \Rightarrow\ &\forall m \in \mathbb{N} : \Delta x_{m \to m+1} = 1 = const. \wedge \Delta y_{m \to m+1} = 0 = const. \end{aligned}$$

Case 2: For the vertical delivery it holds

$$\begin{aligned} &\forall \vec{p}_a : \forall m \in [0, n-1] : POS(fl_{m+1}^a) = (x_{m+1}, y_{m+1}) = (x_m, (y_m + 1) \bmod n) \\ \Rightarrow\ &\forall m \in \mathbb{N} : \Delta x_{m \to m+1} = 0 = const. \wedge \Delta y_{m \to m+1} = 1 = const. \end{aligned}$$

When combining the statements about both cases it holds

$$\begin{aligned} &\forall \vec{p} \in P : \forall h_i, h_{i+1}, h_j, h_{j+1} \in \vec{p} : \\ &\qquad \Delta x_{i \to i+1} = \Delta x_{j \to j+1} = const. \wedge \Delta y_{i \to i+1} = \Delta y_{j \to j+1} = const. \end{aligned}$$

Hence, we see that

$$\begin{aligned} &\forall \vec{p}_i, \vec{p}_j \in P, i \neq j : POS(fl_k^i) \neq POS(fl_k^j) \\ \Leftrightarrow\ &\forall \vec{p}_i, \vec{p}_j \in P, i \neq j : (x_k^i, y_k^i) \neq (x_k^j, y_k^j) \\ \Leftrightarrow\ &\forall \vec{p}_i, \vec{p}_j \in P, i \neq j : (x_k^i + \Delta x, y_k^i + \Delta y) \neq (x_k^j + \Delta x, y_k^j + \Delta y) \\ &\quad \text{with } x_{k+1} = x_k + \Delta x \wedge y_{k+1} = y_k + \Delta y \\ \Leftrightarrow\ &\forall \vec{p}_i, \vec{p}_j \in P, i \neq j : (x_{k+1}^i, y_{k+1}^i) \neq (x_{k+1}^j, y_{k+1}^j) \\ \Leftrightarrow\ &\forall \vec{p}_i, \vec{p}_j \in P, i \neq j : POS(fl_{k+1}^i) \neq POS(fl_{k+1}^j) \end{aligned}$$

which is our statement for $l = k + 1$. So, the lemma holds by principle of induction for $l \in \mathbb{N}$. □

One-To-All TDM Schedule

The One-to-All schedule allows each node to send at most one flit per TDM cycle but each node may receive one flit from each other node. In this section we extend the description of the One-to-All schedule from Section 2.2.2 for a quadratic uni-directional torus with $n \times n$ nodes with formal definitions, give an example for the assumed NoC and proof its freedom from conflicts. One TDM cycle consists of as many rounds as there are nodes in one dimension, which are n rounds for the considered hardware. Additionally, each round consists of n time slots. Section 2.2 states that the length of each time slot is determined by the slot latency L^{slot} and the assumed hardware says that it is equal to the hop latency L^{hop}. Thus, we can deduce the period T_{1a} of this TDM schedule tdm^S_{1a} as follows.

$$T_{1a} = n \cdot n \cdot L^{slot} = n^2 \cdot L^{hop} \tag{4.11}$$

Furthermore, this schedule follows the previously described statements about common characteristics of the schedules. Hence, it shows an equal set of routes as the common one and all paths follow $\vec{p}_{i,j}$ of Equation 4.9 and the corresponding set P^{tdm}_{total} of Equation 4.10.

$$Rt^{tdm}_{1a} = Rt^{tdm}_{common} \tag{4.12}$$

$$|Rt^{tdm}_{1a}| = |P^{tdm}_{1a}| = |Rt^{tdm}_{common}| = n^4 - n^2 \tag{4.13}$$

$$P^{tdm}_{1a} \subset P^{tdm}_{total} \tag{4.14}$$

All paths of the schedules start at the beginning of one of the rounds and are equally and deterministically distributed among them. Since there are n rounds available and there are no paths from a node to itself, each round contains in total the beginning of $n^3 - n$ paths (cf. Equation 4.13). As already mentioned in Section 2.2.2, the r-th round is responsible for initiating the delivery of flits that exhibit a horizontal distance to the destination node of $\Delta x = r$ hops ($r \in [0, n-1]$). Thus, the flit with a receiver in the same vertical ring of the NoC as the sender is started in the first round ($r = 0$), the second round is used for destinations in columns right next to the sender and so on. Following this distribution in each round there starts one path from each node to all other nodes of the regarded target column, which are in total n paths per round and sender node or $n-1$ paths if the sender is located in the same column as the current destination node. Furthermore, all paths of a round begin at the first time slot s^t of that round (similar to the procedure depicted in Figure 4.3a) and after the horizontal transport wait in the corner router for the vertical delivery until the begin of the subsequent round (also similar to Figure 4.3a). Thus, the set of paths of all nodes for first time slot $s^t_{r \cdot n}$ of the TDM cycle's r-th round is given in Equation 4.15.

$$\begin{aligned} P^{s,1a}_{r \cdot n} = \{\vec{p} \mid \vec{p} \in P^{tdm}_{1a}, \exists x, y, w \in [0, n-1] : \\ (\vec{r}_{\vec{p}}.n^{src} = PE(x,y) \wedge \vec{r}_{\vec{p}}.n^{dst} = PE(x + r \bmod n, w)) \\ \wedge (\exists h_{cin}, h_{cout} \in \vec{p} : h_{cin}.ph^{dst} = h_{cout}.ph^{src} = PE(x,w) \\ \wedge \vec{p}.h_{cin}.t + L^{hop}_{h_{cin}.ph} + (n-r) \cdot L^{slot} = \vec{p}.h_{cout}.t)\} \end{aligned} \tag{4.15}$$

The second line of Equation 4.15 determines the source and destination node for a path starting in time slot $s^t_{r \cdot n}$. The third line states the node used as corner router for this path

and the in fourth line the term $(n-r)L^{slot}$ gives the time a flit remains in the corner router when it is transported along the given path. Please note that there are no other paths in the schedule than the stated ones.

$$P_{1a}^{tdm} = \bigcup_{r \in [0,n-1]} P_{r \cdot n}^{s,1a} \tag{4.16}$$

The One-to-All schedule allows each node to send at most one flit per TDM cycle. This informal restriction must be ensured by defining a formal constraint for the schedule that can be added to the schedule's set of constraints $tdm^S.Co$.

$$\begin{aligned} &\forall i \in \mathbb{N} : \forall \vec{p}_l \in tdm_i^C.P^{tdm} : \nexists \vec{p}_m \in tdm_i^C.P^{tdm} : \\ &\quad \vec{p}_l.h_{first}.f \neq \emptyset \wedge \vec{p}_m.h_{first}.f \neq \emptyset \wedge \vec{r}_{\vec{p}_l}.ph_{first}^{src} = \vec{r}_{\vec{p}_m}.ph_{first}^{src} \end{aligned} \tag{4.17}$$

Condition 4.17 considers a specific TDM cycle tdm_i^C of the regarded schedule. In this cycle it prohibits that both of the two paths with the same source node deliver a flit in this cycle tdm_i^C.

In the remainder of this work we call a path that is actually used to delivery data at a regarded period of time an active path. More detailed we also refer to the currently horizontal and vertical delivery of a path. Hence, for example a path is called currently active in its horizontal phase if the path is active and the transported flit is currently forwarded in horizontal direction.

Corollary 4.2. *From Equation 4.17 it follows that all paths of tdm_{1a}^S that are concurrently active and start at the same time slot exhibit disjunct source nodes.*

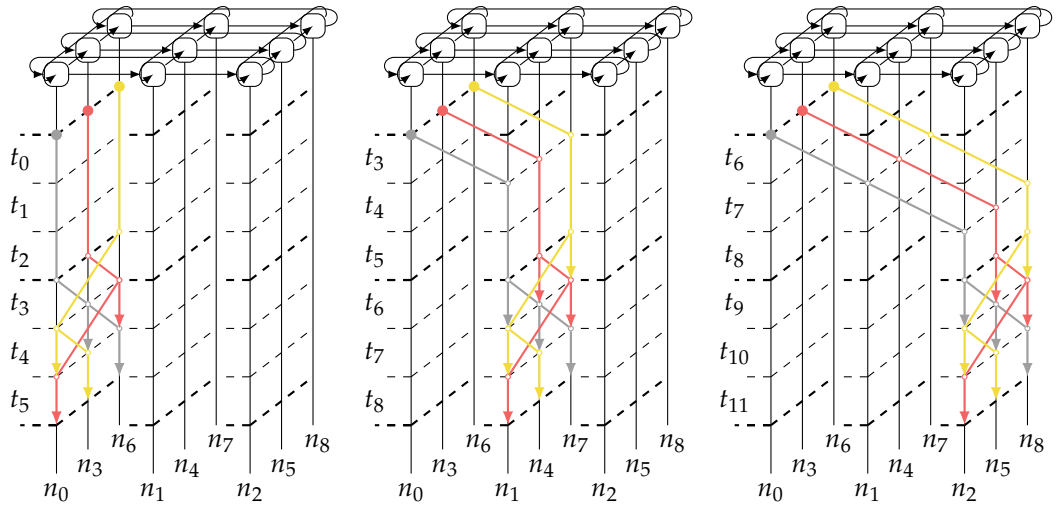
Condition 4.17 is the only element of the One-to-All schedule's set of constraints $tdm_{1a}^S.Co$ and forces each source node to only use one of its available paths per TDM cycle. Hence, in total at most n^2 paths of the total amount of $n^4 - n^2$ paths are actually used in a given TDM cycle.

Figure 4.4 illustrates the defined paths of the One-to-All schedule at an example using a 3×3 uni-directional torus NoC. In each round there are n or $n-1$ paths available for each node, which means that there are 3 or 2 paths per node and round in our example. In the first round flits can be delivered to nodes that are located in the same column as long as the destination node is different from the source node (see Figure 4.4a). The second round is responsible for all nodes that exhibit a horizontal distance of 1 (see Figure 4.4b) and the third and last round of the example transports flits to the nodes with a distance of 2 (see Figure 4.4c). These are in total 72 paths connecting each node to each other node of the NoC.

The constraint presented in Condition 4.17 ensures that each node sends at most one flit per TDM cycle. Thus, even though each node owns the beginning of $n^2 - 1 = 8$ paths, it can only use one of those paths in each TDM cycle.

Though one TDM cycle needs three rounds in this example, there are four rounds illustrated and occupied by the schedule in Figure 4.4. This is caused by the separated depiction of the schedule. Typically, the illustrated fourth round (t_9 to t_{11}) can be overlapped with the first round (t_0 to t_2) of the subsequent TDM cycle.

Subsequently, the focus is put on the schedule's characteristic of relaxed conflict-freedom. Therefore, we firstly show a lemma that gives a statement about the time the pure horizontal or vertical delivery of a path in the One-to-All schedule needs to finish. Afterwards, it is possible to show the relaxed conflict-freedom of the schedule.



(a) The paths starting in the first round with horizontal delivery to nodes of distance 0 from t_0 to t_2 and vertical delivery to nodes of all distances from t_3 to t_5.

(b) The paths starting in the second round with horizontal delivery to nodes of distance 1 from t_3 to t_5 and vertical delivery to nodes of all distances from t_6 to t_8.

(c) The paths starting in the third round with horizontal delivery to nodes of distance 2 from t_6 to t_8 and vertical delivery to nodes of all distances from t_9 to t_{11}.

Figure 4.4: Global view on the One-to-All schedule. On top of each illustration the considered NoC topology is depicted and the black vertical lines mark the timing behaviour of each node. The time goes from top to bottom and is displayed as discrete values in terms of time slots. The paths beginning in one of the schedule's rounds are illustrated in separate subfigures for a better readability. Furthermore, also to support readability, the paths of only one column of the NoC are shown and all paths beginning at the same node are illustrated in an equal colour. The paths of the remaining columns are analogous to the showed column. Please note, that the vertical delivery of the paths of one round happens concurrently to the horizontal delivery of the paths of the next round due to overlapping.

Lemma 4.2. *The complete delivery of a flit along one direction (horizontal or vertical) of a One-to-All schedule's path is always finished within one round ($t_{delivery} \leq t_{round} - t_{start}$). Thereby, $t_{delivery}$ denotes the time needed for the actual flit delivery, t_{round} marks the duration of the round and t_{start} is the period of time until the delivery starts relative to the beginning of the round.*

Proof. For each torus like topology we consider that there are n nodes for each direction (aka the nodes per horizontal or vertical ring). Furthermore, a flit is traversed from the sender's NI to a buffer in the corner router for horizontal delivery and from the buffer in the corner router to the receiver's NI for the vertical delivery. Thus, for both directions we need to consider latencies for $n - 1$ traversed physical links and one insertion to either the buffer of the corner router or the receiver's NI (see assumptions about the hardware in Section 4.1). Since we assume a bufferless forwarding from one router to the next within a ring, the transport of a flit along this distance takes the hop latency L^{hop} for each traversed

physical link, which is $(n-1)L^{hop}$ plus one hop latency for the insertion to one of the buffers. As each time slot exhibits a latency of L^{hop} and each round takes n time slots we can state $t_{round} = n \cdot L^{hop}$. Thus, since all flits are sent in the first time slot of a round, we can state that:

$$t_{delivery} = (n-1+1) \cdot L^{hop} = n \cdot L^{hop} \leq n \cdot L^{hop} = nL^{hop} - 0L^{hop} = t_{round} - t_{start}$$

□

Theorem 4.1. *The One-to-All TDM schedule is relaxed conflict-free according to Definition 2.26.*

Proof. In each round the One-to-All schedule concurrently performs one horizontal and one vertical phase and the schedule consists of n rounds. Thus, the proof about relaxed conflict-freedom of the schedule is split into three parts. These are the conflict-freedom of (1) the horizontal and (2) the vertical phase and finally the last part (3) examines the concurrent execution of both phases.

Part 1 (horizontal phase):
Following Equation 4.15 there are n paths for each node in each round and all of them start at the same time slot $s^t_{r \cdot n}$. However, the schedule's Constraint 4.17 ensures that in one round at most one path of each node is horizontally active. Thus, it follows from Corollary 4.2 that all active paths in a round exhibit disjunct source nodes. As in the horizontal phase no path changes the direction of delivery and thus, stay in the same horizontal ring, Lemma 4.1 ensures the relaxed conflict-free delivery and Lemma 4.2 ensures the finishing in time of all flits and paths.

Part 2 (vertical phase):
Equation 4.15 restricts the paths that start in the same round r to exhibit the same horizontal distance of r physical links from the source to the target node. Furthermore, Lemma 4.2 ensures that each flit reaches its corner router within the one round. Thus, the horizontal delivery of two rounds cannot overlap. Finally the constraint of Equation 4.17 requires that each node sends at most one flit per round. Then, we can state that each flit targets a different corner router. Then, there is at most one flit stored in each node at the end of the round.

Since the end of a round is the same point in time as the beginning of the next round, the number of flits per node holds for the beginning of the vertical delivery. The paths ensure the injection of flits to the vertical rings in the first time slot of the round that is responsible for the vertical transport (see also Equations 4.9 and 4.15).

$$s^t_{rn} + \Delta x + t_{Buf} = rn + r + n - r = (r+1)n = s^t_{(r+1)n}$$

Now, Lemma 4.1 guarantees the relaxed conflict-free transport in vertical direction and Lemma 4.2 causes the vertical delivery to finish within the regarded round.

Part 3 (concurrent horizontal and vertical phase):
Corollary 4.1 directly implies that the concurrent execution of one horizontal and one vertical phase of the One-to-All schedule is relaxed conflict-free. □

All-To-One TDM Schedule

The All-to-One schedule can be seen as the counterpart to the One-to-All schedule. It restricts the number of flits delivered within one TDM cycle on the receiver side by allowing each node to receive at most one flit per TDM cycle. Similar to its counterpart the All-to-One schedule consists of n rounds for a NoC of $n \times n$ nodes. Again all rounds need n time slots, each with a length of the slot latency L^{slot} which is the same as the hop latency L^{hop}. Thus, the period of the All-to-One schedule is equal to the One-to-All schedule.

$$T_{a1} = n \cdot n \cdot L^{slot} = n^2 \cdot L^{hop} \tag{4.18}$$

The schedule follows the definitions of the section about the common features of the schedules. Thus, we can define the set of the schedule's routes and the size of this set equal to those definitions.

$$Rt_{a1}^{tdm} = Rt_{Common}^{tdm} \tag{4.19}$$

$$|Rt_{a1}^{tdm}| = |P_{a1}^{tdm}| = |Rt_{Common}^{tdm}| = n^4 - n^2 \tag{4.20}$$

Another similarity that is shared with the other schedules is the restriction of the routes to the strict dimension-ordered routing and the buffering of flits at corner routers until the beginning of the next round. Thus, the shape of a path is the same as for the other schedules and follows $\vec{p}_{i,j}$ of Equation 4.9 and P_{total}^{tdm} of Equation 4.10.

$$P_{a1}^{tdm} \subset P_{total}^{tdm} \tag{4.21}$$

Apart from the constraint about the allowed number of received flits, the main difference to the other schedules is the points in time when the paths start at their source nodes. In contrast to the One-to-All schedule, in the All-to-One schedule the beginnings of paths are not restricted to a subset of time slots, but there are path beginnings of paths in each time slot. Thereby, the paths can be grouped according to the distances of their destination nodes. In each round r all nodes may send flits to the target nodes with a vertical distance of $\Delta y = n - 1 - r$ between the sender and receiver node. Furthermore, the paths of each round can be subdivided in the paths that start at the same time slot. Thereby, all nodes may send a flit in the q-th time slot of round r to the node with a horizontal distance of $\Delta x = n - 1 - q$. Hence, the horizontal delivery is similar to the procedure depicted in Figure 4.3b. Equation 4.22 displays all paths starting at a time slot $s_{r \cdot n+q}^{t}$. It uses the index $r \in [0, n-1]$ to mark the r-th round of a TDM cycle and the index $q \in [0, n-1]$ stands for the q-th time slot of the considered round.

$$\begin{aligned} P_{r \cdot n+q}^{s,a1} = \{\vec{p} \mid\ & \vec{p} \in P_{a1}^{tdm}, \exists x, y \in [0, n-1] : \vec{r}_{\vec{p}}.n^{src} = PE(x, y) \\ & \wedge \vec{r}_{\vec{p}}.n^{dst} = PE(u, w) = PE((x + n - 1 - q) \bmod n, (y + n - 1 - r) \bmod n) \\ & \wedge (\exists h_{cin}, h_{cout} \in \vec{p} : (h_{cin}.ph^{dst} = h_{cout}.ph^{src} = PE(x, w)) \\ & \quad \wedge (\vec{p}.h_{cin}.t + L_{h_{cin}.ph}^{hop} + L^{slot} = \vec{p}.h_{cout}.t))\} \end{aligned} \tag{4.22}$$

In Equation 4.22 the first line gives the point in time $s_{r \cdot n+q}^{t}$ at which the defined paths start and additionally states the source node of the paths. Afterwards, the second line defines the destination node of each path which is described by referring to the intended node's

position. It follows the determination of the corner router's node in line three and finally line four states the time a flit on the regarded path is buffered in the corner router. In the All-to-One schedule the time for buffering in the corner router is L^{slot} for all paths, as the points in time of starting a path in conjunction with the traversed horizontal distance leads to an arrival at the corner routers in the last time slot of the current round. Finally, the vertical delivery directly begins in the first time slot of the next round (similar to Figure 4.3a) what causes a buffering time of one time slot or the slot latency L^{slot}, respectively.

No other paths than the ones stated in Equation 4.22 are allowed in the described schedule. Hence, we can state the schedule's set of paths as follows.

$$P_{a1}^{tdm} = \bigcup_{r,q \in [0,n-1]} P_{r \cdot n+q}^{s,a1} \tag{4.23}$$

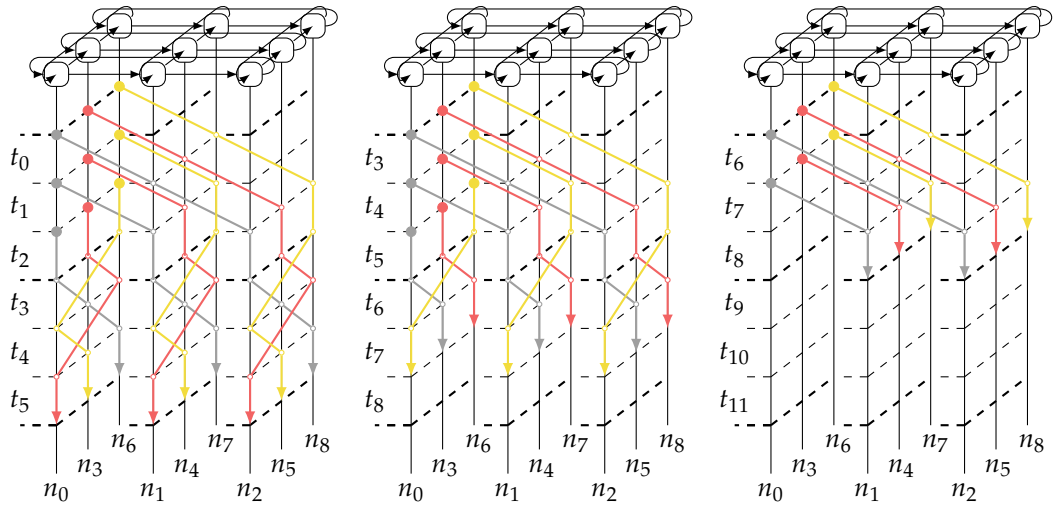
Similar to the constraint about the sender nodes in the One-to-All schedule, there is a constraint targeting the receiver nodes in the All-to-One schedule.

$$\begin{aligned} &\forall i \in \mathbb{N} : \forall \vec{p}_l \in tdm_i^C.P^{tdm} : \nexists \vec{p}_m \in tdm_i^C.P^{tdm} : \\ &\quad \vec{p}_l.h_{last}.f \neq \emptyset \wedge \vec{p}_m.h_{last}.f \neq \emptyset \wedge \vec{r}_{\vec{p}_l}.ph_{last}^{dst} = \vec{r}_{\vec{p}_m}.ph_{last}^{dst} \end{aligned} \tag{4.24}$$

Condition 4.24 is the only element in the set $tdm_{a1}^S.Co$. Since this constraint restricts all nodes to receive at most one flit per TDM cycle, there are at most as many paths applied in one TDM cycle as there are nodes in the NoC, which are n^2 paths.

Figure 4.5 illustrates the defined paths of the All-to-One schedule at an example using a 3×3 uni-directional torus NoC. In each round there are the beginnings of n paths available for each node, which means that there are 3 paths per node and round in our example. In contrast to the One-to-All schedule, which uses each round to deliver all flits to a column with a specific horizontal distance, the All-to-One schedule targets in each round the delivery to the nodes of a row with a specific vertical distance.Thus, in our example the first round exhibits all paths that target nodes with an vertical distance of 2 relative to the sender node and the subsequent rounds target rows with reduced distances. As there is no restriction for the number of active paths per sender node, no two paths of a node are allowed to start in the same time slot. Thus, the paths of a round must be distributed among the time slots available in that round. As each flit must reach the corner router until the end of the round, the paths with the longest horizontal distance starts in the first time slot, the second longest distance starts in the second time slot and the other paths follow in descending order of their horizontal distances.

When considering the paths of all nodes in the NoC, there are multiple paths for each receiver node that reach the node at the same point in time. Thus, the constraint presented in Condition 4.24 is needed to ensure the correct delivery of all flits and the absence of conflicts. Fulfilling this constraint cannot be managed by the TDM protocol itself without any additional components or agreements. One option for satisfying the condition could be to apply a lightweight synchronisation mechanism between sender and receiver to exclude the possibility of receiving flits from two different sender nodes at the same time. Such a mechanism could be designed in a way that before the sender is allowed to deliver data it must receive an acknowledgement from the intended receiver node of the data to ensure that it is the only node that is currently communicating with the regarded receiver node.



(a) The paths starting in the first round with horizontal delivery to nodes of all distances from t_0 to t_2 and vertical delivery to nodes of distance 2 from t_3 to t_5.

(b) The paths starting in the second round with horizontal delivery to nodes of all distances from t_3 to t_5 and vertical delivery to nodes of distance 1 from t_6 to t_8.

(c) The paths starting in the third round with horizontal delivery to nodes of all distances from t_6 to t_8 and vertical delivery to nodes of distance 0 from t_9 to t_{11}.

Figure 4.5: Global view on the All-to-One schedule. On top of each illustration the considered NoC topology is depicted and the black vertical lines mark the timing behaviour of each node. The time goes from top to bottom and is displayed as discrete values in terms of time slots. The paths beginning in one of the schedule's rounds are illustrated in separate subfigures for a better readability. Furthermore, also to support readability the paths of only one column are shown and all paths beginning at the same node are illustrated in an equal colour. The paths of the remaining columns are analogous to the showed column. Please note, that the vertical delivery of the paths of one round happens concurrently to the horizontal delivery of the paths of the next round due to overlapping.

There must be implemented extra hardware for this mechanism, as using standard data flits for the acknowledgement that are transported over the network controlled by the TDM protocol would violate the constraint in the same way as if no synchronisation is used. One synchronisation protocol that meets exactly the needs for this purpose and causes only marginal additional hardware costs is presented by Walter [Wal19], which is based on Frieb et al. [Fri+18]. It is intended for the usage in the One-to-One schedule but can be equally applied to the All-to-One schedule.

To show the freedom of conflicts for this schedule we need to provide several lemmas first. For this purpose we use the symbol P^r_{a1} to refer to the set of all paths that start at a dedicated round r and are active in that round.

Lemma 4.3. *All active paths $\vec{p} \in P^r_{a1}$ of a given schedule's round r reach their corner routers at the same point in time and within the same round r.*

Proof. Equation 4.22 says that the point in time when a path $\vec{p} \in P^r_{a1}$ of a dedicated round r and horizontal distance to its destination node of $\Delta x = n-1-q$ starts is given as $s^{t,r} = r \cdot n + q$ with $r, q \in [0, n-1]$. As we assume bufferless routing for horizontal forwarding (see Equation 4.4), one hop latency to insert a flit in the buffers of the corner router and a hop latency that is equal to the slot latency (see Equation 4.6), the point in time for reaching the corner router can be calculated as follows.

$$s^{t,r} + s^{t,\Delta x} + 1 = (r \cdot n + q) + (n - 1 - q) + 1 = r \cdot n + n \leq (r+1) \cdot n = s^{t,r+1}$$

As the arrival time $(r+1)n$ is independent of q, it is the same point in time for each path $\vec{p} \in P^r$. Furthermore, the calculated point in time does not exceed the timing border to the next round. Hence, the delivery finished in the same round where it starts. □

Lemma 4.4. *At the beginning of the phase for the vertical delivery of active paths $\vec{p} \in P^r$ of a round r there is at most one flit in each corner router that belongs to one of the mentioned paths P^r.*

Proof. Because of the schedule's constraint shown in Equation 4.24 each node is allowed to receive at most one flit per TDM cycle. Thus, there is at most one active path $\vec{p}_i$ per destination node.

$$\begin{aligned} & \forall \vec{p}_i, \vec{p}_j \in P^r, i \neq j : \vec{p}_i.h_{last}.ph^{dst} \neq \vec{p}_j.h_{last}.ph^{dst} \\ \Leftrightarrow\ & \forall \vec{p}_i, \vec{p}_j \in P^r, i \neq j : POS(\vec{p}_i.h_{last}.ph^{dst}).x \neq POS(\vec{p}_j.h_{last}.ph^{dst}).x \\ & \vee POS(\vec{p}_i.h_{last}.ph^{dst}).y \neq POS(\vec{p}_j.h_{last}.ph^{dst}).y \end{aligned} \tag{4.25}$$

Equation 4.22 states that all paths of a round r obtain the vertical coordinate for its destination node $w = (y + n - 1 - r) \bmod n$ when y marks the vertical coordinate of the source node. Therefore, the vertical distance from the source to the destination node is $\Delta y = n - 1 - r$ and is equal for all considered paths. Moreover, according to Equation 4.22 the corner routers' nodes n^i_{cr} of each considered path $\vec{p}_i$ obtains the same vertical distance to the target node as the source node. Thus, for a given target node $PE(u, w)$ the position of the corner router is given with $POS(n^i_{cr}) = (u_i, (w_i - (n - 1 - r)) \bmod n)$. Now from Equation 4.25 it follows:

$$\begin{aligned} & \forall \vec{p}_i, \vec{p}_j \in P^r : POS(\vec{p}_i.h_{last}.ph^{dst}) \neq POS(\vec{p}_j.h_{last}.ph^{dst}) \\ \Leftrightarrow\ & \forall \vec{p}_i, \vec{p}_j \in P^r : (x_i, y_i) \neq (x_j, y_j) \\ \Leftrightarrow\ & \forall \vec{p}_i, \vec{p}_j \in P^r : (x_i, (y_i - (n - 1 - r)) \bmod n) \neq (x_j, (y_j - (n - 1 - r)) \bmod n) \\ \Leftrightarrow\ & \forall \vec{p}_i, \vec{p}_j \in P^r : POS(n^i_{cr}) \neq POS(n^j_{cr}) \\ \Leftrightarrow\ & \forall \vec{p}_i, \vec{p}_j \in P^r : n^i_{cr} \neq n^j_{cr} \end{aligned}$$

Since no two paths $\vec{p} \in P^r$ share the same corner router, there is at most one active path per corner router at the beginning of a round. As there is one flit per active path the lemma holds. □

Lemma 4.5. *The complete delivery of a flit along the vertical direction of a All-to-One schedule's path is always finished within one round ($t_{delivery} \leq t_{round} - t_{start}$). Thereby, $t_{delivery}$ denotes the time needed for the actual flit delivery, t_{round} marks the duration of the round and t_{start} is the period of time until the delivery starts relative to the beginning of the round.*

Proof. For each torus like topology we consider that there are n nodes for each vertical ring. Furthermore, a flit is traversed from the buffer in the corner router to the receiver's NI for the vertical delivery. Thus, we need to consider latencies for at most $n-1$ traversed physical links and one insertion to the receiver's NI (see assumptions about the hardware in Section 4.1). Since we assume a bufferless forwarding from one router to the next within a ring, the transport of a flit along this distance takes the hop latency L^{hop} for each traversed physical link, which is $(n-1)L^{hop}$ plus one hop latency for the insertion to the NI. As each time slot exhibits a latency of L^{hop} and each round takes n time slots we can state $t_{round} = n \cdot L^{hop}$. Thus, since all flits are sent in the first time slot of a round, we can state that:

$$t_{delivery} = (n-1+1) \cdot L^{hop} = n \cdot L^{hop} \leq n \cdot L^{hop} = nL^{hop} - 0L^{hop} = t_{round} - t_{start}$$

□

Theorem 4.2. *The All-to-One TDM schedule is relaxed conflict-free according to Definition 2.26.*

Proof. In each round the All-to-One schedule performs concurrently one horizontal and one vertical phase and the schedule consists of n rounds. Thus, the proof about relaxed conflict-freedom of the schedule is split into three parts. These are the conflict-freedom of (1) the horizontal and (2) the vertical phase and finally the last part (3) examines the concurrent execution of both phases.

Part 1 (horizontal phase):
Suppose, for a contradiction, that the All-to-One TDM schedule is not relaxed conflict-free for the horizontal delivery. Then, a flit transported on a path $\vec{p}_1 \in P_{a1}^{tdm}$ causes a conflict with another flits that is transported by $\vec{p}_2 \in P_{a1}^{tdm}$ during its horizontal delivery. Then, both paths share a physical link in a horizontal ring of the torus network at the same time. Equations 4.4 and 4.9 state that the horizontal delivery of all paths happens completely before any vertical transport is performed. Thus, the conflict happens when both paths perform their transport from the source node to the corner router which completely consists of horizontal forwarding. Following Lemma 4.3 the delivery of both paths begin in the same round r and the flits of the paths reach their corresponding corner routers at the same point in time. Furthermore, Equation 4.9 requires the same hop latency for all paths and no buffering is applied for pure horizontal transport. Thus, the characteristics about the forwarding are the same for both paths $\vec{p}_1$ and $\vec{p}_2$.

As both paths belong to the same round, traverse the same physical link at the same point in time, the timing characteristics about horizontal forwarding are equal and both paths must reach their corner router at the same time, it follows that $\vec{p}_1$ and $\vec{p}_2$ must obtain the same corner router. Then, there are two active paths in the same corner router at the end of the horizontal delivery and thus, also at the beginning of the phase for the vertical delivery.

This is a contradiction to Lemma 4.4.

Part 2 (vertical phase):
Following Lemma 4.4 there is at most one flit and thus active path in each router at the beginning of the phase for vertical delivery. Furthermore, Equation 4.22 states that the flit of

each path starts its vertical delivery directly at the beginning of this phase. Then, the vertical delivery of the All-to-One schedule is conflict-free following Lemma 4.1. Furthermore, Lemma 4.5 ensures the finishing of the phase in the same round and therefore, prohibits overlapping of successively executed vertical phases.

Part 3 (concurrent horizontal and vertical phase):
Corollary 4.1 directly implies that the concurrent execution of one horizontal and one vertical phase of the All-to-One schedule is relaxed conflict-free. □

One-To-One TDM Schedule

The One-to-One schedule combines the restrictions of the One-to-All and All-to-One schedule. Thus, it allows the nodes to only send one flit to any other node of the NoC and only receive one flit from an other node per TDM cycle. Caused by the combination of both restrictions the number of rounds per TDM cycle can be reduced, while keeping the schedule relaxed conflict-free at the same time. Thus, in contrast to the previously presented schedules the One-to-One schedule is composed of only one round of n time slots. The length of the time slots remains the same as for the other schedules, which is the slot latency L^{slot} or the hop latency L^{hop}, respectively. The reduction in the number of rounds leads to a reduced period T_{11}.

$$T_{11} = n \cdot L^{slot} = n \cdot L^{hop} \tag{4.26}$$

Despite the reduced period the schedule consists of the same number of paths as the other general-purpose state-of-the-art schedules. Furthermore, they consist of the same pairs of source and destination nodes. Thus, the definition of the schedule's set of routes, is equal to the definitions of the previous schedules.

$$Rt_{11}^{tdm} = Rt_{Common}^{tdm} \tag{4.27}$$

$$|Rt_{11}^{tdm}| = |P_{11}^{tdm}| = |Rt_{Common}^{tdm}| = n^4 - n^2 \tag{4.28}$$

Similar to previously described schedules the One-to-One schedule restricts the routes to the strict dimension-ordered routing and the buffering of flits at corner routers until the next round begins. This restriction causes the paths to exhibit a very similar shape as the other schedules and follows $\vec{p}_{i,j}$ of Equation 4.9 and P_{total}^{tdm} of Equation 4.10.

$$P_{11}^{tdm} \subset P_{total}^{tdm} \tag{4.29}$$

In the One-to-One schedule each path starts in the first time slot s_0^t of a TDM cycle. This set of paths is defined in Equation 4.30. No other paths than stated in this equation are allowed within one TDM cycle.

$$\begin{aligned} P_0^{s,11} = \{\vec{p} \;\; | \;\; & \vec{p} \in P^{tdm}, \exists x, y, u, w \in [0, n-1], x \neq u \vee y \neq w : \\ & (\vec{r}_{\vec{p}}.n^{src} = PE(x,y) \wedge \vec{r}_{\vec{p}}.n^{dst} = PE(u,w)) \\ & \wedge (\exists h_{cin}, h_{cout} \in \vec{p} : h_{cin}.ph^{dst} = h_{cout}.ph^{src} = PE(x,w) \\ & \wedge \vec{p}.h_{cin}.t + L_{h_{cin}.ph}^{hop} + (1 + 2n - (2 + \Delta x + \Delta y)) \cdot L^{slot} = \vec{p}.h_{cout}.t)\} \\ = & P_{11}^{tdm} \end{aligned} \tag{4.30}$$

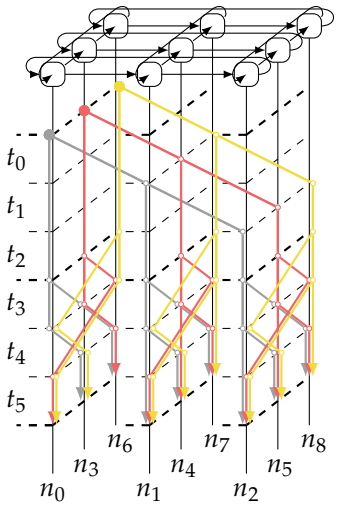


Figure 4.6: Global view on the One-to-One schedule with one round and TDM cycle with a horizontal delivery to nodes of all distances from t_0 to t_2 and vertical delivery to nodes of all distances from t_3 to t_5. The illustration depicts on top the considered NoC topology and the black vertical lines mark the timing behaviour of each node. The time goes from top to bottom and is displayed as discrete values in terms of time slots. Furthermore, to support readability the paths of only one column are shown and all paths beginning at the same node are illustrated in an equal colour. The paths of the columns that are not illustrated behave analogous to the shown column. Please note, that the vertical delivery of the paths of one round happens concurrently to the horizontal delivery of the paths of the next round due to overlapping.

In Equation4.30 the second line defines the source and destination nodes of each path and the third line identifies the corner node. The fourth line states the time a flit is buffered in the corner router. Thereby, $\Delta x = (u - x) \bmod n$ marks the horizontal distance between the source and destination node, while $\Delta y = (w - y) \bmod n$ states the corresponding vertical distance. The applied time for buffering is composed of the storage time of the horizontal phase $n - 1 - \Delta x$ and the buffering time of the vertical phase, which is $n - 1 - \Delta y$. Additionally, the buffering of $1 \cdot L^{slot}$ is needed to perform the transition of the horizontal to the vertical delivery. Hence, the horizontal delivery is similar to the procedure depicted in Figure 4.3a, while the vertical transportation follows Figure 4.3b.

As already mentioned the One-to-One schedule's set of constraints $tdm^S.Co$ is a union of the sets of the One-to-All and All-to-One schedules.

$$tdm^S_{11}.Co = tdm^S_{1a}.Co \cup tdm^S_{a1}.Co \tag{4.31}$$

Following Equation 4.31 the set of constraints of the One-to-One schedule contains the constraints defined in Equations 4.17 and 4.24. These constraints force a TDM cycle to apply at most n^2 paths and no two path are allowed to share neither their source node nor their target node.

Figure 4.6 illustrates the defined paths of the One-to-One schedule at an example using

a 3×3 uni-directional torus NoC. As the described schedule exhibits only one round, all paths of this schedule start in that round. Thus, there is one paths from each node to each other node of the network, which are $n^2 - 1 = 8$ paths per node. Nevertheless, each sender node is only allowed use one of those paths per TDM cycle and no two active paths are allowed to target the same destination node in the same TDM cycle. As each node may only send one flit per round and TDM cycle, they can be injected to the NoC in the first time slot of the round and than forwarded horizontally without producing conflicts. After finishing horizontal transport, the flits are stored in their corner routers until the begin of the next round. Since there is no restriction to the traversed horizontal distance, there are possibly multiple flits in one router when vertical transport starts. However, as the flits were only delivered horizontal and there is only one flit per source node in the network, there are at most n flits in one router. Additionally, the constraint about the number of received flits per node and TDM cycle guarantees that there are no more than n flits present in one complete column of the network at that point in time.

As there are n flits arbitrarily distributed on n nodes of a vertical ring, an appropriate strategy for delivery is needed that is sufficient for all possible assignments. To guarantee that no conflicts occur, we apply the same strategy as with the All-to-One schedule in the horizontal delivery. Hence, each flit is injected to the NoC in a time slot $n - 1 - \Delta y$ if Δy marks the vertical distance of the destination node to the corner router. This causes all flits to a separate path along the vertical ring and to arrive at the receiver node exactly in the last time slot of the round.

Like in the other schedules the delivery is split into a horizontal and a vertical delivery whereby the vertical delivery takes place in the round that is successively following the round for the horizontal delivery. Thus, the horizontal delivery of the current TDM cycle is overlapped with the vertical delivery of the previous TDM cycle.

Lemma 4.6. *All active paths $\vec{p} \in P^r$ of a given One-to-One schedule's round r finish their horizontal delivery within the same round ($t_{delivery} \leq t_{round} - t_{start}$). Thereby, $t_{delivery}$ denotes the time needed for the actual flit delivery, t_{round} marks the duration of the round and t_{start} is the period of time until the delivery starts relative to the beginning of the round.*

Proof. For each torus like topology we consider that there are n nodes per horizontal ring. Furthermore, a flit is traversed from the sender's NI to a buffer in the corner router for horizontal delivery. Thus, we need to consider latencies for $n - 1$ traversed physical links and one insertion to the buffer of the corner router (see assumptions about the hardware in Section 4.1). Since we assume a bufferless forwarding from one router to the next within a ring, the transport of a flit along this distance takes the hop latency L^{hop} for each traversed physical link, which is $(n-1)L^{hop}$ plus one hop latency for the insertion to the buffer in the corner router. As each time slot exhibits a latency of L^{hop} and each round takes n time slots we can state $t_{round} = n \cdot L^{hop}$. Thus, since all flits are sent in the first time slot of a round, we can state that:

$$t_{delivery} = (n - 1 + 1) \cdot L^{hop} = n \cdot L^{hop} \leq n \cdot L^{hop} = nL^{hop} - 0L^{hop} = t_{round} - t_{start}$$

□

Lemma 4.7. *All active paths $\vec{p} \in P^r$ starting at a given One-to-One schedule's round r reach their destination node at the same point in time and within round $r+1$ that directly follows round r.*

Proof. From Equation 4.30 we can deduce the point in time when a path $\vec{p} \in P^r$ of a dedicated round r and vertical distance Δy to its destination node starts its vertical delivery. Hence, the point in time for the start of the vertical delivery of a path to a destination with $\Delta y = n - 1 - q$ is given as follows ($q \in [0, n-1]$).

$$\begin{aligned} s^{t,rv} &= s^{t,r} + \Delta x + t_{buf} \\ &\overset{(4.30)}{=} rn + \Delta x + 1 + 2n - (2 + \Delta x + \Delta y) \\ &= r \cdot n + n + q \end{aligned}$$

Thereby, $s^{t,rv}$ indicates the point in time when a paths from round r starts its vertical delivery and $s^{t,r}$ marks the time of the start of round r and is $s^{t,r} = rn$. As we assume bufferless routing for vertical forwarding (see Equation 4.4), one hop latency to insert a flit in the buffer of the receiver's NI and a hop latency that is equal to the slot latency (see Equation 4.6), the point in time for reaching the destination node can be calculated as follows.

$$s^{t,rv} + L^{\Delta y} + 1 = (r \cdot n + n + q) + (n - 1 - q) + 1 = (r+1) \cdot n + n \leq (r+1) \cdot n + n = s^{t,r+2}$$

As the arrival time $(r+1)n + n$ is independent of q, it is the same point in time for each path $\vec{p} \in P^r$. Furthermore, the calculated point in time does not exceed the timing border to the next round $r+2$. Hence, the delivery finished in the directly subsequent round from where it starts. □

Theorem 4.3. *The One-to-One TDM schedule is relaxed conflict-free according to Definition 2.26.*

Proof. In each TDM cycle the One-to-One schedule performs concurrently one horizontal and one vertical phase, as the current TDM cycle is used for the vertical delivery of the previous TDM cycle. Thus, the proof about relaxed conflict-freedom of the schedule is split into three parts. These are the conflict-freedom of (1) the horizontal and (2) the vertical phase and finally the last part (3) examines the concurrent execution of both phases.

Part 1 (horizontal phase):
Following Equation 4.30 there are $n^2 - 1$ paths for each node in each round and all start at the same time slot s_0^t at the beginning of the TDM cycle. However, the constraint of Equation 4.17 that is included in $tdm_{11}^S.Co$ ensures that at most one path of each node is horizontal active in a given TDM cycle. Thus, the horizontal delivery of the One-to-One schedule is conflict-free following Lemma 4.1 and according to Lemma 4.6 it reaches the corner router within the same round.

Part 2 (vertical phase):
Suppose, for a contradiction, that the One-to-One TDM schedule is not relaxed conflict-free for the vertical delivery. Then, a flit transported on a path $\vec{p}_1 \in P_{11}^{tdm}$ causes a conflict with another flit that is transported by $\vec{p}_2 \in P_{11}^{tdm}$ during its vertical delivery. Then, both paths share a physical link in a vertical ring of the torus network. Equations 4.4 and 4.9 state that

the vertical delivery of all paths are completely executed after all horizontal transports. Thus, the conflict happens when both paths perform their transport from the corner router to the destination node which completely consists of vertical forwarding. Following Lemma 4.7 the delivery of both paths begin in the same round r and the flits of the paths reach their corresponding destination node at the same point in time. Furthermore, Equation 4.9 requires the same hop latency for all paths and no buffering is applied for pure vertical transport. Thus, the characteristics about the forwarding is the same for both paths $\vec{p}_1$ and $\vec{p}_2$.

As both paths belong to the same round, traverse the same physical link at the same time, the timing characteristics about vertical forwarding are equal and both paths must reach their destination node at the same time, it follows that $\vec{p}_1$ and $\vec{p}_2$ must obtain the same destination node.

This is a contradiction to the constraint stated in Equation 4.24, which belongs to the One-to-One schedule's set of constraints $tdm^S_{11}.Co$.

Part 3 (concurrent horizontal and vertical phase):
Corollary 4.1 directly implies that the concurrent execution of one horizontal and one vertical phase of the One-to-One schedule is relaxed conflict-free. □

All-To-All TDM Schedule

The All-to-All schedule is already described in Section 2.2.2 in an informal way. Here it follows the formal definition of the schedule's components T_{aa}, P^{tdm}_{aa} and Co_{aa}. In addition to the pure definitions we provide an example for the assumed NoC and a proof about the schedule's correctness in terms of conflict freedom is presented. Please note that we slightly adapt the schedule of Mische and Ungerer [MU14] here to fit to our hardware model. In concrete we additionally consider the latency needed for inserting a flit to a buffer in the corner router or the NI at a receiver node.

As stated in Section 2.2.2 the All-to-All schedule allows each node to send and receive one flit to and from each other node within one TDM cycle. Hence, the number of rounds in one TDM cycle depends on the number of nodes per NoC dimension. More precisely it is half of the number of nodes per dimension or it is rounded to the next higher integer. The timing for a round is caused by the two performed deliveries of n flits each, one to a short horizontal distance $\Delta x = a$ with $a \in [0, \lceil \frac{n}{2} \rceil]$ and one to a long distance $\Delta x = (n - 1 - a)$. In the remainder of this work we call each of those deliveries a half round. Thereby, when considering a specific round r in a TDM cycle the value of a is set to $a = r$ to reveal the actual horizontal distances. For the calculation of the latency of a complete round we add up the terms about the horizontal distances of both half rounds and consider the times for injecting a flit to the buffers, too. Hence, each round consists of $n(a + 1) + n(n - 1 - a + 1) = n^2 + n$ time slots. The only exception builds the last round of a schedule for a NoC if there is an odd number of nodes per dimension (e.g. a NoC size of 3×3). In those cases the last round of a TDM cycle takes only half the time slots of the previous rounds, as it only needs to process one half round to the horizontal distance of $\lceil \frac{n}{2} \rceil$. According to Section 2.2 the length of each time slot is determined by the slot latency L^{slot} and the assumed hardware says that this latency is equal to the hop latency L^{hop}. Therefore, there is the following period T_{aa} for

the schedule tdm^S_{aa}.

$$T_{aa} = \frac{n}{2} \cdot (n^2 + n) \cdot L^{slot} = \frac{n}{2} \cdot (n^2 + n) \cdot L^{hop} \tag{4.32}$$

When taking a closer look to the applied routes and paths, one can see that the routes are the same as for the common case of Equations 4.7 to 4.10. Thus, each path follows the characteristics of $\vec{p}_{i,j}$ in Equation 4.9 and P^{tdm}_{total} in Equation 4.10.

$$Rt^{tdm}_{aa} = Rt^{tdm}_{common} \tag{4.33}$$

$$|Rt^{tdm}_{aa}| = |P^{tdm}_{aa}| = |Rt^{tdm}_{common}| = n^4 - n^2 \tag{4.34}$$

$$P^{tdm}_{aa} \subset P^{tdm}_{total} \tag{4.35}$$

Subsequently, we refer to a specific time slot s^t used for the beginning of a path in a All-to-All schedule's TDM cycle by using three variables. These are the current round of the slot $r \in [0, \frac{n}{2}]$, the current half round $d \in \{0, 1\}$ used for the delivery of flits to a given horizontal distance and the indicated path within a given round and half round $q \in [0, n-1]$. As the data transport in a round consists of two half rounds each targeting flit deliveries to nodes with different horizontal distances, we use $d = 0$ to refer to the first half round with the short distance and $d = 1$ marks the second half round to the long distance. Thus, the indices for the time slot $s^t_{ind_{aa}}$ in which a All-to-All schedule's path begins is given as follows.

$$\begin{aligned} ind_{aa}(r, d, q) = {} & r(n^2 + n) \\ & + dn(r + 1) \\ & + (1 - d)(r + 1)q + d(n - r)q \end{aligned} \tag{4.36}$$

Thereby, the first addend marks the offset for the current round, the second addend gives the offset for the current half round and the last line of Equation 4.36 calculates the time slot within the current round and half round. The first addend considers the duration of a round $(n^2 + n)$ for each previously executed round, which are r rounds. The term for the current half round is multiplied with the factor d, as only the second half round must consider an offset here. Furthermore, it uses a factor n as there are n paths in one half round. Additionally, r gives the first half round's horizontal distance to the corner router and thus, $(r + 1)$ marks the time that a previous path needs to traverse its way on the horizontal ring. There are two terms for the calculation of the actual time slot in the third line, as there is a different calculation depending on the current half round ($d = 0$ or $d = 1$). The distinction of both cases is realized by applying the factors d and $(1 - d)$. The terms $(r + 1)$ and $(n - r)$ are used to describe the horizontal distance of a path in the current round and half round and mark the duration until the NoC is free to deliver the next flits. Finally, q indicates the q-th path executed in the current round and half round and thus, determines how many previously executed paths must be considered.

In addition to the time slots' indices we need to refer to a specific path in the schedule with a successively increasing index z.

$$z = r \cdot 2n + dn + q \tag{4.37}$$

The offset for each round is here $2n$, as there are $2n$ paths in each round (except the last one for odd NoC dimensions) and similarly there are n paths in each half round. Thus, for example a schedule's path in the third time slot ($q = 3$) of round $r = 2$ and in the second half round ($d = 1$) is identified with $z = 2 \cdot 2n + 1 \cdot n + 3 = 5n + 3$ and given with $\vec{p}_z = \vec{p}_{5n+3}$.

Each used time slot $s^t_{ind_{aa}}$ obtains exactly one path per node. Thereby, Equations 4.38 and 4.39 show the horizontal and vertical coordinate of the corresponding destination node $n^{dst} = PE(u^{aa}, w^{aa})$ for a given source node $n^{src} = PE(x, y)$. Both equations are given modulo to n since all rows and columns of the NoC are implemented as horizontal or vertical rings.

$$\begin{aligned} u^{aa} = (&x \\ &+(1-d)r \\ &+d(n-1-r)) \bmod n \end{aligned} \tag{4.38}$$

Equation 4.38 targets the destination node's horizontal coordinate u^{aa}. It takes the source node's horizontal coordinate x as starting point and adds the needed offset, which is given with two addends. Thereby, the factors $(1-d)$ and d define which of the terms actually effect the calculation according to the current half round. The first term $(1-d)$ is used for the paths of the first half round ($d = 0$) with the short distances r, while the second term d calculates the coordinate for the second half round ($d = 1$) with long distances $n-1-r$.

$$\begin{aligned} w^{aa} = \Big(&y \\ &+(1-(q \bmod 2)) \left(n-1-\left\lfloor \frac{q}{2} \right\rfloor\right) \\ &+(q \bmod 2) \left\lfloor \frac{q}{2} \right\rfloor \qquad\qquad \Big) \bmod n \end{aligned} \tag{4.39}$$

Similar to the horizontal destination the vertical destination coordinate takes the source node's vertical coordinate y as starting point and calculates the coordinate using offsets. The vertical delivery alternates between a path with a long distance $\Delta y = n-1-b$ and one with a short distance $\Delta y = b$ where b is a variable with $b \in [0, \lceil \frac{n}{2} \rceil]$. We use the term $\lfloor \frac{q}{2} \rfloor$ to refer to a specific pair consisting of those two parts. This term grows with a value of 1 when q raises with 2. Additionally, the term $(q \bmod 2)$ is applied to determine if the q-th path is the first ($q \bmod 2 = 0$) or the second path ($q \bmod 2 = 1$) of such a pair. Hence, the second addend in Equation 4.39 uses $(1-(q \bmod 2))$ to calculate the offset for the first path of each pair and causes with the term $(n-1-\lfloor \frac{q}{2} \rfloor)$ the consideration of a long vertical distance. On the other hand due to the factor $(q \bmod 2)$ the third addend refers to the pair's second path and adds up a short vertical distance $(\lfloor \frac{q}{2} \rfloor)$ to the vertical destination coordinate w^{aa}.

To give a full definition of the All-to-All schedule's paths there remains the determination of the buffering times in the corner routers for each path. For the calculation of those times we take the point in time when a considered path $\vec{p}_z$ starts the vertical delivery t^{vin}_z and subtract the point in time when the path finishes its horizontal delivery t^{hout}_z. Both times use the start of the TDM cycle as reference point.

$$\Delta t^{aa}_z = t^{vin}_z - t^{hout}_z \tag{4.40}$$

The starting point in time for the vertical delivery t^{vin}_z is the time slot that is located directly after the finishing of the previously executed path $\vec{p}_{z-1}$ or after finishing the horizontal

delivery of the first half round if we consider the TDM cycle's first path. This first half round takes $n(\Delta x + 1)L^{hop} = n(0+1)L^{hop} = nL^{hop}$. Therefore, we determine the finishing time of the previous path by calculating the time needed for all previous vertical deliveries and adding an offset of n to obtain t_z^{vin}.

$$\begin{aligned} t_z^{vin} = \Bigg(& n \\ & + \left\lfloor \frac{z}{n} \right\rfloor \frac{n(n+1)}{2} \\ & + \left\lfloor \frac{z \bmod n}{2} \right\rfloor (n+1) \\ & + ((z \bmod n) \bmod 2) \left(n - \left\lfloor \frac{z \bmod n}{2} \right\rfloor \right) \Bigg) L^{slot} \end{aligned} \tag{4.41}$$

Thereby, the term $\left\lfloor \frac{z}{n} \right\rfloor$ is used to count the previously executed half rounds. Since in each half round there are vertical deliveries to all distances, the duration of each half round's vertical delivery is given by $\sum_{\Delta y=0}^{n-1}(\Delta y + 1) = \frac{n(n+1)}{2}$. Furthermore, the term $\left\lfloor \frac{z \bmod n}{2} \right\rfloor$ is used similar to $\left\lfloor \frac{q}{2} \right\rfloor$ in Equation 4.39 and refers to a specific pair of paths consisting of paths with a long and a short vertical distance Δy. Likewise, the term $((z \bmod n) \bmod 2)$ is used to indicate one path of the pair. Thereby, the operation modulo n in the terms $\left\lfloor \frac{z \bmod n}{2} \right\rfloor$ and $((z \bmod n) \bmod 2)$ is needed to stay consistent if z exceeds the value of n and n is an odd number. In summary the first addend of Equation 4.41 states the constant offset of n for the first path and the second addend effects the vertical delivery's duration for all previously executed half rounds. The third term takes all executed pairs of paths of the current half round into account and the fourth addend determines the time needed for the previously performed path of the current pair, if present.

The points in time for finishing the horizontal delivery can be easily calculated by applying Equation 4.36. This equation is used to identify the time slot for the beginning of a path which is exactly at the point in time when the previous path has finished its horizontal delivery. With the obtained time slot the duration of the path for the horizontal delivery $(u^{aa} - x) \bmod n$ must be added to obtain the correct time t_p^{hout}.

$$t_p^{hout} = s_{ind_{aa}}^t.t + ((u^{aa} - x) \bmod n)L^{slot} \tag{4.42}$$

With the previously presented Equations 4.38, 4.39 and 4.40 it is possible to define the sets of paths that group all paths according to their starting points in terms of time slots. Thus, $P_{ind_{aa}}^{s,aa}$ gives the set of all paths starting at time slot $s_{ind_{aa}}^t$.

$$\begin{aligned} P_{ind_{aa}}^{s,aa} = \{ \vec{p} \mid \vec{p} \in P^{tdm}, \exists x, y \in [0, n-1] : \\ (\vec{r}_{\vec{p}}.n^{src} = PE(x,y) \wedge \vec{r}_{\vec{p}}.n^{dst} = PE(u^{aa}, w^{aa})) \\ \wedge (\exists h_{cin}, h_{cout} \in \vec{p} : h_{cin}.ph^{dst} = h_{cout}.ph^{src} = PE(x, w^{aa}) \\ \wedge \vec{p}.h_{cin}.t + L_{h_{cin}.ph}^{hop} + \Delta t_z^{aa} = \vec{p}.h_{cout}.t) \} \end{aligned} \tag{4.43}$$

Equation 4.43 describes all paths that are included in the All-to-All schedule. Therefore, the

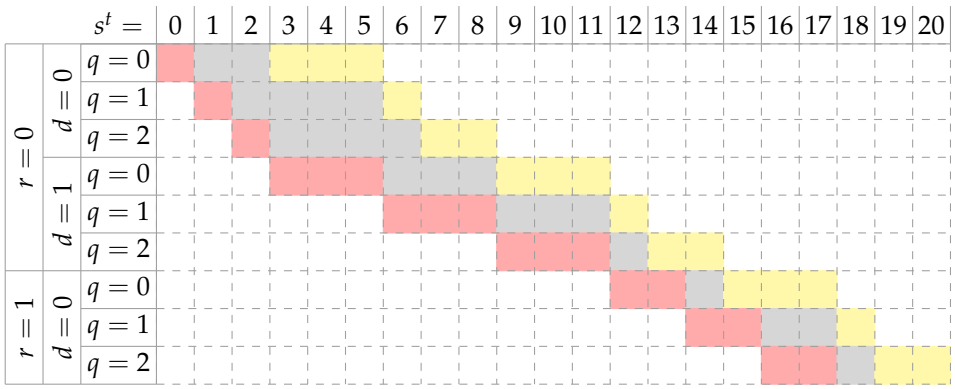
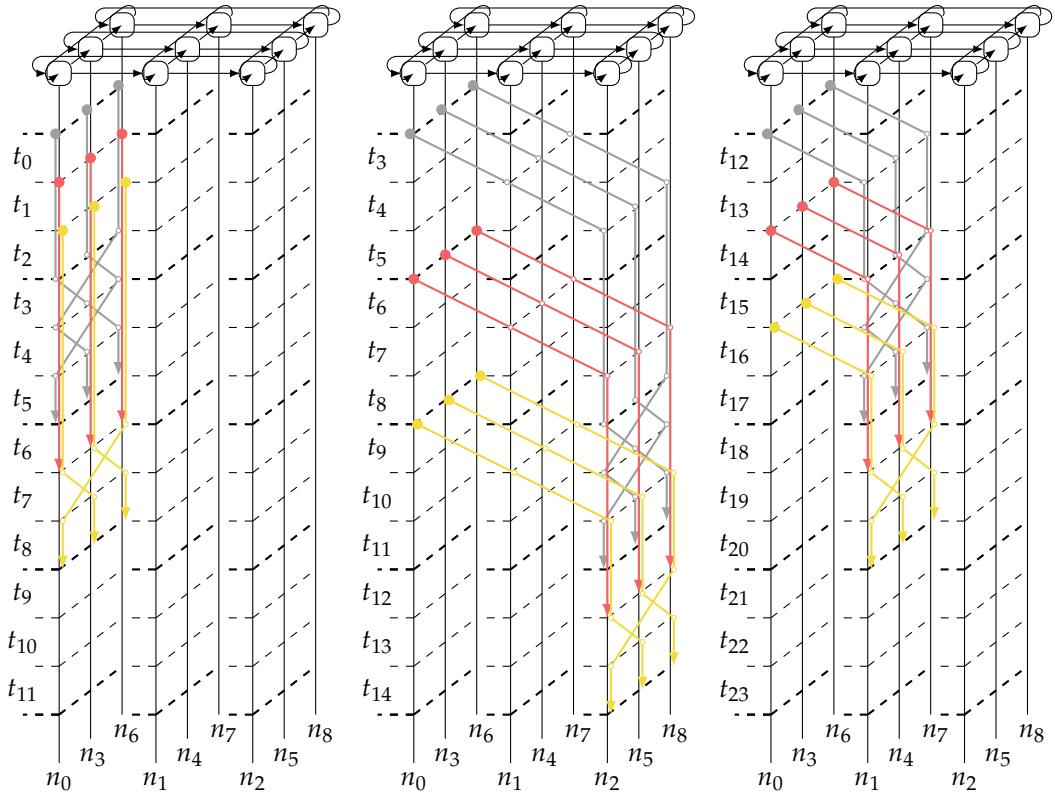


Figure 4.7: The execution of one TDM cycle for one node in the All-to-All schedule. Each row indicates one path starting at that node while each column marks one time slot. The paths are referred to by giving their round r, half round d and the path of the regarded half round q. The colours indicate the current execution state of a path. The red areas indicate that the path is horizontally active at that time, while the yellow areas indicate vertical deliveries. Finally, the gray fields indicate that the path is waiting in the corner router. Please note that the paths of all nodes show the same timing behaviour as displayed here.

full set of paths of this schedule is given as follows.

$$P_{aa}^{tdm} = \bigcup_{\substack{r \in \left[0;\frac{n}{2}\right] \\ d \in \{0;1\} \\ q \in [0;n-1]}} P_{ind_{aa}(r,d,q)}^{s,aa} \tag{4.44}$$

Figure 4.7 and 4.8 illustrate the execution of the All-to-All schedule at an example using a 3×3 unidirectional torus NoC. Thereby, Figure 4.8 targets the arrangement of the paths across the complete NoC at a global view, while Figure 4.7 focuses on the arrangement and execution of the paths of one node in detail to provide deeper insights in the schedule's execution. The illustrated schedule exhibits $\left\lceil \frac{n}{2} \right\rceil = \left\lceil \frac{3}{2} \right\rceil = 2$ rounds. The first round is split in two half rounds, while the second one consists of one half round. Every half round is responsible for the delivery of flits to one specific horizontal distance Δx from all source nodes to their corresponding destination nodes. As there are $n = 3$ nodes with the same horizontal distance for each node, there are 3 paths per half round for each node. Since there is no constraint $co \in Co_{aa}$ present in this schedule, all these paths can be utilized concurrently in the same TDM cycle. Thus, they must be arranged in a way that no conflicts can occur. Hence, for each node the first path of a half round starts in the first time slot of that half round. Afterwards, the subsequent paths always start not before the horizontal delivery of the previous paths have finished to ensure that the current path does not conflict with previous ones of other nodes. Furthermore, each half round begins directly after the finishing of the horizontal delivery of the previous half round which leads to an overlapping of the horizontal and vertical delivery of subsequent paths, half rounds, rounds and TDM cycles.



(a) The paths starting in the first round and half round with horizontal delivery to nodes of distance 0 from t_0 to t_2 and vertical delivery to nodes of all distances from t_3 to t_8.

(b) The paths starting in the first round and second half round with horizontal delivery to nodes of distance 2 from t_3 to t_{11} and vertical delivery to nodes of all distances from t_9 to t_{14}.

(c) The paths starting in the second round with horizontal delivery to nodes of distance 1 from t_{12} to t_{17} and vertical delivery to nodes of all distances from t_{15} to t_{20}.

Figure 4.8: Global view on an example of the All-to-All schedule. On top of each illustration the considered NoC topology is depicted and the black vertical lines mark the timing behaviour of each node. The time goes from top to bottom and is depicted as discrete values in terms of time slots. The paths beginning in one of the schedule's half rounds are summarized and depicted separately for a better readability. Furthermore, also to support readability the paths of only one column are shown and for each round all paths beginning in the same time slot are illustrated in an equal colour. The paths of the remaining columns are analogous to the showed column. Please note, that the vertical delivery of the paths happens concurrently to the horizontal delivery of other paths due to overlapping.

In the vertical delivery the vertical distances Δy of the paths of one half round alternate similar to the alternating of the horizontal distances Δx of the half rounds in the horizontal delivery. Thus, at first a distance of $\Delta y = n - 1 - q$ is served followed by the distance of $\Delta y = q$ for each pair of paths. Similar to the horizontal delivery all paths need to be arranged in a way that no conflicts can occur. Hence, the traversal of each flit starting from a dedicated corner router starts not before the previously flit from that router has fully finished its vertical delivery.

In the remainder of this section we proof that data is forwarded conflict-free when the All-to-All schedule is applied. At first, four lemmas are provided and afterwords the theorem about conflict-freedom of the All-to-All schedule follows.

Lemma 4.8. *No path of the All-to-All schedule starts before its direct predecessor has finished its horizontal delivery.*

Proof. Equation 4.43 states that in each time slot $s^t_{ind_{aa}}$, and thus for each tuple (r, d, q) consisting of the round r, half round d and current time slot q, there begins one path for each node. The same equation states that all paths of a specific time slot share the same relative horizontal distance ($\Delta x = (u^{aa} - x) \bmod n$). Furthermore, according to the assumed hardware (see beginning of Section 4.1) the delivery of a flit in horizontal direction takes $t^h = (\Delta x + 1)L^{hop}$ and the time interval between the start of two successively executed paths is calculated by subtracting the corresponding time slots s^t_z from s^t_{z+1}. Therefore, we need to show that the following equation holds for all possible parameter combinations (r, d, q).

$$s^t_{z+1} - s^t_z \leq t^h_z \tag{4.45}$$

Since each time slot of the All-to-All schedule is dependent on the three parameters r, d and q, we need to distinguish between three cases for the differentiation of the time slots.

Case 1 ($q \to q + 1$; r and d remain equal):
This case considers two time slots s^t_z and s^t_{z+1} that are located within the same round r and half round d. Thus, the values of r and d remain equal for s^t_z and s^t_{z+1} and the value of q increases with 1 for $z + 1$. This parameter configuration results in the following equations about the terms $s^t_{z+1} - s^t_z$ and t^h.

$$\begin{aligned} s^t_{z+1} - s^t_z &= s^t_{ind_{aa}(r,d,q+1)} - s^t_{ind_{aa}(r,d,q)} \\ &\overset{(4.36)}{=} (r(n^2+n) + dn(r+1) + (1-d)(r+1)(q+1) + d(n-r)(q+1))L^{slot} \\ &\quad -(r(n^2+n) + dn(r+1) + (1-d)(r+1)q + d(n-r)q)L^{slot} \\ &= ((1-d)r + d(n-1-r) + 1)L^{slot} \end{aligned} \tag{4.46}$$

$$\begin{aligned} t^h_z &= (\Delta x + 1)L^{slot} \\ &= (u^{aa} - x + 1)L^{slot} \\ &\overset{(4.38)}{=} (x + (1-d)r + d(n-1-r) - x + 1)L^{slot} \\ &= ((1-d)r + d(n-1-r) + 1)L^{slot} \end{aligned} \tag{4.47}$$

Then, with Equations 4.46 and 4.47 we can state that

$$s_{z+1}^t - s_z^t = ((1-d)(r) + d(n-1-r) + 1)L^{slot} = t_z^h$$

Thus, the relation of Equation 4.45 holds.

Case 2 ($d = 0 \rightarrow d = 1$; r remains equal):
This case considers two time slots s_z^t and s_{z+1}^t that are within the same round but in different half rounds. Hence, the value of r remains equal for s_z^t and s_{z+1}^t and the value d must change from 0 to 1 as this is the only possible change for the parameter d when staying in the same round but changing the half round. Furthermore, the behaviour of d in this case causes the values of q to $q = n-1$ for s_z^t and to $q = 0$ for s_{z+1}^t. This parameter configuration results in the following equations about the terms $s_{z+1}^t - s_z^t$ and t^h.

$$\begin{aligned} s_{z+1}^t - s_z^t &= s_{ind_{aa}(r,1,0)}^t - s_{ind_{aa}(r,0,n-1)}^t \\ &\overset{(4.36)}{=} (r(n^2+n) + 1n(r+1) + (1-1)(r+1)0 + 1(n-r)0)L^{slot} \\ &\quad -(r(n^2+n) + 0n(r+1) + (1-0)(r+1)(n-1) + 0(n-r)(n-1))L^{slot} \\ &= (r+1)L^{slot} \end{aligned} \tag{4.48}$$

$$\begin{aligned} t_z^h &= (\Delta x + 1)L^{slot} \\ &= (u^{aa} - x + 1)L^{slot} \\ &\overset{(4.38)}{=} (x + (1-d)r + d(n-1-r) - x + 1)L^{slot} \\ &= (x + (1-0)r + 0(n-1-r) - x + 1)L^{slot} \\ &= (r+1)L^{slot} \end{aligned} \tag{4.49}$$

Then, with Equations 4.48 and 4.49 we can state that

$$s_{z+1}^t - s_z^t = (r+1)L^{slot} = t^h$$

Thus, the relation of Equation 4.45 holds.

Case 3 ($r \rightarrow r+1$):
This case considers the situation when the round changes for the time slots s_z^t and s_{z+1}^t. Hence, the value of r raises with one when going from s_z^t to s_{z+1}^t. The behaviour of r in this case causes the values of d to change from $d = 1$ for s_z^t to $d = 0$ for s_{z+1}^t. Furthermore, the value of q behaves similar as in Case 2. It exhibits the values $q = n-1$ for s_z^t and to $q = 0$ for s_{z+1}^t. Then, we see the following equations for the terms $s_{z+1}^t - s_z^t$ and t^h.

$$\begin{aligned} s_{z+1}^t - s_z^t &= s_{ind_{aa}(r+1,0,0)}^t - s_{ind_{aa}(r,1,n-1)}^t \\ &\overset{(4.36)}{=} ((r+1)(n^2+n) + 0n((r+1)+1) \\ &\qquad + (1-0)((r+1)+1)0 + 0(n-(r+1))0)L^{slot} \\ &\quad -(r(n^2+n) + 1n(r+1) + (1-1)(r+1)(n-1) + 1(n-r)(n-1))L^{slot} \\ &= (n-r)L^{slot} \end{aligned} \tag{4.50}$$

$$\begin{aligned} t_z^h &= (\Delta x + 1)L^{slot} \\ &= (u^{aa} - x + 1)L^{slot} \\ &\overset{(4.38)}{=} (x + (1-d)r + d(n-1-r) - x + 1)L^{slot} \\ &= (x + (1-1)r + 1(n-1-r) - x + 1)L^{slot} \\ &= (n-r)L^{slot} \end{aligned} \tag{4.51}$$

Then, with Equations 4.50 and 4.51 we can state that

$$s_{z+1}^t - s_z^t = (n-r)L^{slot} = t^h$$

Thus, the relation of Equation 4.45 holds.

Finally, as the equation $s_{z+1}^t - s_z^t = t^h$ holds for each case, Equation 4.45 holds for all possible combinations r, d and q. □

Lemma 4.9. *No path of the All-to-All schedule starts its vertical delivery before its direct predecessor starting at the same corner router has finished its vertical delivery.*

Proof. Equation 4.43 states that in each time slot $s_{ind_{aa}}^t$, and thus for each tuple (r, d, q), there begins one path for each node and all paths of a specific time slot share the same relative horizontal ($\Delta x = (u^{aa} - x) \bmod n$) and vertical ($\Delta y = (w^{aa} - y) \bmod n$) distances. Furthermore, according to the assumed hardware (see beginning of Section 4.1) the delivery in vertical direction takes $t^v = (\Delta y + 1)L^{hop}$. The time interval between the start in vertical direction of two successively executed paths is calculated by subtracting the corresponding times t_{z+1}^{vin} and t_z^{vin} from each other. Therefore, we need to show that the following equation holds for all possible parameter combinations (r, d, q).

$$t_{z+1}^{vin} - t_z^{vin} \leq t_z^v \tag{4.52}$$

According to Equation 4.41 each time t^{vin} of the All-to-All schedule is dependent on the terms $\lfloor \frac{z}{n} \rfloor$, $\left\lfloor \frac{z \bmod n}{2} \right\rfloor$ and $((z \bmod n) \bmod 2)$. Hence, we need to distinguish between three cases. Please note that according to Equation 4.37 we can state $z \bmod n = q$.

Case 1 ($((z \bmod n) \bmod 2) = 0 \wedge (((z+1) \bmod n) \bmod 2) = 1$):
In this case we consider two paths belonging to the same half round and pair of vertical delivery. Since the term $((z \bmod n) \bmod 2)$ indicates a specific path of such a pair, it has the value 0 for z and the value 1 for $z+1$. The assumption about being in the same pair causes the terms $\lfloor \frac{z}{n} \rfloor$ and $\left\lfloor \frac{z \bmod n}{2} \right\rfloor$ to remain equal for z and $z+1$ as they indicate the number of previously executed half rounds and pairs. Thus, we substitute these terms in a following way.

$$\left\lfloor \frac{z}{n} \right\rfloor = \left\lfloor \frac{z+1}{n} \right\rfloor = a \qquad \left\lfloor \frac{z \bmod n}{2} \right\rfloor = \left\lfloor \frac{(z+1) \bmod n}{2} \right\rfloor = b \tag{4.53}$$

$$
\begin{aligned}
t^{vin}_{z+1} - t^{vin}_{z} \overset{(4.41)}{=} & \left(n + \left\lfloor \frac{z+1}{n} \right\rfloor \frac{n(n+1)}{2} + \left\lfloor \frac{(z+1) \bmod n}{2} \right\rfloor (n+1) \right. \\
& \left. + (((z+1) \bmod n) \bmod 2) \left(n - \left\lfloor \frac{(z+1) \bmod n}{2} \right\rfloor \right) \right) L^{slot} \\
& - \left(n + \left\lfloor \frac{z}{n} \right\rfloor \frac{n(n+1)}{2} + \left\lfloor \frac{z \bmod n}{2} \right\rfloor (n+1) \right. \\
& \left. + ((z \bmod n) \bmod 2) \left(n - \left\lfloor \frac{z \bmod n}{2} \right\rfloor \right) \right) L^{slot} \\
\overset{(4.53)}{=} & \left(n + a\frac{n(n+1)}{2} + b(n+1) + 1\,(n-b) \right) L^{slot} \\
& - \left(n + a\frac{n(n+1)}{2} + b(n+1) + 0\,(n-b) \right) L^{slot} \\
= & \quad (n-b) L^{slot} \qquad (4.54)
\end{aligned}
$$

$$
\begin{aligned}
t^{v}_{z} &= (\Delta y + 1) L^{slot} \\
&= (w^{aa} - y + 1) L^{slot} \\
&\overset{(4.39)}{=} (y + (1 - (q \bmod 2)) \left(n - 1 - \left\lfloor \frac{q}{2} \right\rfloor \right) + (q \bmod 2) \left\lfloor \frac{q}{2} \right\rfloor - y + 1) L^{slot} \\
&\overset{(q = z \bmod n)}{=} \left(\left(n - 1 - \left\lfloor \frac{z \bmod n}{2} \right\rfloor \right) + 1 \right) L^{slot} \\
&\overset{(4.53)}{=} (n-b) L^{slot} \qquad (4.55)
\end{aligned}
$$

Then, with Equations 4.54 and 4.55 we can state

$$t^{vin}_{z+1} - t^{vin}_{z} = (n-b) L^{slot} = t^{v}_{z}$$

Thus, the relation of Equation 4.52 holds.

Case 2 ($\left\lfloor \frac{z \bmod n}{2} \right\rfloor = b \wedge \left\lfloor \frac{(z+1) \bmod n}{2} \right\rfloor = b + 1$):
In this case we consider two paths that are located in the same half round but in successive pairs of vertical delivery. Thus, $\left\lfloor \frac{z \bmod n}{2} \right\rfloor$ is set to b, while $\left\lfloor \frac{(z+1) \bmod n}{2} \right\rfloor$ is set to $b+1$. Since both paths belong to the same half round, it holds $\left\lfloor \frac{z}{n} \right\rfloor = \left\lfloor \frac{z+1}{n} \right\rfloor = a$. The behaviour of $\left\lfloor \frac{z \bmod n}{2} \right\rfloor$ in this case causes the term $((z \bmod n) \bmod 2)$ to the values of $((z \bmod n) \bmod 2) = (q \bmod 2) = 1$ and $(((z+1) \bmod n) \bmod 2) = ((q+1) \bmod 2) = 0$.

$$
\begin{aligned}
t^{vin}_{z+1} - t^{vin}_{z} \overset{(4.41)}{=} & \left(n + a\frac{n(n+1)}{2} + (b+1)(n+1) + 0\,(n-b+1) \right) L^{slot} \\
& - \left(n + a\frac{n(n+1)}{2} + b(n+1) + 1\,(n-b) \right) L^{slot} \\
= & \quad (b+1) L^{slot} \qquad (4.56)
\end{aligned}
$$

$$\begin{aligned}
t_z^v &= (\Delta y + 1) L^{slot} \\
&= (w^{aa} - y + 1) L^{slot} \\
&\overset{(4.39)}{=} \left(y + (1 - (q \bmod 2)) \left(n - 1 - \left\lfloor \frac{q}{2} \right\rfloor \right) + (q \bmod 2) \left\lfloor \frac{q}{2} \right\rfloor - y + 1 \right) L^{slot} \\
&= \left(\left\lfloor \frac{q}{2} \right\rfloor + 1 \right) L^{slot} \\
&\overset{(q = z \bmod n)}{=} \left(\left\lfloor \frac{z \bmod n}{2} \right\rfloor + 1 \right) L^{slot} \\
&\overset{(4.53)}{=} (b+1) L^{slot}
\end{aligned} \tag{4.57}$$

Then, with Equations 4.56 and 4.57 we can state

$$t_{z+1}^{vin} - t_z^{vin} = (b+1) L^{slot} = t^v$$

Thus, the relation of Equation 4.52 holds.

Case 3 ($\left\lfloor \frac{z}{n} \right\rfloor = a \wedge \left\lfloor \frac{z+1}{n} \right\rfloor = a + 1$):
In this case we consider two paths that are located in successive half rounds. Thus, the assumption about $\left\lfloor \frac{z}{n} \right\rfloor$ and $\left\lfloor \frac{z+1}{n} \right\rfloor$ in this case results in the fact that $z \bmod n = q = n - 1$ and $z + 1 \bmod n = q = 0$.

$$\begin{aligned}
t_{z+1}^{vin} - t_z^{vin} &\overset{(4.41)}{=} \left(n + (a+1) \frac{n(n+1)}{2} + \left\lfloor \frac{0}{2} \right\rfloor (n+1) + (0 \bmod 2) \left(n - \left\lfloor \frac{0}{2} \right\rfloor \right) \right) L^{slot} \\
&\quad - \Bigg(n + a \frac{n(n+1)}{2} + \left\lfloor \frac{n-1}{2} \right\rfloor (n+1) \\
&\qquad + ((n-1) \bmod 2) \left(n - \left\lfloor \frac{n-1}{2} \right\rfloor \right) \Bigg) L^{slot} \\
&= \left(\frac{n(n+1)}{2} - \left\lfloor \frac{n-1}{2} \right\rfloor (n+1) - ((n-1) \bmod 2) \left(n - \left\lfloor \frac{n-1}{2} \right\rfloor \right) \right) L^{slot}
\end{aligned} \tag{4.58}$$

$$\begin{aligned}
t_z^v &= (\Delta y + 1) L^{slot} \\
&= (w^{aa} - y + 1) L^{slot} \\
&\overset{(4.39)}{=} \left(y + (1 - (q \bmod 2)) \left(n - 1 - \left\lfloor \frac{q}{2} \right\rfloor \right) + (q \bmod 2) \left\lfloor \frac{q}{2} \right\rfloor - y + 1 \right) L^{slot} \\
&= \left((1 - ((n-1) \bmod 2)) \left(n - 1 - \left\lfloor \frac{n-1}{2} \right\rfloor \right) \right) L^{slot} \\
&\quad + \left(((n-1) \bmod 2) \left\lfloor \frac{n-1}{2} \right\rfloor + 1 \right) L^{slot}
\end{aligned} \tag{4.59}$$

According to Equations 4.58 and 4.59 there are two different cases possible depending if the dimension of the NoC n is even or odd. The first case (a) assumes n is even and thus, that $(n-1) \bmod 2 = 1$, while the second case (b) focuses an odd value n which causes $(n-1) \bmod 2 = 0$.

Case 3a) $((n-1) \bmod 2 = 1)$:
The assumed behaviour states that $n-1$ is an odd number. Thus, the following equation holds.

$$\left\lfloor \frac{n-1}{2} \right\rfloor = \frac{n-1}{2} - \frac{1}{2} = \frac{n-2}{2} \tag{4.60}$$

$$\begin{aligned} t_{z+1}^{vin} - t_z^{vin} &\overset{(4.58)}{=} \left(\frac{n(n+1)}{2} - \left\lfloor \frac{n-1}{2} \right\rfloor (n+1) - ((n-1) \bmod 2) \left(n - \left\lfloor \frac{n-1}{2} \right\rfloor \right) \right) L^{slot} \\ &\overset{(4.60)}{=} \left(\frac{n(n+1)}{2} - \frac{n-2}{2}(n+1) - 1 \left(n - \frac{n-2}{2} \right) \right) L^{slot} \\ &= \frac{n}{2} L^{slot} \end{aligned} \tag{4.61}$$

$$\begin{aligned} t_z^v &\overset{(4.59)}{=} \left((1 - ((n-1) \bmod 2)) \left(n - 1 - \left\lfloor \frac{n-1}{2} \right\rfloor \right) + ((n-1) \bmod 2) \left\lfloor \frac{n-1}{2} \right\rfloor + 1 \right) L^{slot} \\ &\overset{(4.60)}{=} \left((1-1) \left(n - 1 - \frac{n-2}{2} \right) + 1 \left(\frac{n-2}{2} \right) + 1 \right) L^{slot} \\ &= \frac{n}{2} L^{slot} \end{aligned} \tag{4.62}$$

Then, with Equations 4.61 and 4.62 we can state

$$t_{z+1}^{vin} - t_z^{vin} = \frac{n}{2} L^{slot} = t_z^v$$

Thus, the relation of Equation 4.52 holds.

Case 3b) $(n-1 \bmod 2 = 0)$:
The assumed behaviour implicitly states that $n-1$ is an even number. Thus, the following equation holds.

$$\left\lfloor \frac{n-1}{2} \right\rfloor = \frac{n-1}{2} \tag{4.63}$$

$$\begin{aligned} t_{z+1}^{vin} - t_z^{vin} &\overset{(4.58)}{=} \left(\frac{n(n+1)}{2} - \left\lfloor \frac{n-1}{2} \right\rfloor (n+1) - ((n-1) \bmod 2) \left(n - \left\lfloor \frac{n-1}{2} \right\rfloor \right) \right) L^{slot} \\ &\overset{(4.63)}{=} \left(\frac{n(n+1)}{2} - \frac{n-1}{2}(n+1) - 0 \left(n - \frac{n-1}{2} \right) \right) L^{slot} \\ &= \frac{n+1}{2} L^{slot} \end{aligned} \tag{4.64}$$

$$\begin{aligned} t_z^v &\overset{(4.59)}{=} \left((1 - ((n-1) \bmod 2)) \left(n - 1 - \left\lfloor \frac{n-1}{2} \right\rfloor \right) + ((n-1) \bmod 2) \left\lfloor \frac{n-1}{2} \right\rfloor + 1 \right) L^{slot} \\ &\overset{(4.63)}{=} \left((1-0) \left(n - 1 - \frac{n-1}{2} \right) + 0 \left(\frac{n-1}{2} \right) + 1 \right) L^{slot} \\ &= \frac{n+1}{2} L^{slot} \end{aligned} \tag{4.65}$$

Then, with Equations 4.64 and 4.65 we can state

$$t_{z+1}^{vin} - t_z^{vin} = \frac{n+1}{2} L^{slot} = t^v$$

Thus, the relation of Equation 4.52 holds.

Finally, as the equation $s_{z+1}^t - s_z^t = t^h$ holds for each case, Equation 4.52 holds for all possible combinations r, d and q. □

Lemma 4.10. *The horizontal delivery of an All-to-All TDM cycle takes no more time than the duration of an All-to-All period T_{aa}.*

Proof. According to the proof of Lemma 4.8 the time between two starts of paths s_z^t and s_{z+1}^t is exactly the duration of the delivery of the first of both paths t_z^h as equation $s_{z+1}^t - s_z^t = t^h$ holds. Thus, each path starts directly after the completion of the previous one and the complete execution time of all paths of a TDM cycle can be calculated with a global sum over all durations.

$$\sum_{\substack{r\in\left[0;\frac{n}{2}\right]\\ d\in\{0;1\}\\ q\in[0;n-1]}} t^h_{z(r,d,q)} = \sum_{\substack{r\in\left[0;\frac{n}{2}\right]\\ d\in\{0;1\}\\ q\in[0;n-1]}} (\Delta x_{z(r,d,q)} + 1)L^{slot}$$

For all distances Δx_z there are n paths, as there are n nodes per column of the NoC. Furthermore, there are n different distances to serve ($\Delta x_z \in [0, n-1]$). According to this argumentation we can transform the sum over all possible z to a sum over all possible Δx_z.

$$\sum_{\substack{r\in\left[0;\frac{n}{2}\right]\\ d\in\{0;1\}\\ q\in[0;n-1]}} (\Delta x_{z(r,d,q)} + 1)L^{slot} = n\sum_{\Delta x_z=0}^{n-1} (\Delta x_z + 1)L^{slot} = n\frac{n(n+1)}{2}L^{slot}$$

Thus, we can state the following equation.

$$\sum_z t^h = \frac{n^2(n+1)}{2}L^{slot} = T_{aa} \tag{4.66}$$

Hence, the Lemma holds, as the overall duration of the horizontal delivery of a TDM cycle is the sum over all durations t^h. □

Lemma 4.11. *The vertical delivery of an All-to-All TDM cycle takes no more time than the duration of an All-to-All period T_{aa}.*

Proof. According to the proof of Lemma 4.9 the time between two starts of paths' vertical deliveries t_z^{vin} and t_{z+1}^{vin} is exactly the duration of the delivery of the first of both paths t_z^v as equation $t_{z+1}^{vin} - t_z^{vin} = t_z^v$ holds. Thus, each paths starts directly after the completion of the previous one and the complete execution time of all paths of a TDM cycle can be calculated with a global sum over all durations.

$$\sum_{\substack{r\in\left[0;\frac{n}{2}\right]\\ d\in\{0;1\}\\ q\in[0;n-1]}} t^v_{z(r,d,q)} = \sum_{\substack{r\in\left[0;\frac{n}{2}\right]\\ d\in\{0;1\}\\ q\in[0;n-1]}} (\Delta y_{z(r,d,q)} + 1)L^{slot}$$

For all distances Δy_z there are n paths, as there are n nodes per row of the NoC. Furthermore, there are n different distances to serve ($\Delta y_z \in [0, n-1]$). According to this argumentation

we can transform the sum over all z to a sum over all possible Δy_z.

$$\sum_{\substack{r\in\left[0;\frac{n}{2}\right]\\ d\in\{0;1\}\\ q\in[0;n-1]}} (\Delta y_{z(r,d,q)}+1)L^{slot} = n\sum_{\Delta y_z=0}^{n-1}(\Delta y_z+1)L^{slot} = n\frac{n(n+1)}{2}L^{slot}$$

Thus, we can state the following equation.

$$\sum_z t^v = \frac{n^2(n+1)}{2}L^{slot} = T_{aa} \tag{4.67}$$

Hence, the Lemma holds, as the overall duration of the vertical delivery of a TDM cycle is the sum over all durations t^v. □

Theorem 4.4. *The All-to-All TDM schedule is conflict-free according to Definition 2.25.*

Proof. In each TDM cycle the All-to-All schedule performs concurrently horizontal and vertical delivery. Thus, the proof about conflict-freedom of the schedule is split into three parts. These are the conflict-freedom of (1) the horizontal and (2) the vertical phases and finally the last part (3) examines the concurrent execution of deliveries to both directions.

Equation 4.43 states that in each time slot $s^t_{ind_{aa}}$ with an index ind_{aa}, and thus for each tuple (r,d,q), there begins one path for each node and all paths of a specific time slot share the same relative horizontal ($\Delta x = (u^{aa}-x) \bmod n$) and vertical ($\Delta y = (w^{aa}-y) \bmod n$) distance. Since all these paths with the same distances release their flit to the NoC (either horizontally or vertically) at the same point in time and start from different nodes or corner routers, Lemma 4.1 ensures a conflict-free delivery of all those paths along the according direction. Thus, it remains to show for both phases (horizontal and vertical) that each path begins not before the previous path of the same source node has completely finished its horizontal or vertical delivery, respectively. Furthermore, all delivery either horizontal or vertical of a TDM cycle must be finished when the corresponding (horizontal or vertical) delivery of the next TDM cycle starts.

Part 1 (horizontal phase):
Lemma 4.8 states that no paths of the All-to-All schedule starts before the previous paths starting at the same node has finished its horizontal delivery. Hence, it is not possible that paths of the same TDM cycle which start at the same node can conflict each other. Furthermore, Lemma 4.10 states that the complete horizontal delivery takes no more than the TDM cycle's period and the first paths of a current TDM cycle ($r=0, d=0, q=0$) start at time slot $s^t_{ind_{aa}(0,0,0)} = s^t_0$ (see Equation 4.36). Since the horizontal delivery starts directly at the beginning of the TDM cycle and its duration is smaller or equal to the schedule's period T_{aa}, the horizontal delivery finishes always within the same TDM cycle as it starts. Hence, no conflicts can occur between horizontal deliveries of different TDM cycles.

Part 2 (vertical phase):
Lemma 4.9 states that no paths of the All-to-All schedule starts its vertical delivery before the previous paths starting at the same corner router has finished its vertical delivery. Hence, it is not possible that paths of the same TDM cycle which start at the same corner router

can conflict each other. Furthermore, Lemma 4.11 states that the complete vertical delivery takes no more than the TDM cycle's period and the first paths of a current TDM cycle ($r = 0$, $d = 0$, $q = 0$) start their vertical delivery at time $t^{vin}_{z=0} = nL^{slot}$ (see Equation 4.41). Since each vertical delivery starts at the same point in time within a TDM cycle and its duration is smaller or equal to the schedule's period T_{aa} each vertical delivery overlaps its execution with its successively performed TDM cycle with the same period of time that is smaller or equal to the starting time $t^{vin}_{z=0}$. Hence, no conflicts can occur between vertical deliveries of different TDM cycles.

Part 3 (concurrent horizontal and vertical phase):
Corollary 4.1 directly implies that the concurrent execution of one horizontal and one vertical phase of the All-to-All schedule is conflict-free. □

4.1.2 Composition of General-Purpose TDM Schedules

There are several different patterns of communication that are performed in an application (see Section 2.3). These communications are basically build on the situations where one sender sends data to N receivers (1:N) or N sender deliver data to one receiver (N:1). Please note that the communication where one sender communicates to one receiver is a special case of the mentioned situations.

Looking at the timing of the general-purpose schedules, there is no schedule that is optimal or near the optimum in terms of bandwidth for both possible communication directions (1:N and N:1) and for all group sizes. Thereby, the bandwidth is considered optimal when the admission time t^a is low ($t^a \rightarrow 1 \cdot L^{slot}$). Though, the One-to-All schedule is near the optimum for communicating data in N:1 fashion in large groups, there is a high admission time for the 1:N communication and for small group sizes. In contrast, All-to-One behaves exactly vice versa in a way that it is nearly optimal for 1:N communication in large groups, but suffers in bandwidth for N:1 communication and for small group sizes. Although, the All-to-All and One-to-One schedules behave equal for both directions, they do not get near the optimal t^a, except for the One-to-One schedule when using only small groups. In the case of small groups the One-to-One schedule exhibits a near optimal bandwidth for both communication directions. The All-to-All schedule is works only well for the concurrent execution of both communication directions (1:N and N:1) with large group sizes. Subsequently, we refer to such an communication as all-to-all or N:N communication.

Application specific schedules only consider the connections between nodes that are truly communicating to each other in the regarded application. Thus, it provides a highly optimized period and average admission times that are developed according to the individual communication needs of each node. Thereby, connections that have high bandwidth demands obtain a low average admission time while connections that require only low bandwidth are provided with larger admission times. Although, application specific schedules show advantages in performance, they suffer in terms of composability and adaptability due to being dependent on task placement and an application specific communication considered at design time (see Section 2.2.1). Due to the described lack of performance for general-purpose schedules in various situations and the lack of composability for ap-

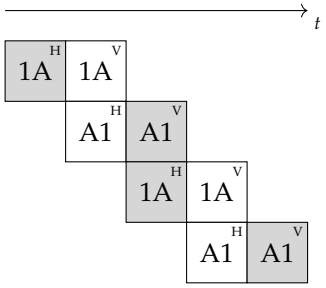


Figure 4.9: Overview of overlapped execution of the Alternate schedule. Each white or grey column marks one round, while each row indicates a set of paths that contains the paths starting in the same round. Typically, the paths start in one round with horizontal delivery and perform their vertical delivery in the subsequent round. The phases of horizontal data transfer and vertical data transfer are marked with the symbols H and V. The mapping of the paths and rounds to the applied basic schedules are indicated by the symbols 1A (One-to-All) and A1 (All-to-One).

plication specific schedules, there is a need for providing schedules that exhibit a better trade-off between performance and composability. Hence, schedules are in demand that on the one hand show the advantages of a general-purpose schedule and on the other hand concurrently offer a near optimal admission time for sending and receiving data in 1:N and in N:1 fashion.

In this section we propose two different TDM schedules that target low admission times for both communication directions (1:N and N:1) at the same time. All subsequently presented schedules base on the already existing state-of-the-art schedules described in Sections 2.2.2 and 4.1.1. The new schedules alternate between different basic schedules or utilize free NoC capacity that remains available when using one of the schedules to concurrently provide an additional schedule. With this technique we are able to mask a schedule's behaviour with the behaviour of another schedule in situations when the first schedule shows poor performance characteristics. At first, we combine the One-to-All and All-to-One schedules to obtain an acceptable performance for both communication directions 1:N and N:1. Afterwards, we extend that schedule and provide an alternative optimization which combines the three schedules One-to-All, All-to-One and One-to-One in order to increase the performance of 1:N and N:1 communications with small group sizes.

Alternate Schedule

The *Alt (Alternate)* TDM schedule targets to combine the advantages of the One-to-All and All-to-One schedules to reveal a high bandwidth for both communication directions 1:N and N:1. Both basic schedules exhibit an equal number of rounds, the rounds are of an equal size and both schedules divide their paths in the two phases of a horizontal and a vertical delivery. Hence, they can easily be combined by alternating their execution. We use the same execution scheme as depicted in Figure 4.2 with the overlapping of horizontal and vertical phases and for each round we alternate the execution of the basic schedules. An overview of the execution scheme of the Alternate schedule is illustrated in Figure 4.9.

This schedule completely contains both schedules One-to-All and All-to-One. Thereby, each node is allowed to take place in one of the schedules within a given round. However, each node can change its participation in a basic schedule at the border between two rounds. Hence, according to the intended communication (1:N or N:1) the basic schedule which exhibits the higher bandwidth for the current case can be used. The NI must know which basic schedule it has to apply for a given flit, for the correct usage of this schedule. Thus, a suitable mechanism is needed in the sender nodes. One possibility could be to send a flit with the All-to-One part when the NI is told to deliver the flit to more than one target and to utilize the One-to-All part otherwise. Another way would be to tell the NI the desired behaviour by setting a flag in a control register for each flit.

Though this schedule exploits the advantages of both basic schedules, there are some trade-offs. As the One-to-All and All-to-One schedules are combined by alternating their execution, one TDM cycle must include all rounds of both schedules. Therefore, the number of rounds of the basic schedules must be summed up for the Alternate schedule, which causes a total number of $n + n = 2n$ rounds. The increased number of rounds leads to a larger period T_{alt} for the new schedule tdm^S_{alt}.

$$T_{alt} = T_{1a} + T_{a1} = n^2 \cdot L^{hop} + n^2 \cdot L^{hop} = 2n^2 \cdot L^{hop} \tag{4.68}$$

Similar to the period the sets of paths of the basic schedules are combined as well. Thus, the number of paths for the Alternate TDM schedules is as follows.

$$|P^{tdm}_{alt}| = |P^{tdm}_{1a}| + |P^{tdm}_{1a}| = 2(n^4 - n^2) \tag{4.69}$$

Within a round the paths are arranged in the same way as for the basic schedules. Therefore, all paths can be grouped according to the time slot when they begin their delivery. Due to the similarity to the One-to-All and All-to-One schedules, the sets are defined equally to Equations 4.15 and 4.22.

$$P^{s,alt}_{2r \cdot n} = P^{s,1a}_{r \cdot n} \tag{4.70}$$

$$P^{s,alt}_{(2r+1) \cdot n+q} = P^{s,a1}_{r \cdot n+q} \tag{4.71}$$

In Equations 4.70 and 4.71 n is the number of time slots contained in one round, $q \in [0, n-1]$ is an index that refers to a specific time slot in a round and $r \in [0, n-1]$ is used to indicate the desired round. Thereby, $2r$ refers to rounds with an even index, while $2r+1$ indicates odd rounds. The complete set of paths of the Alternate schedule is stated as follows.

$$P^{tdm}_{alt} = \bigcup_{r,q \in [0,n-1]} (P^{s,alt}_{2r \cdot n} \cup P^{s,alt}_{(2r+1) \cdot n+q}) \tag{4.72}$$

In addition to the paths we also reuse the constraints of both basic schedules. There are only small adaptions needed, as each constraint is only relevant for the corresponding parts of its original schedule.

$$\begin{aligned} &\forall i \in \mathbb{N} : \forall \vec{p}_l \in P^{s,alt}_{2r \cdot n} : \nexists \vec{p}_m \in P^{s,alt}_{2r \cdot n} : \\ &\quad \vec{p}_l.h_{first}.f \neq \emptyset \wedge \vec{p}_m.h_{first}.f \neq \emptyset \wedge \vec{r}_{\vec{p}_l}.ph^{src}_{first} = \vec{r}_{\vec{p}_m}.ph^{src}_{first} \end{aligned} \tag{4.73}$$

$$\begin{aligned} &\forall i \in \mathbb{N} : \forall \vec{p}_l \in P^{s,alt}_{(2r+1) \cdot n+q} : \nexists \vec{p}_m \in P^{s,alt}_{(2r+1) \cdot n+q} : \\ &\quad \vec{p}_l.h_{last}.f \neq \emptyset \wedge \vec{p}_m.h_{last}.f \neq \emptyset \wedge \vec{r}_{\vec{p}_l}.ph^{dst}_{last} = \vec{r}_{\vec{p}_m}.ph^{dst}_{last} \end{aligned} \tag{4.74}$$

As Equations 4.73 and 4.74 are the only constraints for the Alternate schedule, the size of this set is two ($|tdm^{S}_{alt}.Co| = 2$). We do not display the shape of the path for the example with a 3×3 NoC here, as all paths of each round can easily be seen in Figures 4.4 and 4.5.

Theorem 4.5. *The Alternate TDM schedule is relaxed conflict-free according to Definition 2.26.*

Proof. Theorems 4.1 and 4.2 show that both basic schedules are relaxed conflict-free. Thus, it remains to show that the overlapping of the basic schedules is relaxed conflict-free.

In Lemma 4.2 it is shown that both delivery phases (horizontal and vertical) of the One-to-All schedule finish within the same round of their beginning. Furthermore, Lemma 4.3 shows the same characteristic for the horizontal All-to-One delivery and Lemma 4.5 states the finishing in the same round for the All-to-One schedule's vertical delivery. Additionally, Equation 4.70 states that each path of the Alternate schedule's One-to-All part begins in a round $2r$ and Equation 4.71 shows that each path of the All-to-One part starts in a round $2r + 1$. Thus, the horizontal rounds of the One-to-All part and the All-to-One part never start in the same round, but they alternate between each other. As all horizontal and vertical deliveries of both basic schedules take one round and all vertical deliveries start in the round that is successively following the horizontal delivery, also the vertical phases of both schedules' parts are alternating. Thus, the horizontal phase of the One-to-All schedule only overlaps with the All-to-One schedule's vertical phase and the One-to-All schedule's vertical phase only overlaps with the horizontal phase of the All-to-One schedule. No other parts of the schedule overlap each other.

That implies that there is only an overlapping between a horizontal and a vertical phase, but no overlapping between two horizontal or two vertical phases. Then, we can apply Corollary 4.1 to prove that those overlappings are conflict-free. Thus, the complete Alternate schedule is relaxed conflict-free. □

Triplet Schedule

Even though the Alternate schedule provides acceptable admission times for both kinds of communications (1:N and N:1), it still suffers in performance when communicating in small groups. To overcome this lack in performance a schedule is desired that behaves similar to the One-to-One schedule for small groups, but exhibits the performance of the Alternate schedule for large group sizes. Such a behaviour is achieved by providing a suitable combination of both schedules. We call this schedule the *Tri (Triplet)* schedule as it in fact combines the three basic schedules One-to-All, All-to-One and One-to-One. Similar to the Alternate schedule each node is allowed to take part in only one of the basic schedules at the same time and it can change the currently applied basic schedule after each round of the actual schedule. Its construction is presented hereafter.

The Triplet schedule uses the Alternate schedule as basis and utilizes network capacities that remain free to execute the paths of the One-to-One schedule. Thereby, it uses the facts that the procedure for the horizontal delivery is exactly the same for the One-to-All and the One-to-One schedule and that each node is allowed to only take part in one of the three basic schedules at the same time. Thus, for a node that participates in the One-to-One schedule the slots of the corresponding One-to-All schedule's path starting at that node are not occupied. Consequently, there is a free horizontal path to any possible corner router in the horizontal

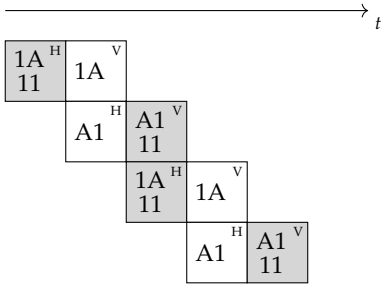


Figure 4.10: Overview of overlapped execution of the Triplet schedule. Each white or grey column marks one round, while each row indicates a set of paths of the Alternate schedule that contains the paths starting in the same round. Typically, the paths start in one round with horizontal delivery and perform their vertical delivery in the subsequent round. In the One-to-One part a pair of two rows marks the set of paths that start in the same round as these paths obtain a waiting time of one round in the corner routers. The phases of horizontal data transfer and vertical data transfer are marked with the symbols H and V. The mapping of the paths and rounds to the applied basic schedules are indicated by the symbols 1A (One-to-All), A1 (All-to-One) and 11 (One-to-One).

phase of the One-to-All part that can be used for the One-to-One part. Furthermore, the vertical delivery of the All-to-One basic schedule occupies a relatively small part of the network in each round. Additionally, due to the restriction that each node is allowed to only take part in one of the basic schedules, the All-to-One paths of nodes which take part in the One-to-One schedule are not occupied in that round. Moreover, the All-to-One and the One-to-One schedules share the same constraint about receiving at most one flit per node and TDM cycle. Altogether, this indicates the possibility to combine the vertical phases of the All-to-One and the One-to-One parts of the Triplet schedule.

As there are a lot more paths to serve for the vertical delivery in one round for the One-to-One schedule than for the All-to-One schedule, we take the procedure of the One-to-One schedule's vertical delivery as basis. In that procedure each flit is released to the vertical delivery in a way that they all reach their target nodes at the end of the current round. This is possible since each node receives at most one flit per round. The restriction about receiving at most one flit per round holds true for the All-to-One schedule, too. Since each node is only participating in one of the basic schedules, the One-to-One paths targeting the nodes that participate in the All-to-One schedule remain free. Thus, these paths can be used to deliver the corresponding All-to-One schedule's flit to that node. Please note that the horizontal delivery of the One-to-One schedule is during the horizontal delivery of the One-to-All basic schedule, while the One-to-One schedule's vertical delivery is performed concurrent to the All-to-One schedule. Hence, this procedure forces each flit of the One-to-One schedule to wait one complete round in the corner router before it starts the vertical delivery. An overview of the Triplet schedule's execution scheme is illustrated in Figure 4.10.

The schedule is applied in the same way as the Alternate schedule for large groups with a size larger or equal to n nodes and for small groups sizes with less than n participants the One-to-One part is applied. Similar to the Alternate schedule in this schedule the sender

nodes need to know which basic schedule must be used. Hence, the same mechanisms are needed as in Alternate schedule (e.g. using the All-to-One part for delivering data to multiple target nodes and utilizing One-to-All otherwise). Moreover, also the corner routers must be aware of which flit shall be forwarded directly in the next round for the Alternate part or must be stored for an additional round to deliver it in the One-to-One part. As there is no possibility to decide this when taking into account the point in time when the flit is reaching the corner router, the flits must carry this information with them. Thus, either one bit of each flit must be reserved to carry this scheduling information or there must be one additional hardware wire for each physical link in the NoC that is used to transport this information. The additional hardware costs consisting of the additional wire and the larger buffer size required in each router to store the One-to-One flits for an extra round are a disadvantage of this schedule compared to the other general-purpose schedules.

In contrast to all previously presented schedules the Triplet schedule exhibits two different periods $T_{Tri.11}$ and $T_{Tri.Alt}$. Thereby, $T_{Tri.Alt}$ is the same as for the Alternate schedule and holds for all nodes that apply either the One-to-All or the All-to-One part of the schedule.

$$T_{Tri.Alt} = T_{Alt} = 2n^2 \cdot L^{hop} \tag{4.75}$$

On the other hand, when the schedule's One-to-One part is applied, the period is oriented to the one of the One-to-One schedule, which is significantly shorter as it consists of only one round. Nevertheless, since the Triplet's One-to-One part begins during the round for the horizontal One-to-All part and ends not before the end of the All-to-One part's vertical delivery, the TDM cycle of the One-to-One part is twice as long as in the original One-to-One schedule. Thus, the period of the Triplet schedule's One-to-One part is given as follows.

$$T_{Tri.11} = 2 \cdot T_{11} = 2n \cdot L^{hop} \tag{4.76}$$

As the lengths of the rounds of the Alternate part and the One-to-One part differ likewise, we subsequently refer to an Alternate round for a round with a length of nL^{slot} of the Alternate part and to an One-to-One round for a round with a length of $2nL^{slot}$ of the One-to-One part.

The set of paths for the Triplet schedule basically consists of the paths of the Alternate and the One-to-One schedule. However, due to the combination of both sets and the resulting adaptations the definitions of the paths change. The One-to-All and One-to-One schedule perform exactly the same horizontal delivery in each round for all paths (similar to Figure 4.3a) and the horizontal part of the All-to-One part remains unchanged when constructing the Triplet schedule. Hence, there is no adaptation needed in the definitions of the paths for the horizontal deliveries. For that reason the changes concentrate on the vertical delivery and the buffering times of the All-to-One and the One-to-One parts of the schedule. Thus, we take the path definition of the basic schedules as basis and adapt them in an appropriate way to fit for the Triplet schedule.

As the paths of the One-to-All part do not need any adaptation, we reuse their definition from Equation 4.15 without changes in the same way as for the Alternate schedule.

$$P^{s,Tri.1a}_{2r \cdot n} = P^{s,1a}_{r \cdot n} \tag{4.77}$$

Like described above the paths of the One-to-One schedule (see Equation 4.30) also remain unchanged except for a change in the times a flit lasts in the corner routers. This adaptation

is caused by the fact that each flit must wait one additional Alternate round of nL^{slot} until it is forwarded in vertical direction. Hence, the definition of the paths for the One-to-One part are given as follows.

$$\begin{aligned} P^{s,Tri.11}_{2r\cdot n} = \{\vec{p} \mid\ & \vec{p} \in P^{tdm}, \exists x,y,u,w \in [0,n-1] : \\ & (\ \vec{r}_{\vec{p}}.n^{src} = PE(x,y) \wedge \vec{r}_{\vec{p}}.n^{dst} = PE(u,w)) \\ \wedge\ & (\ \exists h_{cin}, h_{cout} \in \vec{p} : h_{cin}.ph^{dst} = h_{cout}.ph^{src} = PE(x,w) \\ & \wedge \vec{p}.h_{cin}.t + L^{hop}_{h_{cin}.ph} + nL^{slot} \\ & \quad +(1+2n-(2+\Delta x+\Delta y))\cdot L^{slot} = \vec{p}.h_{cout}.t)\} \end{aligned} \tag{4.78}$$

In this equation the term nL^{slot} is added to the last line compared to the original Equation 4.30. This term causes the additional waiting in the corner router of one Alternate round. Furthermore, the vertical parts of the paths of the All-to-One schedule must fit to the paths of the One-to-One schedule, which provide a vertical delivery similar to the one of Figure 4.3b. Hence, the definition of those paths from Equation 4.22 must be adapted accordingly.

$$\begin{aligned} P^{s,Tri.a1}_{(2r+1)\cdot n+q} = \{\vec{p} \mid\ & \vec{p} \in P^{tdm}, \exists x,y \in [0,n-1] : \vec{r}_{\vec{p}}.n^{src} = PE(x,y) \\ & \wedge \vec{r}_{\vec{p}}.n^{dst} = PE(x+n-1-q \bmod n, y+n-1-r \bmod n) \\ & \wedge (\exists h_{cin}, h_{cout} \in \vec{p} : (h_{cin}.ph^{dst} = h_{cout}.ph^{src} = PE(x,w)) \\ & \quad \wedge (\vec{p}.h_{cin}.t + L^{hop}_{h_{cin}.ph} + (1+n-(w-y))\cdot L^{slot} = \vec{p}.h_{cout}.t))\} \end{aligned} \tag{4.79}$$

In this equation the term $(n-(w-y))L^{slot}$ is added in the last line to the term $1 \cdot L^{slot}$ of the original equation. Thereby, the term $w-y$ denotes the vertical distance Δy from the source to the target node. This adaptation forces each flit to start the vertical transport in a way that it reaches its destination node exactly at the end of the current round and thus to follow the same strategy as the One-to-One part of the Triplet schedule.

Similar to the Alternate schedule the points in time when the paths start their delivery must be adapted as well to fit for the Triplet schedule. Since the Triplet schedule bases on the Alternate schedule, the starting points for the One-to-All and the All-to-One parts are the same as in Equations 4.70 and 4.71 and the beginning of the One-to-One paths is set to every even Alternate round of the Triplet schedule. In Equations 4.77, 4.78 and 4.79 n is the number of time slots contained in one Alternate round, $q \in [0, n-1]$ is an index that refers to a specific time slot in an Alternate round and $r \in [0, n-1]$ is used to indicate the desired round. Thereby, $2r$ refers to Alternate rounds with an even index, while $2r+1$ indicates odd rounds.

With the previously described paths it is possible to state the complete set of paths of the Triplet schedule.

$$P^{tdm}_{Tri} = \bigcup_{r,q \in [0,n-1]} (P^{s,Tri.1a}_{2r\cdot n} \cup P^{s,Tri.a1}_{(2r+1)\cdot n+q} \cup P^{s,Tri.11}_{2r\cdot n}) \tag{4.80}$$

The constraints of the Triplet schedule are very similar to the ones of its basic schedules, as they need to ensure the same characteristics to the respective schedule parts as for the

basic schedules.

$$\forall i \in \mathbb{N} : \forall \vec{p}_l \in P^{s,Tri.1a}_{2r \cdot n} \cup P^{s,Tri.11}_{2r \cdot n} : \nexists \vec{p}_m \in P^{s,Tri.1a}_{2r \cdot n} \cup P^{s,Tri.11}_{2r \cdot n} : \\ \vec{p}_l.h_{first}.f \neq \emptyset \wedge \vec{p}_m.h_{first}.f \neq \emptyset \wedge \vec{r}_{\vec{p}_l}.ph^{src}_{first} = \vec{r}_{\vec{p}_m}.ph^{src}_{first} \tag{4.81}$$

$$\forall i \in \mathbb{N} : \forall \vec{p}_l \in P^{s,Tri.a1}_{(2r+1) \cdot n+q} \cup P^{s,Tri.11}_{2r \cdot n} : \nexists \vec{p}_m \in P^{s,Tri.a1}_{(2r+1) \cdot n+q} \cup P^{s,Tri.11}_{2r \cdot n} : \\ \vec{p}_l.h_{last}.f \neq \emptyset \wedge \vec{p}_m.h_{last}.f \neq \emptyset \wedge \vec{r}_{\vec{p}_l}.ph^{dst}_{last} = \vec{r}_{\vec{p}_m}.ph^{dst}_{last} \tag{4.82}$$

Please note that Equation 4.81 includes the constraint for sending only one flit per TDM cycle in the One-to-All part as well as in the One-to-One part and Equation 4.82 is the constraint about receiving only one flit per TDM cycle for the All-to-One and the One-to-One part. Thus, no more than these two constraints are needed for the set of constraints of the Triplet schedule and the size of this set is 2 ($|tdm^S_{Tri}.Co| = 2$). Similar to the Alternate schedule we do not display the example for the 3×3 NoC, as it is very similar to the examples of the previously presented schedules.

In the remainder of this section we show that the Triplet schedule is relaxed conflict-free.

Theorem 4.6. *The Triplet TDM schedule is relaxed conflict-free according to Definition 2.26.*

Proof. The Triplet schedule is composed of the Alternate and the One-to-One schedule with some adaptations. As the Alternate schedule (see Theorem 4.5) and the One-to-One schedule (see Theorem 4.3) are relaxed conflict-free, it remains to show that the applied adaptations and the composition of both schedules do not violate the relaxed conflict-freedom. Thus, at first we focus on the applied adaptation and consider the composition of the TDM schedules afterwards.

Conflict-freedom of Adaptations:
There are two adaptations compared to the original schedules. The first one is the equal increase in waiting time in the corner routers for all paths of the One-to-One schedule and the second adaptation is the change of the vertical delivery in the All-to-One part of the schedule.
a) Adaptation in One-to-One part:
In the One-to-One part the vertical delivery is delayed for a duration of nL^{hop} cycles for each path of the One-to-One part. Due to the equality of the adaptation for all paths, that change does not effect the arrangement of the paths relative to each other in the vertical delivery among these paths and thus, the characteristic about relaxed conflict-freedom of the vertical delivery in the One-to-One schedule holds for this adaptation. Furthermore, as all paths are equally delayed no matter in which TDM cycle they start, no additional overlapping is introduced between the paths of different One-to-One TDM cycles and hence there is no additional overlapping between different TDM cycles that can cause conflicts.
b) Adaptation in All-to-One part:
The vertical delivery of the All-to-One part is changed to the procedure of the vertical delivery of the One-to-One schedule. The same assumptions hold for the All-to-One schedule as for the One-to-One schedule at the beginning of the vertical transportation phase. These assumptions are the restriction that there are no more than n flits in each column of the NoC.

Thereby, the flits can be arbitrarily distributed among the corner routers of the regarded NoC column. Moreover, both schedules share the same constraint about receiving flits in one TDM cycle (see Equations 4.24 and 4.31). Thus, the proof about relaxed conflict-freedom for the adapted vertical All-to-One delivery is the same as the one in the proof of Theorem 4.3 for the One-to-One schedule.

Conflict-freedom of Composition:
To proof the absence of conflicts when considering the mentioned composition, we at first focus on the concurrent execution of the horizontal One-to-All and One-to-One parts and then regard to the concurrent execution of the vertical All-to-One and One-to-One parts.
a) Concurrent horizontal delivery:
We consider the concurrent use of the One-to-All and the One-to-One parts of the Triplet schedule. Each node of the NoC takes part in only one schedule for a given round or TDM cycle. Thus, it either belongs to the One-to-All or the One-to-One part but never to both parts at the same time. Then, the paths reserved for a node in a particular schedule remain free for the regarded round or TDM cycles when the node takes part in the other schedule. Furthermore, the One-to-All part delivers all flits starting at a given round to the same designated horizontal distance and each sender releases the flits in the first time slot of the round. Thus, all paths reach their corner router in the time slot $s^t = rn + \Delta x + 1$, which is always within the same round as the begin of the traversal. Then, there is no traffic in horizontal rings during the period which starts at the time slot when the flits arrive at the corner routers and ends at the end of the round.

In a given round the points in time when the paths start are equal for the One-to-All and the One-to-One part and both apply a subset of paths from Equation 4.10. Thus, the occupied physical links and slots in the horizontal delivery are the same for both schedules except for unused links that occur for the various horizontal distances that cause an early arrival at the corner routers when Δx is smaller than n. Thus, in a round r it is possible to replace a path of a sender node n_{src} of one schedule with a path of the other schedule that begins in the same round r and at the same sender node n_{src}. This replacement does not cause any additional overlapping with existing paths due to the similarity of the regarded paths. Thus, it is not possible to produce conflicts when executing the One-to-All and One-to-One horizontal delivery concurrently, as every node takes part in only one of them in the considered round.
b) Concurrent vertical delivery:
We consider the concurrent use of the All-to-One and One-to-One parts of the schedule. Each node of the NoC takes part in only one schedule at the same time. Thus, it either applies the All-to-One or the One-to-One part but never both parts in the same round. Then the paths reserved for a node in a particular schedule remain free for the regarded round or TDM cycle when the node takes part in the other schedule. Furthermore, the All-to-One part delivers in a given round only flits to one vertical distance and the corresponding corner routers release the flits in the same way for both schedules which is at time slot $s^t = n - \Delta y - 1$ of the regarded round. As both schedules obtain the same constraint about receiving only one flit per TDM cycle (see Condition 4.82) and each node takes part in only one schedule there is at maximum one flit received at each node at the end of the round. As the procedure for starting and delivering is equal for both schedules and both schedules

meet the same constraints, a vertical delivery of a path of the All-to-One schedule can be interchanged with one of the One-to-One schedule without producing overlapping. Thus, no conflicts can occur when they are executed concurrently. □

4.2 Formal Timing Analysis

In this section we provide details about obtaining the worst-case bandwidth and latency of a message consisting of multiple flits. The concrete calculations in this section base on the work of Mische and Ungerer [MU14] and on our own previous work [Ste+16] and are extended to suit for application specific as well as for general-purpose TDM schedules. Firstly, the general approach is presented, that can be applied to any hardware and TDMA mechanism that is conform with our system model from Chapter 2. Afterwards, we specialize this approach to the NoC and schedules assumed and presented in Section 4.1.

4.2.1 General Approach

The formal timing analysis mainly concentrates on the determination of the admission (see Definition 2.32) and transportation times (see Definition 2.35) and the points in time when flits are released to the NoC as they are the building blocks for bandwidth and latency calculations. Thereby, the release times are directly influenced by the basic admission times. At first, we present the timing characteristics of the transportation time before we switch the focus to the admission time.

As the transportation time t^t denotes the time a flit needs to traverse the network and the applied TDM schedule ensures the absence of conflicts with other flits and unintended waiting times, the transportation time from a node n_i to a node n_j is given by the actual latency of the path used. Thus, it is calculated as follows:

$$t^t_{i,j} = L^{path}_{\vec{p}_{i,j}} \tag{4.83}$$

As stated in Equation 2.13 the latency $L^{path}_{\vec{p}_{i,j}}$ remains equal for all paths $\vec{p}_{i,j}$ that share the same route $\vec{r}_{\vec{p}_{i,j}}$.

Determining the concrete admission times t^a for each flit that can be sent over the NoC is difficult. Nevertheless, as these times are needed for obtaining a correct timing analysis, an approximation must be provided for them. Since this work targets HRT systems, the approximations are only valid if upper bounds to the real timing values are provided. Hence, if not stated otherwise, we always mean the upper bound of the actual admission time and transportation time obtained for the worst case when we talk about t^a and t^t.

The admission time $t^a_{i,j}$ for a route $\vec{r}_{i,j}$ from n_i to n_j is dependent on the TDM schedule's local view of the node n_i. This view describes the time slots, when n_i is allowed to send flits along that route $\vec{r}_{i,j}$. If there is only one slot for sending a flit along that route in the schedule, the admission time $t^a_{i,j}$ is equal to the schedule's period ($t^a_{i,j} = T$) for all flits when assuming the worst case. However, in application specific schedules often there are multiple time slots available in each TDM cycle. In those cases the actual worst-case admission time $t^a_{i,j}$ may vary depending on the time when the flit is ready for injection to the NoC. Figure 4.11

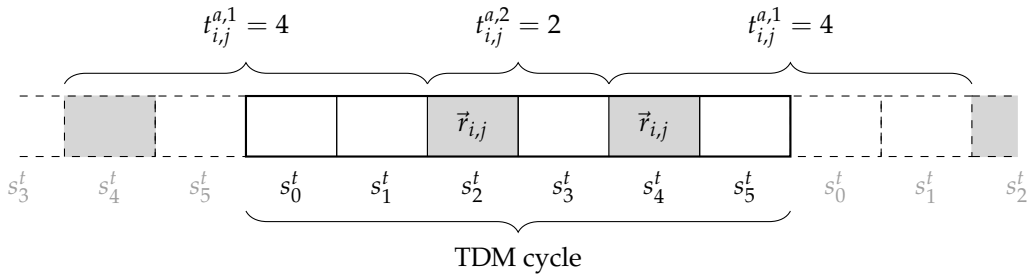


Figure 4.11: The local view of a node n_i participating in a TDM schedule. In the schedule there are multiple time slots available in one TDM cycle to deliver flits to a node n_j. Each TDM cycle exhibits a total amount of six time slots and the appropriate time slots for $\vec{r}_{i,j}$ are slot s_2^t and s_4^t. The admission time of the first time slot s_2^t is $t_{i,j}^{a,1} = 4$ in the worst case and for the second slot s_4^t it is $t_{i,j}^{a,2} = 2$.

illustrates that situation. The admission time for a given route $\vec{r}_{i,j}$ is always considered from the previous available time slot for that route to the current one. In the example of Figure 4.11 the admission time of a flit for routes $\vec{r}_{i,j} \in tdm^R$ is $t_{i,j}^{a,2} = 2$ if the flit becomes ready to send within time slots s_2^t or s_3^t. In all other time slots the admission time is $t_{i,j}^{a,1} = 4$.

As the determination of the exact point in time when a flit is ready to send is extremely difficult, an upper bound must be developed that fits for all flits of a message on a route $\vec{r}$. The straightforward upper bound is the maximum of all admission times $t^{a,x}$.

$$t_{i,j}^{a,max} = \max\{t_{i,j}^{A}\} \tag{4.84}$$

For sending small messages consisting of only one flit, this is the valid upper bound for t^a. However, the bound leads to a high overestimation if multiple flits are sent along the targeted route. Hence, a better approximation is preferred to reduce the overestimation when sending a message consisting of multiple flits. This is possible since we assume that there is no concurrent delivery of multiple messages allowed that start from the same node. Since we need to consider all admission times used for the flits of a message that are sent before the actual flit, we use the begin of sending the message as reference point $t = 0$ for the calculation of the admission time of the x-th flit of a message $t_{i,j}^{a,msg.x}$. To show the approximation for $t_{i,j}^{a,msg.x}$ some additional definitions must be provided.

Subsequently, we use the operation $\geq (A)$ to generate a list with the elements given by a set A which is sorted according to the operation $\geq$. In case of a set of times the sorting is in descending order according to the length of the times. Thus, the first element exhibits the longest time. Furthermore, the $\{.\}$-operator can be used to refer to a specific element of the tuple. Thus, following the expression $\langle \ldots \rangle.i$ refers to the i-th element of the tuple.

If the number of flits of a message is less or equal to the number of flits that are possible to send along the regarded route within a TDM cycle, the admission time of the message can be calculated by adding up the appropriate number of the longest admission times of the TDM cycle. This procedure is formulated as follows.

$$t_{i,j}^{am,f} = \sum_{l=0}^{f-1} (\geq (t_{i,j}^{A})).l \tag{4.85}$$

Since all admission times $t_{i,j}^{a,x}$ of a schedule add up to the value of the schedule's period T, the calculation of a message's admission time with a message consisting of more flits than time slots available in one TDM cycle is calculated as follows.

$$t_{i,j}^{am,f} = \left\lfloor \frac{f}{|t_{i,j}^{A}|} \right\rfloor \cdot T + \sum_{l=0}^{(f \bmod |t_{i,j}^{A}|)-1} (\geq (t_{i,j}^{A})).l \tag{4.86}$$

Thereby, the first term is used to add up the time for all completely used TDM cycles and the second term calculates the timing of the remaining flits which are sent by applying only partly used TDM cycles. These remaining flits belong to at most two partly utilized TDM cycles. One of those TDM cycles occurs when the message's delivery is starting in the middle of a TDM cycle. In those cases this TDM cycle is only used to some extend for sending the message. The other partly applied TDM cycle is used to send the last flits of a message that do not occupy a complete TDM cycle any more.

Equation 4.86 always considers the highest $t_{i,j}^{a,x}$ for the utilized time slots of non complete TDM cycles. Thereby, it does not matter which flits of the message cause the partial usage of a TDM cycle. The equation holds if the usage of these time slots is caused by the first or the last flits of the message or from some flits on both sides. Thereby, the equal arrangement of the time slots in each TDM cycle ensures the correctness as the same time slots of the TDM schedule are never used in the first and in the last TDM cycle for the same message. At the beginning of the message the last time slots of a TDM schedule are applied, while at the end of the message the first time slots of the schedule are used. An overlapping is not possible as this would lead to the consideration of one additional complete TDM cycle. Please note that Equation 4.86 is a generalization of Equations 4.84 and 4.85.

After presenting the building blocks for the timing's calculation, we take a closer look to the calculation of the bandwidth and latencies. As stated in Equation 2.27 the bandwidth is calculated using the number of paths per TDM cycle for a given route $\vec{r}$, the amount of data per flit and the schedule's period T. The number of paths can also be given by the number of usable time slots per TDM cycle. Thus, the bandwidth for the communication from a node n_i to another node n_j in terms of flits per time unit is given as follows.

$$B_{i,j}^{tdm} = \frac{|t_{i,j}^{A}|}{T} \tag{4.87}$$

The calculation of the worst-case latency for sending a message of f flits is based on Equation 2.30, but for the admission time we use the time for the complete message which is calculated with Equation 4.86 and the transportation time of the applied route is stated in Equation 4.83. Thus, the worst-case latency of a message from node n_i to node n_j consisting of f flits is calculated as follows.

$$WCTT^{msg} = t_{i,j}^{am,f} + t_{i,j}^{t} \tag{4.88}$$

4.2.2 Application to General-Purpose TDM Schedules

Subsequently, we apply the previously described general approach for the calculation of bandwidth, latency and their building blocks admission and transportation time to the

schedules presented in Section 4.1. A common characteristic shared by all introduced schedules is that each node exhibits only one applicable time slot per TDM cycle for sending data to a specific destination node. Thus, the admission times for all flits of a message are the same and can be upper bounded to the time of the regarded schedule's period T. Furthermore, except for the All-to-All schedule each of the flits travelling along a path defined by the schedules reach their destination node within two rounds. Thereby, they utilize the first round for horizontal delivery and the second round is used for the vertical transportation. Though some paths exhibit a shorter latency L^{path}, we use an upper bound for the transportation time of all paths that is exactly the time needed for performing two rounds of a schedule. With that approximation it is possible to give latencies that are independent of the localization of the source and destination node, which enables a timing analysis independent of task placement.

As already mentioned in Section 2.3 the building blocks for the different communication pattern are the communication from one sender to multiple receivers (1:N) and on the other hand the communication from multiple sender nodes to one receiver node (N:1). Furthermore, Section 4.1 states the dimension of the considered NoC as n, which causes a NoC size of n^2 nodes. In the remainder of this work let χ be the number of receivers (1:N) and the number of senders (N:1), respectively. We use f for the number of flits in one message. The time needed to perform one round is indicated as T^r while T represents the time for a complete TDM cycle consisting of multiple rounds.

The schedules All-to-One, One-to-All and One-to-One require an equal time for transportation, which is in particular maximal one round for each direction in the network (horizontal and vertical) and thus t^t equals $2 \cdot T^r = 2nL^{hop}$ cycles. As the buffers in the corner routers are only used to store flits arrived early to wait for delivery in the next round, the time a flit stays in those buffers is already included in the stated transportation time. In contrast to the transportation time, the admission time differs for the considered schedules. While One-to-All and All-to-One are characterized by different admission times for 1:N and N:1 communications, the One-to-One schedule requires an equal time for both cases.

In the One-to-All schedule a node may only send one flit per TDM cycle but is allowed to receive $n^2 - 1$ flits in a TDM cycle consisting of n rounds. As a round takes $T^r = n \cdot L^{hop}$, a period lasts $T = n \cdot T^r = n^2 L^{hop}$. When considering 1:N communication, each flit has to be sent to all χ receivers. Hence, χ TDM cycles are needed to deliver one flit. Thus, a 1:N communication utilizing One-to-All takes $t^a = T \cdot \chi \cdot f = n^2 \chi f L^{hop}$ cycles for admission. In summary the WCTT is

$$WCTT_{1A}^{1:N} = (n^2 \chi f + 2n) L^{hop} \tag{4.89}$$

In contrast, a flit can be received from all senders within one period. Therefore, the admission takes $t^a = T \cdot f = n^2 f L^{hop}$ cycles for N:1 communication:

$$WCTT_{1A}^{N:1} = (n^2 f + 2n) L^{hop} \tag{4.90}$$

The bandwidth of the One-to-All schedule behaves differently for the senders and receivers as well as for the consideration of the applied communication (1:N or N:1). As a sender is allowed to send at most one flit per TDM cycle the bandwidth on sender side is equal for both communications.

$$B_{1A,snd}^{tdm,1:N} = B_{1A,snd}^{tdm,N:1} = \frac{1}{T} = \frac{1}{n^2} \tag{4.91}$$

However, on the receiver side the schedule behaves differently. In the N:1 communication there are χ different senders and each one sends one flit to the same receiver. Thus, the bandwidth depends on the number of sender nodes χ.

$$B_{1A,rcv}^{tdm,N:1} = \frac{\chi}{T} = \frac{\chi}{n^2} \tag{4.92}$$

On the other hand the receivers' bandwidth for the 1:N communication is mainly restricted by the data delivery of the sender node. As that node is only able to send one flit in χ TDM cycles to a particular receiver, the bandwidth on receiver side is restricted as follows.

$$B_{1A,rcv}^{tdm,1:N} = \frac{1}{\chi T} = \frac{1}{\chi n^2} \tag{4.93}$$

The All-to-One schedule behaves vice versa to One-to-All. A node can send $n^2 - 1$ flits and receive only one flit per TDM cycle. Hence, the schedule exhibits admission times that are vice versa to One-to-All for 1:N and N:1 communication. Its admission time takes $n \cdot T^r \cdot f = n^2 f L^{hop}$ cycles for 1:N communication and $n \cdot T^r \cdot \chi \cdot f = n^2 \chi f L^{hop}$ cycles for the N:1 communication.

$$WCTT_{A1}^{1:N} = (n^2 f + 2n) L^{hop} \tag{4.94}$$

$$WCTT_{A1}^{N:1} = (n^2 \chi f + 2n) L^{hop} \tag{4.95}$$

Similar to the WCTTs the bandwidth behave vice versa, too. Thus, the bandwidth of a sender node is stated as follows.

$$B_{A1,snd}^{tdm,1:N} = \frac{\chi}{T} = \frac{\chi}{n^2} \tag{4.96}$$

$$B_{A1,snd}^{tdm,N:1} = \frac{1}{\chi T} = \frac{1}{\chi n^2} \tag{4.97}$$

The bandwidth of Equation 4.97 is caused by the restriction about receiving flits, which is one flit per TDM cycle. As there are χ nodes that want to send flits to the same node each sender node is allowed to send one flit in every period χT to meet the given constraint. On the other hand the receiver node exhibit a bandwidth of

$$B_{A1,rcv}^{tdm,1:N} = B_{A1,rcv}^{tdm,N:1} = \frac{1}{T} = \frac{1}{n^2} \tag{4.98}$$

Since each flit must be sent to all χ receivers (1:N), in One-to-One it takes χ TDM cycles (each targeting a different destination node) until one flit is fully delivered to all destinations. Therefore, the admission of f flits takes $t^{a,msg.f} = T \cdot \chi \cdot f = n \chi f L^{hop}$ cycles:

$$WCTT_{11}^{1:N} = WCTT_{11}^{N:1} = (n\chi f + 2n) L^{hop} \tag{4.99}$$

In each TDM cycle it is possible to send at most one flit of data from one node to one other node and also to receive at most one flit per node. Therefore, the bandwidth for sender and receiver nodes as well as the communications 1:N and N:1 are equal.

$$B_{11,snd}^{tdm} = B_{11,rcv}^{tdm} = \frac{1}{T} = \frac{1}{n} \tag{4.100}$$

To meet the constraints needed for performing an All-to-All schedule, Section 4.1 states that a period and thus, the admission time t^a must be set to $\frac{n^2(n+1)}{2}$. Additionally, when

assuming that $n \geq 2$, Equation 4.40 about the buffering time in the corner routers of the All-to-All schedule can be upper bounded according to the following relation.

$$\Delta t_z^{aa} \leq 1 + \frac{n^2}{2} - \frac{n}{2} \leq \frac{n^2}{2} \tag{4.101}$$

Thereby, Δt_z^{aa} exhibits its maximal values when setting the parameters r, d and q to the values $r = 0$, $d = 0$ and $q = n - 1$. Thus, in All-to-All a flit stays maximal $\frac{n^2}{2}$ hop latencies in the buffer of the corner router and is transported n hops per ring. Thus, All-to-All's transportation time t^t is set to $\frac{n^2}{2} + 2n$. Since the schedule prohibits sending multiple flits from the same sender to the same receiver in one TDM cycle, the admission time has to be considered for each flit f. The argumentation holds for 1:N as well as for the N:1 communication with All-to-All and hence, both communications exhibit the same WCTT. Therefore, the WCTT for a 1:N and N:1 communication utilizing All-to-All is

$$WCTT_{AA} = \left(\frac{n^2(n+1)}{2} \cdot f + \frac{n^2}{2} + 2n \right) L^{hop} \tag{4.102}$$

Since in the All-to-All schedule each can send one flit to each other node within one TDM cycle, the actual bandwidth is dependent on the performed communication (1:N or N:1) and the number of participating nodes. For the 1:N communication it is possible to send a flit to all participating nodes within one TDM cycle, which are in total χ flits of data. However, on the receiver side each node is restricted to only receive one flit per TDM cycle for the applied communication (1:N), because the sender can only send on flit per TDM cycle to that particular receiver node. Thus, the All-to-All schedule exhibits the following bandwidths for a 1:N communication.

$$B_{AA,snd}^{tdm,1:N} = \frac{\chi}{T} = \frac{2\chi}{n^2(n+1)} \tag{4.103}$$

$$B_{AA,rcv}^{tdm,1:N} = \frac{1}{T} = \frac{2}{n^2(n+1)} \tag{4.104}$$

Since there are χ senders and one receiver for the N:1 communication the bandwidth for that communication behaves exactly vice versa.

$$B_{AA,snd}^{tdm,N:1} = B_{AA,rcv}^{tdm,1:N} = \frac{1}{T} = \frac{2}{n^2(n+1)} \tag{4.105}$$

$$B_{AA,rcv}^{tdm,N:1} = B_{AA,snd}^{tdm,1:N} = \frac{\chi}{T} = \frac{2\chi}{n^2(n+1)} \tag{4.106}$$

The One-to-All and All-to-One schedules exhibit the problem of providing low latencies and high bandwidth characteristics for only one kind of communication (1:N or N:1). Hence, the schedule enhancements presented in Section 4.1.2 follow the intention of providing good characteristics for both communication directions. The Alternate schedule obtains a period of $T = 2n^2 L^{hop}$. Furthermore, the decision about which part of the schedule is actually used (One-to-All or All-to-One) depends on the direction of the targeted communication. The 1:N communication applies the All-to-One part, while the N:1 communication is put to the One-to-All part of the schedule. Thus, we get the following timing behaviour:

$$WCTT_{Alt}^{1:N} = WCTT_{Alt}^{N:1} = (2n^2 f + 2n) L^{hop} \tag{4.107}$$

Since the applied basic schedule parts are chosen according to the performed communication (1:N or N:1), the bandwidths behave accordingly.

$$B_{Alt,snd}^{tdm,1:N} = B_{Alt,rcv}^{tdm,N:1} = \frac{\chi}{T} = \frac{\chi}{2n^2} \tag{4.108}$$

$$B_{Alt,rcv}^{tdm,1:N} = B_{Alt,snd}^{tdm,N:1} = \frac{1}{T} = \frac{1}{2n^2} \tag{4.109}$$

The second schedule presented in Section 4.1.2 is the Triplet schedule. It combines the Alternate and the One-to-One schedule to take advantage of the One-to-One schedule's low admission time when communicating in small groups. In this schedule the length of the period differs depending on which part (One-to-One or Alternate) is used. The One-to-One part leads to a period of $T = 2nL^{hop}$ while the other part exhibit a period of $T = 2n^2L^{hop}$. Although the One-to-One part has a very small period it must be considered that this type of schedule needs more TDM cycles than the Alternate part of the schedule to deliver a flit from or to N senders or receivers, respectively. Hence, the admission time for sending or receiving f flits to or from χ nodes takes $t^a = 2n\chi fL^{hop}$ when using the One-to-One part and $t^a = 2n^2fL^{hop}$ for the Alternate part. Thereby, the One-to-One part is applied for group sizes smaller than n, which is the break even point for the admission time of both parts of the schedule and the Alternate part is used otherwise. Thus, the WCTTs for each communication direction is shown here:

$$WCTT_{Tri}^{1:N} = WCTT_{Tri}^{N:1} = (2nf \cdot \min(\chi, n) + 2n)L^{hop} \tag{4.110}$$

Similar to the WCTTs the bandwidths are dependent on the applied part of the schedule. Thus, with Equations 4.100, 4.108 and 4.109 we get the following bandwidths.

$$B_{Tri,snd}^{tdm,1:N} = B_{Tri,rcv}^{tdm,N:1} = \begin{cases} \frac{1}{2n} & \text{if } \chi \leq n \\ \frac{\chi}{2n^2} & \text{if } \chi > n \end{cases} \tag{4.111}$$

$$B_{Tri,rcv}^{tdm,1:N} = B_{Tri,snd}^{tdm,N:1} = \begin{cases} \frac{1}{2n} & \text{if } \chi \leq n \\ \frac{1}{2n^2} & \text{if } \chi > n \end{cases} \tag{4.112}$$

4.3 Discussion

In this section we evaluate each presented schedule using the formulas for 1:N and N:1 from Section 4.2.2 to get a basic understanding of each schedule and provide a comparison among them. Afterwards, we compare the worst-case performance guarantees of general-purpose schedules to application specific schedules and discuss their differences.

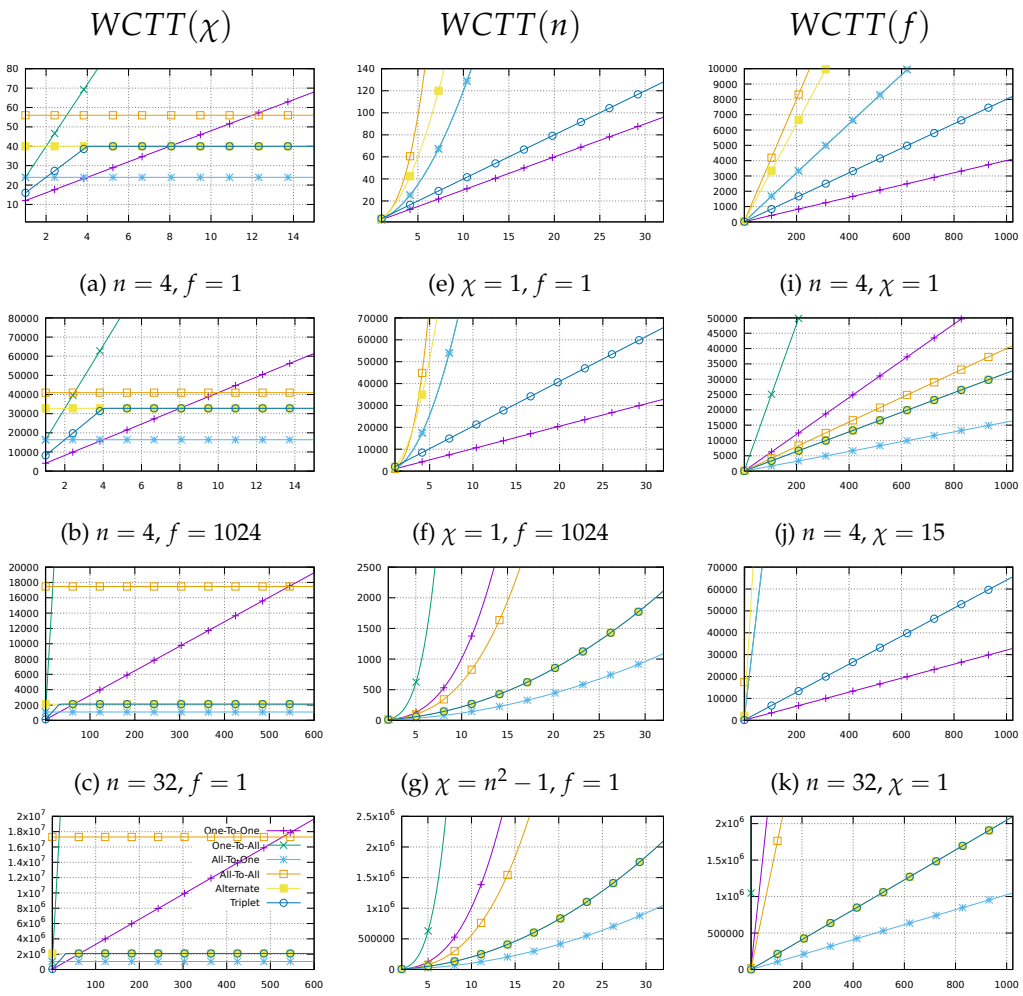
4.3.1 Comparison of General-Purpose TDM Schedules

For the evaluation we assume different group sizes (χ), NoC sizes (n^2) and message lengths (f). Thereby, two of the mentioned parameters will be fixed and the schedules will be compared as a function of the third parameter. Finally, a statement about the applicability of the schedules is given.

Figure 4.12 shows the behaviour of considered schedules for 1:N communication. Please note that N:1 communication exhibits the same WCTTs for all schedules as 1:N communication, except that One-to-All and All-to-One behave exactly vice versa. Thus, we omit the presentation of the timing for the N:1 communication here due to the small differences. As already stated, when looking at the state-of-the-art schedules, the All-to-All and One-to-One schedules exhibit the same behaviour for both kinds of communication, while One-to-All and All-to-One behave differently. The new schedules Alternate and Triplet exhibit an equal timing behaviour for 1:N and N:1 communication, too. The three columns of Figure 4.12 each depicts the variation of one parameter χ (Figure 4.12a to 4.12d), n (Figure 4.12e to 4.12h) or f (Figure 4.12i to 4.12l). In each column there are four diagrams that display the WCTTs as a function of the variable parameter and fixed values for the other parameters. Thereby, for each diagram the fixed parameters are set to two different values. In all columns the first row obtains low values for both constant parameters, while in the second and third row in each case one of the parameters is set to a high value and one to a low value. In the last row both parameters are set to high values. With that variations it is possible to investigate the timing behaviour of the schedules under different circumstances. The legend for all diagrams is only depicted in the lower left image (Figure 4.12d) to support readability.

Figures 4.12a to 4.12d display the schedules' behaviour as a function of χ. Thereby, in Figure 4.12a the dimension of the NoC is fixed to a size $n = 4$, which causes a NoC with $n^2 = 16$ nodes, and the delivered message comprises 1 flit. When looking at the state-of-the-art schedules, the WCTT of All-to-All and All-to-One is constant for all group sizes, while the schedules One-to-One and One-to-All increase linearly with χ. However, despite All-to-All is not increasing, its WCTT exhibiting 56 cycles is the highest for $\chi < 4$ and the second highest time for $4 \leq \chi < 12$. One-to-All performs worse than All-to-All for $\chi > 5$ due to a high gradient caused by a factor of n^2 and One-to-One features a higher WCTT for $\chi > 37$. However, the One-to-One schedule performs best for group sizes smaller than n but is beaten for larger group sizes by All-to-One, which features a WCTT of 24 cycles for all χ. The new schedules show an intermediate timing behaviour in this case. Both do not exceed a WCTT of 40 cycles for all group sizes which is between the WCTTs of the All-to-All and the All-to-One schedule. However, the Triplet schedule shows a decrease in WCTT for group sizes $\chi < n$. Thereby, it exhibits with 16 cycles its best value for $\chi = 1$, which is near the performance of the One-to-One schedule (12 cycles for $\chi = 1$).

Figure 4.12b shows the timing behaviour for large messages. A comparison with Figure 4.12a exhibits that the behaviour of the schedules relative to the other schedules is quite similar, because the number of flits influence each WCTT in the same way. Nevertheless, there are small relative differences when comparing Figure 4.12a and Figure 4.12b. However, the main difference is the total increase in WCTT for each schedule, which is caused by the additional data that must be transported.



(a) $n = 4$, $f = 1$ (e) $\chi = 1$, $f = 1$ (i) $n = 4$, $\chi = 1$

(b) $n = 4$, $f = 1024$ (f) $\chi = 1$, $f = 1024$ (j) $n = 4$, $\chi = 15$

(c) $n = 32$, $f = 1$ (g) $\chi = n^2 - 1$, $f = 1$ (k) $n = 32$, $\chi = 1$

(d) $n = 32$, $f = 1024$ (h) $\chi = n^2 - 1$, $f = 1024$ (l) $n = 32$, $\chi = 1023$

Figure 4.12: WCTT of 1:N communication. Each diagram varies one parameter while the other parameters remain fixed. In the left column the timing is depicted as a function of the group size χ, in the middle the WCTT is stated as a function of the number of nodes per NoC dimension n and in the right column the diagrams show the timing as a function of the message size f. The caption of each diagram states the configuration for the fixed parameters. Thereby, it follows the specification of showing low values in the first row, exhibiting one low and one high value in the second and third row and displaying the configuration when both parameters are high in the last row. The shape of the plotted lines is the same for all figures and the according legend is plotted in the lower left diagram. Please note the different scales for WCTT in each diagram.

When looking at a large NoC with 1024 nodes ($n = 32$) like in Figure 4.12c and 4.12d, one can see an increase of the WCTTs for all schedules compared to smaller NoCs. Nevertheless, it is to mention that the All-to-All schedule exhibits with 17472 cycles a very high value for all group sizes and the schedule performing worst is the One-to-All schedule. Also, the One-to-One schedule exhibits a very high WCTT for large group sizes. On the other hand the Alternate and Triplet schedules show a WCTT not higher than 2112 cycles for all group sizes and the All-to-One schedule performs best for $\chi \geq n$ with 1088 cycles. One-to-One shows the lowest WCTTs for $\chi < n$. Again nearly the only difference for small and large messages is a factor which is quite similar for all schedules.

The behaviour as a function of n is shown in figures 4.12e to 4.12h. The group sizes are fixed to $\chi = 1$ for investigating small groups and $\chi = n^2 - 1$ for large groups the same message lengths are applied as for the investigation of χ ($f = 1$ and $f = 1024$). When looking at small groups the All-to-All schedule raises with $O(n^3)$, whereas One-to-One increases arithmetically. One-to-All and All-to-One feature the same WCTT for $\chi = 1$ and raise faster than One-to-One but slower than All-to-All, as both are characterized by $O(n^2)$. The new schedules behave different, too. While the Alternate raises with $O(n^2)$ the Triplet schedule shows an arithmetical increase of the WCTT. Thus, while the Alternate schedule is nearly as expensive as the All-to-All schedule in this case, the Triplet schedule exhibits the second lowest WCTT. Again shifting to larger messages does not effect the comparison of the schedules dramatically except for the increase of the absolute times. In Figures 4.12g and 4.12h the group size is set to the maximal available size for each considered NoC. Thus, the size is set to $\chi = n^2 - 1$ and hence, also varies for each considered n. This, configuration causes the One-to-All schedule to raise with $O(n^4)$ and the One-to-One schedule to raise with $O(n^3)$. The other schedules raise with an equal rate as for the small group size, which is $O(n^3)$ for the All-to-All schedule and $O(n^2)$ for All-to-One, Alternate and Triplet. Thereby, the best performance shows the All-to-One schedule and the values of the Alternate and Triplet are equal and show the second lowest timing.

Figures 4.12i to 4.12l show, that the WCTTs of all schedules increase arithmetically with the number of flits. However, there are big differences in their gradients. The highest WCTT of all schedules for small group sizes exhibits All-to-All. This holds true for all sizes of a message. However, its performance compared to the other schedules gets better for the large group size. All-to-One performs better than One-to-One as soon as χ raises above a size of n and the Triplet schedule is outperformed if χ is larger than $\frac{n}{2}$. Hence, the All-to-One beats all schedules for the large group size. For small groups the One-to-One schedule performs best for all message sizes, no matter if a small or a larger NoC is applied. The second best schedules for the large group size are the Alternate and Triplet schedules. However, for small groups only the Triplet schedule remains competitive.

In summary, All-to-All schedule is not suitable for performing pure 1:N or N:1 communication. It is intended to perform a complete all-to-all communication (N:N) within one period. Since the considered communications send n times less flits than all-to-all, a lot of free bandwidth remains unused and causes a huge overhead in time. The schedules One-to-All and All-to-One are optimised each for one kind of communication, One-to-All for N:1 and All-to-One for 1:N. They are able to perform the communication they are optimized for within one period and therefore, are stable due to different group sizes. However, the op-

posite communication direction causes significant overhead and performs even worse than All-to-All for moderate and large group sizes. In contrast, One-to-One fits to both communications, but its WCTT is dependent on the group size. Hence, the schedule performs best for $\chi < n$ in all cases compared to the other schedules and still exhibits competitive traversal times for moderate χ. Nevertheless, its behaviour is becoming infeasible for large group sizes. None of the two newly presented schedules can perform best in any of the displayed configurations. However, the Alternate schedule shows an applicable performance for all configurations with a group size of $\chi \geq n$. Furthermore, in contrast to all other schedules the Triplet schedule provides an acceptable performance for all displayed configurations. This universality is an unique selling point for that schedule.

4.3.2 Comparison of Application Specific versus General-Purpose TDM Schedules

Application specific schedules are hard to compare to general-purpose schedules due to their dependence on the executed application and mapping to hardware nodes. Thus, we provide two representatives for application specific schedules and compare them to the presented general-purpose schedules by varying the parameters group size χ, NoC size n and message length f. The first application specific schedule is optimal regarding to the investigated communication, which results in a schedule that only consists of the needed paths and the nodes are arranged in a way that they obtain short latencies for flit traversal. Such a schedule typically is not applicable for real applications, as there is only one communication in one direction. However, it provides the theoretical optimum for the communication handled with TDMA, if all circumstances are in a way that they do not hinder the communication. On the other hand the second schedule assumes that each core wants to communicate with each other core of the NoC with the same bandwidth, which causes the paths to fully connect each node with each other one. Thus, a high contention must be handled in that schedule. Please note that we do not compare the general-purpose schedules with an application specific schedule that grants higher bandwidths to paths competing with the investigated communication. The reason is that the natural extrema for that case is to reduce the investigated communication's bandwidth to zero. Since the extrema would lead to a infinite latency, we do not consider that case as a practical baseline for comparing application specific schedules with general-purpose ones.

We use the same communication as in Section 4.3.1 for this evaluation. Therefore, a sender is considered that delivers a message of f flits to χ different receiver nodes (1:N). Furthermore, we omit the communication where χ senders communicate with one receiver as the timing behaviour is the same except for the One-to-All and All-to-One schedules which behave exactly vice versa.

The optimal schedule for the 1:N communication with χ receivers is a schedule that consists of only χ paths arranged in a way that the sender is possible to send a flit to one of the receivers in each time slot. With such a schedule the period of the schedule is as follows.

$$T_{opt}^{1:N} = \chi L^{hop} \tag{4.113}$$

Furthermore, all nodes participating in the communication are placed as close as possible

to each other in the NoC which causes a shape of a square and a maximal horizontal and vertical distance from the sender to a receiver of $\lceil\sqrt{\chi}\rceil$. Thus, this schedule exhibits the following admission and transportation times.

$$t^a_{opt} = \chi L^{hop} \tag{4.114}$$

$$t^t_{opt} = \lceil 2\sqrt{\chi}\rceil L^{hop} \tag{4.115}$$

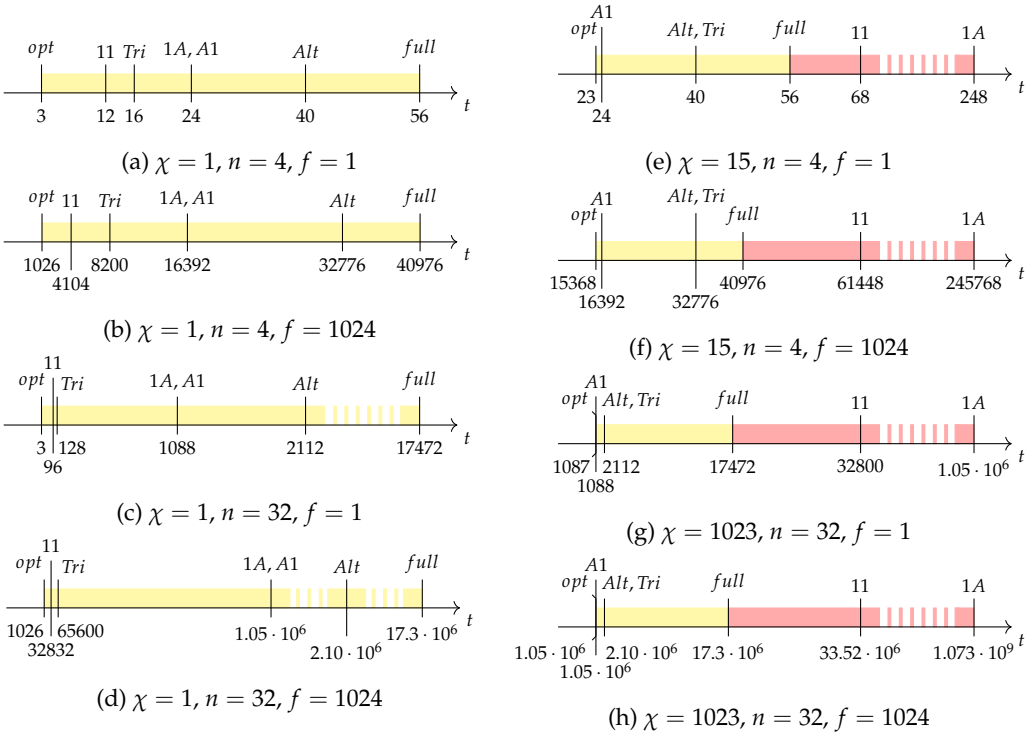
An optimal schedule for a N:1 communication arranges the paths in a way that the receiver node can receive one flit per time slot. Additionally, when the placement is optimized similar to the previously described one, this schedule leads to the same period, admission and transportation time as the optimal schedule for the 1:N communication. Thus, for both optimal schedules the WCTT is given as follows.

$$WCTT_{opt} = (\chi f + \lceil 2\sqrt{\chi}\rceil)L^{hop} \tag{4.116}$$

In contrast, the second application specific schedule additionally assumes the communication of all nodes to all other nodes. In fact the All-to-All schedule is an application specific schedule that exhibits an equal bandwidth from each node to each other node. Hence, we take the All-to-All schedule as representative for fully connected application specific schedules with equal bandwidths. Therefore, we get the following WCTT for the fully connected application specific schedule.

$$WCTT_{full} = WCTT_{AA} = \left(\frac{n^2(n+1)}{2}\cdot f + \frac{n^2}{2} + 2n\right) L^{hop} \tag{4.117}$$

Figure 4.13 shows the comparison of the application specific schedules with the general-purpose schedules in terms of WCTT. The left column (Figure 4.13a to 4.13d) illustrates the timing for the smallest possible group size. Thereby, one sender is transmitting data to one receiver, aka it depicts a 1:1 communication. With that configuration the performance of each general-purpose schedule lies between the theoretical optimum and the fully connected schedule. The two best general-purpose schedules are the One-to-One and the Triplet schedule for all such configurations. Despite their larger WCTT compared to the optimal one, they are both much nearer to the optimum than to the fully connected schedule. On the other hand the One-to-All and All-to-One schedules exhibit the same WCTT for a 1:1 communication. They are quit in the middle between both application specific schedules for small NoC sizes but still nearer to the optimum than to the fully connected schedule. For large NoCs there is still a large gap between the One-to-All and All-to-One and the nearly optimal schedules (One-to-One and Triplet). However, relatively to the application specific schedules they are much nearer to the optimum than to a fully connected schedule. The Alternate schedule is nearest to the fully connected schedule and for small NoCs also nearer to the All-to-All than to the optimum. However, the fully connected schedule (All-to-All) shows significant higher WCTTs for large NoCs than all other schedules which causes all general-purpose schedules to be much nearer to the optimum than to the fully connected schedule. The reason for the bad performance of the All-to-All in terms of WCTT is the huge number of connections between all nodes that the TDMA mechanism needs to handle.



(a) $\chi = 1, n = 4, f = 1$

(b) $\chi = 1, n = 4, f = 1024$

(c) $\chi = 1, n = 32, f = 1$

(d) $\chi = 1, n = 32, f = 1024$

(e) $\chi = 15, n = 4, f = 1$

(f) $\chi = 15, n = 4, f = 1024$

(g) $\chi = 1023, n = 32, f = 1$

(h) $\chi = 1023, n = 32, f = 1024$

Figure 4.13: Comparison of application specific and general-purpose schedules in terms of WCTT. Each diagram varies one parameter while the other parameters remain fixed. In the left column the timing is depicted for the smallest possible group size $\chi = 1$ and in the right column the diagrams show the timing when all nodes of a NoC participate in the communication. The caption of each diagram states the corresponding parameter configuration. Thereby, it follows the specification of permuting the parameters in a way that there is a high and a low value for each parameter and each possible combination is depicted in one figure. The vertical lines are on top labeled with a schedule and mark the position of that schedule on the time line. Additionally, below each mark their is a number that shows the actual WCTT for the schedule with the corresponding configuration. To assist the comparison of application specific and general-purpose schedules, the yellow bar indicates the range from the theoretical optimal and the fully connected application specific schedule. Additionally, the red bar indicates the areas that are outside the application specific range. A dashed bar indicates that the length of that bar is not proportional to the time values given on the depicted time line.

The right column of Figure 4.13 (Figure 4.13e to 4.13h) shows the comparison of application specific and general-purpose schedules when one sender is communicating with all other nodes of the NoC, which act as receivers. Thus, a 1:N communication is illustrated where all nodes of the NoC participate in. With that configuration the Alternate and the Triplet schedule always show the same WCTT. Furthermore, now there are schedules that behave worse than the fully connected (All-to-All) schedule. These are the One-to-One and the One-to-All schedule in all depicted cases. The reason is that these two schedules are only allowed to send one flit within one TDM cycle no matter if the flits target the same receiver or different ones. On the other hand the other schedules are allowed to send multiple flits within one TDM cycle if they target different receivers.

The best general-purpose schedule is the All-to-One schedule which exhibits only small differences to the theoretical optimum. Despite this good results it is to mention that this schedule is optimized for 1:N communication. When looking at the N:1 communication the One-to-All and All-to-One schedules would be exchanged in all illustrations of Figure 4.13. The second best general-purpose schedules for large group sizes are the Alternate and the Triplet schedule. They are quite in the middle between the optimum and the fully connected schedule for small NoCs but with a tendency to the fully connected one. This tendency raises with the size of the message sent. However, for large NoCs they are much nearer to the optimum than to the fully connected schedule. In concrete the WCTTs of the Alternate and Triplet schedule are always around twice as high as the optimum. Thus according to the large increase of the All-to-All's time when increasing the NoC size, relatively seen the composed general-purpose schedules move nearer to the optimum for large NoC sizes.

In summary one can say that there is for each configuration at least one general-purpose schedule that exhibits a WCTT relatively near to the theoretical optimum. These are the One-to-One schedule for small group sizes and the All-to-One or One-to-All schedules for large group sizes depending on the applied communication 1:N or N:1. However, for all three schedules there are configurations where they are definitely not competitive to application specific schedules. Since this can be a relatively strong disadvantage, the Triplet schedule tries to overcome these problems by applying a trade-off between performance and generality. Hence, in no configuration it is the winner against all presented schedules, but it is relatively near to the theoretical optimum in most cases and significantly better than the fully connected schedule in all cases.

5

Timing Analysis of MPI-based Communication Operations

Based on a mechanism to guarantee timing upper bounds for the transportation of data (see Chapter 4), it is possible to calculate the worst-case timing for delivering a message with an appropriate operation. In addition to the pure admission and tranportation times the consideration of a communication operation comprises parts of local computations to manage the correct delivery of all flits. Since MPI provides standard patterns that are commonly used in parallel applications we target the analysis of those operations to provide calculations that can be used for the timing analysis of complete parallel programs as a module. Hence, for a given communication operation we provide one calculation that can be parameterized with a configuration (including for example the message length) that fits to the program under analysis. The investigated communication operations include basic point-to-point communication but also more complex operations. For those operations there exist sophisticated state-of-the-art communication patterns to effectively perform the communication. We target with our analysis relevant patterns for the provided communications.

In this chapter we firstly state the assumptions we rely on for the analysis and present the overall workflow of the analysis of a communication operation. Afterwards, the formalism is presented that provides the calculations to obtain an operation's timing upper bound. Thereby, we firstly focus on point-to-point communications. Building on these results we extend the formalism to the timing of collective communiction. The timing of communications that are based on logical process topologies is similar to the one of collective communication with uniform data exchange, but we let this analysis to future work. The last part of this chapter provides a discussion about the analysed communication operations and their appropriate communication patterns. Thereby, we focus on the comparison of the relative proportions of the building blocks of an operation, compare the different communication

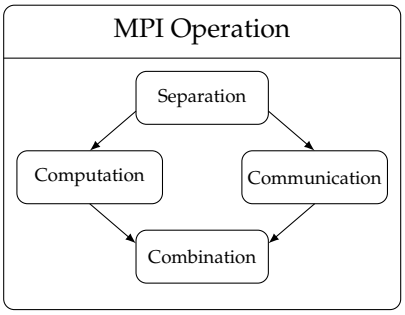


Figure 5.1: Workflow for timing analysis of parallel MPI operations. The workflow is split in four different phases. The rectangles mark these phases while the arrows mark precedence constraints.

patterns provided and propose the usage of patterns to perform best in terms of worst-case timing. Additionally, we examine potentials for enhancing the calculated timing bounds. Overall, this chapter is an extension of our former work [Ste+18].

5.1 Assumptions and Workflow for Communication Operations

As stated in Figure 5.1 the analysis of a communication operation is partitioned in four different challenges. The first of those challenges is the separation of parts with pure local computation and parts with pure communication tasks. The next challenges obtain the timing bounds of each of the identified tasks in the computation and communication parts and the last challenge combines the obtained timing to a full end-to-end timing upper bound for the communication operation.

When regarding to the separation of computation and communication we need to know the concrete points in code where both parts meet each other. There are basic send operations that serve as starting points for data transmission and receive operations mark the end of a transmission. Code example 5.1 and Table 5.1 show an example of how to separate code of a MPI operation to instances. The parts including local execution start at the beginning of a MPI operation or at a basic communication operation, while they end at a basic communication operation or the communication's end. To be precise the time needed to locally execute a communication operation must be considered, too. Thus, two computation and one communication instances meet on a single basic communication operation (send or receive), (see Figure 5.2). Thereby, the local execution of a basic send operation belongs to the previous computation instance, while the execution of a receive operation is part of the subsequent computation. Thereby, the communication partners of each point-to-point communication must be identified in order to define the concrete data transfer instances.

In Code example 5.1 and Table 5.1 each flit is separately sent by a basic operation. If hardware support for message-passing is used (e.g. a DMA) there may be only one basic software operation that targets complete messages consisting of a number of flits instead of targeting only one flit. The sending of each single flit is then implemented in hardware. In

code section id	starting point	end point
1	/* A */	/* B */
2	/* B */	/* C */
3	/* C */	/* C */
4	/* C */	/* D */

Table 5.1: Separation of code instances of a MPI operation. Starting and end points refer to comments of Code example 5.1. Please note that code section 3 refers to one iteration of the displayed loop.

Code example 5.1 Small pseudocode example for illustration of code section separation of a MPI operation. In this example each flit is sent by applying a separate basic send operation. The comments mark points of separation.

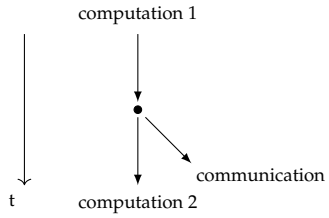
```
send_msg() {
        /* A */ ...
        send(first flit);
        /* B */ ...
        for(all remaining flits) { ...
                send(actual flit);
                /* C */ ...
        }
        ... /* D */
}
```

those cases the hardware's timing for sending or receiving one flit must be obtained by an appropriate technique or tool. Subsequently, we see this hardware timing as part of the local computation parts and can be used in an equal way as WCET upper bound of software. The only situation that must be treated differently is when a single local computation section is composed of a software and a hardware part. In those cases the timing of this section is calculated as follows:

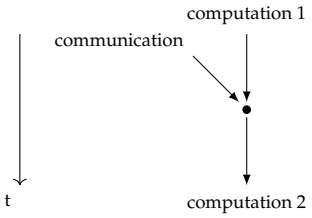
$$WCET^{local} = WCET^{SW} + WCET^{HW} \tag{5.1}$$

Here $WCET^{local}$ denotes the total worst-case response time of the considered local computation section and $WCET^{SW}$ and $WCET^{HW}$ indicate the worst-case timing of the software or hardware part, respectively. In the remainder of this work we see a basic send or receive operation as the last action performed before an actual send or receive of one flit takes place, no matter if it is done in hardware or software.

The timing of the local computation sections is exhibited by calculating their WCET bounds while the timing of the communication sections is stated by WCTTs. Please note that according to Section 2.2 we see a flit that is waiting in the NI to be released as a flit in send state and a flit that is waiting in the NI to be fully received by handling it locally as a flit in receive state. The border between WCET to WCTT is the point in time when data is entering the send state, which is at the end of the execution of the corresponding send operation. In contrary the border from WCTT to WCET is when data is arriving at the target node and thus, entering the receive state. Hence, the time a message stays in send state belongs to the WCTT, while the time a message remains in the receive state belongs to the



(a) Connection at a basic send operation. The dot marks the point in time when the local execution of the send operation finishes.



(b) Connection at a basic receive operation. The dot marks the point in time when the local execution of the receive operation begins.

Figure 5.2: Connection of computation and communication instances at basic communication operations

WCET. This is because the duration that data spends in send state is driven by the TDM schedule's admission time t^a whereas the time spend in the receive state is driven by the local execution time of the receiving node.

After describing the separation of the operation's tasks the focus is put to estimate the timing bound for the tasks. In the work at hand we consider that the worst-case timing bounds for the execution of local computations is obtainable with state-of-the-art methods. Thus, we see the acquiring of those timing bounds as out of scope of this work and assume that they are obtained by suitable methods or tools (e.g. OTAWA [Bal+11]).

The WCTT bound of each single flit must be considered for analysing MPI operations. Though typically there are a lot of flits delivered, the effort for the calculation of timing bounds is restricted. The reason is that under some assumptions a message sent along a certain path through the NoC exhibits the same timing for each flit of the message. These assumptions are that the maximum admission time t^a of each flit is equal for a certain process and that the transportation time t^t of each flit traversing the same path between two nodes remains the same, too. The assurance of those assumptions is guaranteed by applying one of the general-purpose schedules presented in Chapter 4.

As each node can emit a fixed number of flits per TDM cycle, the admission time of a message is directly related to the period of a TDM cycle and the number of flits forming the message. The transportation time of most flits is covered by the admission time of subsequent flits. Thus, only the last flit must be taken into account for the transportation time.

Under certain circumstances the assumption about the admission time does not hold. These circumstances may be for example the application of an application specific TDM schedule where multiple flits are allowed to enter the NoC in a non-uniform distribution within the same TDM cycle. In those cases an appropriate mapping of the actual timing behaviour to the utilized t^a for calculation must be provided to limit the effort for obtaining the WCTT bounds of flits. Details about obtaining the WCTT of a message is provided in Chapter 4 and forms the basis for the calculation in the remainder of this chapter.

After estimating the timing of the local computation and the communication parts it remains the correct assembling of all times to gain the end-to-end timing of the analysed

communication operation. Thereby, the procedures vary according to the applied communication patterns. Details about that procedures are provided in the subsequent Section 5.2.

Please note that the analyses and formalisms are based on the assumption that the receiver node is ready to receive data from the sender node. Therefore, the description of the timing analysis excludes the analysis of a rendezvous protocol. Nevertheless, the timing of these protocols needs to be considered in the analysis. If the protocol is realized in software, it consists of local code execution parts and flit traversal parts. Thus, the rendezvous can be analysed and integrated to the communication in the same way as the rest of the communication by using the approach stated in this section.

There are suitable synchronisation protocols realized in hardware, too. The work of Walter [Wal19] is an example of such a protocol. Next to a pure rendezvous protocol it additionally implements flow control to prevent buffer overflows in the receive buffers in the NIs. Please note that the worst-case timing for pure data delivery is not effected by the flow control in this protocol as it uses additional hardware (including network hardware) and the calculated worst-case timing is determined by the slowest part of execution which implicitly considers the slow down caused by the flow control. Only the costs for the rendezvous protocol at the beginning of a communication must be considered (as we do in the evaluation in Section 5.3).

Depending on the utilized rendezvous or synchronization protocol, the receive buffer may be able to overflow when the low level receive operations take longer time than the corresponding send operations. In that case the results of the analysis have to be fed back to implementation and the send operation needs to be slowed down. But the application of this feedback does not affect the worst-case timing analysis and has to be done only once per communication operation and architecture.

Furthermore, all timing bound parts assure the worst-case preconditions and previous states (as they are statically analysed). Thus, two equal code chunks that are executed in different processes still exhibit the same static timing upper bound. Moreover, all processes participating in the analysed communication are assumed to begin their execution of the operation in the same point of time. This is a logical assumption to ensure the consideration of the worst case and leads to consequences when applying the analysis of a complete parallel program. As the nodes are typically reaching a collective not exactly at the same time, it must be assumed that a collective starts only at the time when all participants have finished their previous work. Thus, waiting times are assumed in the analysis, that actually may not be present. These assumptions introduce a certain amount of overestimation. However, on the other hand they enable the usage of timing bounds for communication operations as a module. Thus, the calculated timing behaviour can be reused for applications analysed in future.

5.2 Assembling Phase and Formalism of Timing Analysis

Subsequently, we provide the procedure for obtaining the end-to-end WCET upper bound of a MPI operation in a formalism. Thereby, we assume that the analysis of local computation and parallel communication is finished for the formalisation. That means that all WCETs of local computations as well as the admission and transportation times (t^a and t^t) are known

for all flits and processes. Additionally, the communication patterns for the investigated MPI operation are obtained in the separation phase of the analysis. Thus, the remainder of this section focuses on assembling of the revealed results.

There are several kinds of MPI operations where each one causes distinct issues for the assembling phase. The different communications and the corresponding communication patterns applied are described in detail in Section 2.3. Subsequently, we firstly focus on the analysis of point-to-point communication with explicit blocked send and receive operations, followed by the consideration of the characteristics of concurrent send-receive operations (see Section 2.3.1). After regarding the basic point-to-point communications the focus is on collective communications. Thereby, we distinguish between communication based on a central process and communication based on uniform data exchange.

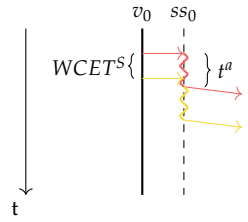
We assume that the WCET upper bound for sending one flit may differ between processes but remains equal for each flit on the same process. Therefore, the WCET upper bound for sending one flit on a process v_i is denoted as $WCET_i^S$. The same assumption holds true for receiving and forwarding a flit and for receiving a flit. Thus, there are the symbols $WCET_i^{RF}$ for receive and forward a flit and $WCET_i^R$ for solely receiving a flit on process v_i. The time that passes between the arrival of two consecutive flits at a process v_i is stated as t_i^r. Please also note that in our formalism we skip the timing of possibly necessary initialisation code that is executed from all processes in the same way to support readability. Of course in our discussion in Section 5.3 these times are considered for the calculation of timing bounds.

5.2.1 Point-to-Point Communication

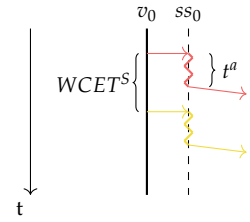
In this section we examine the point-to-point communication in terms of timing. Thereby, this kind of communication typically comprises the operations of simply sending or receiving data and the operation where a send and a receive operation is executed concurrently in a send-receive operation. At first, we consider the timing of transmitting data between two processes by using explicit blocked send and receive operations. Afterwards, our focus is turned to send-receive communication.

Starting with explicit send and receive operations we begin with describing the issues that must be considered locally at each process and combine the timings of sender and receiver afterwards. The borders between computation and communication contain some issues that must be considered to obtain a correct end-to-end WCET upper bound. On the sender side there is a difference if sending is driven by admission time or by local code execution time (see Figures 5.3a and 5.3b). If the code WCET bound between two send operations ($WCET^S$) is smaller or equal to the admission time t^a, the release of the last of x flits to the NoC starts after a duration of $x \cdot t^a + WCET^S$. In contrast, if $WCET^S > t^a$ holds true the release starts after $x \cdot WCET^S + t^a$. Please note that a simple accumulation is possible, as the borders between WCET and WCTT bounds are well defined. Thus, the worst-case release time (W^S) of a number of x flits in a sender process is calculated as follows.

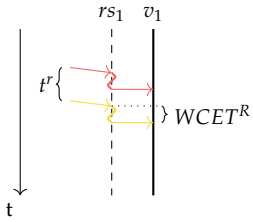
$$
\begin{aligned}
W_i^S(x) &= \begin{cases} x \cdot t_i^a + WCET_i^S & \text{if } WCET_i^S \leq t_i^a \\ x \cdot WCET_i^S + t_i^a & \text{if } WCET_i^S > t_i^a \end{cases} \\
&= (x-1) \cdot \max\left(WCET_i^S, t_i^a\right) + WCET_i^S + t_i^a
\end{aligned} \tag{5.2}
$$



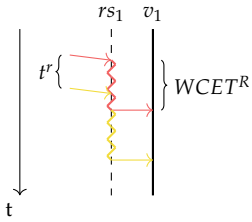
(a) send driven by t^a



(b) send driven by $WCET^S$



(c) receive driven by t^r



(d) receive driven by $WCET^R$

Figure 5.3: Boundaries between WCET and WCTT. v_0 and v_1 mark the execution of code while ss_0 and rs_1 illustrate the time flits stay in send or receive state, respectively. The colours mark different flits.

We further define a symbol for the term that mainly drives the calculation of W^S to facilitate the presentation of calculations in the remainder of this work.

$$t_i^s(x) = x \cdot \max\left(WCET_i^S, t_i^a\right) \tag{5.3}$$

On the receiver side a similar aspect has to be considered (see Figures 5.3c and 5.3d). The timing behaviour directly corresponds to the relation between the WCET estimate of the local execution for receiving a flit ($WCET^R$) and the time passing between the arrival of two flits (t^r). If $WCET^R \leq t^r$ holds true, the time when the last of x flits is received is at $(x-1) \cdot t^r + WCET^R$ after the first flit has arrived. On the other hand, when the time for receiving is driven by the time for local computation ($WCET^R > t^r$) it takes $x \cdot WCET^R$. The corresponding equation is indicated as follows.

$$\begin{aligned} W_i^R(x) &= \begin{cases} (x-1) \cdot t_i^r + WCET_i^R & \text{if } WCET_i^R \leq t_i^r \\ x \cdot WCET_i^R & \text{if } WCET_i^R > t_i^r \end{cases} \\ &= (x-1) \cdot \max\left(t_i^r, WCET_i^R\right) + WCET_i^R \end{aligned} \tag{5.4}$$

Based on Equations 5.2 and 5.4 the end-to-end timing bound of a point-to-point communication of a message with multiple flits between one sender and one receiver process can be calculated. As already mentioned there are some general assumptions that facilitate the calculations. One assumption is that the low-level code executed for sending and receiving is the same for each flit on a process, which in turn implies that the corresponding WCET bound is the same for each flit. Furthermore, it should hold true that the transportation time of a specific hop does not vary for all flits. Thus, the delivery of a flit takes the same amount of time no matter if it is the first, last or any flit in the middle of a message.

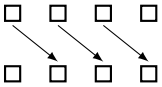
On receiver side the point in time when a message with f flits is completely received is determined by the times t^r and $WCET^R$. Since t^r is completely determined with the timing characteristics of the sending the flits, the timing of the sender process is implicitly included when considering W^R for the entire message ($W^R(f)$). Hence, for a point-to-point communication from process v_i to process v_j the time t_j^r is equal to the time $t_i^s(x = 1)$. Only the duration from beginning of the communication until the arrival of the first flit at the receiver process must be considered in addition. This consideration of the first flit's timing can be obtained by considering W^S with a message of 1 flit ($W^S(1)$) and adding up the needed transportation time $t_{i,j}^t$. Thus, the upper timing bound for point-to-point communications with explicit blocked send and receive operations is calculated as follows.

$$\begin{aligned} W_{i,j}^{P2P}(f = |msg|) &= W_i^S(1) + t_{i,j}^t + W_j^R(f) \\ &\overset{(5.2)}{=} WCET_i^S + t_i^a + t_{i,j}^t + (f-1) \cdot \max\left(t_j^r, WCET_j^R\right) + WCET_j^R \\ &\overset{t_j^r = t_i^s(1)}{=} (f-1) \cdot \max\left(WCET_i^S, WCET_j^R, t_i^a\right) \\ &\quad + WCET_i^S + t_i^a + t_{i,j}^t + WCET_j^R \end{aligned} \tag{5.5}$$

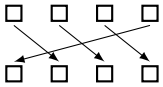
After examining the worst-case timing bound of delivering a message from one process to another process we focus on concurrent send-receive operations subsequently. These operations perform a communication that concurrently executes one send and one receive operation. Thereby, the according communication partners may be the same process for both communications or they are different processes. Since the lengths of the messages sent (msg^s) and received (msg^r) may differ, we provide the symbol $f_i^s = |msg_i^s|$ for the number of flits to send by this operation and $f_i^r = |msg_i^r|$ is the number of flits that must be received by a process v_i. When applying different target processes for sending and receiving it is possible to form rows $cp^{sr,row} \in CP$ or rings $cp^{sr,ring} \in CP$ of processes in a way that each one executes a point-to-point communication. Thereby, each process of a ring or in the middle of a row is executing a send-receive operation and the first and last process of a row are executing a send or a receive operation, respectively. With those communication patterns it is possible to shift data along the row or ring. Figure 5.4 illustrates the communication along a row or ring with point-to-point operations.

The concurrent execution could be easily realized with forking two threads one performing a send and the other performing a receive operation. Moreover, the workload could be assigned to a dedicated hardware like for example a DMA unit. However, we do not restrict the nodes $n \in N$ to provide a special hardware and the concurrent execution of threads may be serialized when there exists only one processing element per node. Since our system model does not require special hardware and our analysis shall not consider operating systems overhead like the one for providing multithreading, we assume that there is only one thread running that performs the communication. Thus, we emulate the concurrent execution by alternating the send and receive operations for each flit. Thereby, the blocked receive operation is only executed when there is actually a flit present in the receive buffer of the NI. Code example 5.2 illustrates the assumed implementation in pseudocode. Please note that we omit the synchronization mechanism, since we do not restrict the approach to a particular one (see Section 2.1.1).

The implementation of the send-receive operation in Code example 5.2 uses the basic



(a) point-to-point communication along a row of processes



(b) point-to-point communication along a ring of processes

Figure 5.4: Possible communication patterns for performing multiple send-receive communications at the same time. The rectangles of one line indicate the processes that participate in the communication. The two lines stand for the status of the communication before and after it is performed. The arrows mark the simple point-to-point communications. In the ring variant all processes perform a concurrent send-receive operation, while in the row variant only the inner processes execute a send-receive operation. The outer processes perform a simple send or a simple receive operation, respectively.

Code example 5.2 Implementation of a send-receive operation in pseudocode. The first loop is executed as long as there remain flits for sending. Thereby, in each iteration on flit is sent and one flit is received if one is present in the receive buffer. The second loop handles the remaining flits of the receive operation that are not handled when the send operation has finished.

```
sendrecv(src, dst) {
        while(!sendMsg(dst) complete) {
                sendFlit(dst, data);
                if(flit in recv buf) {
                        recvFlit(src, data);
                }
        }
        while(!recvMsg(src) complete) {
                recvFlit(src, data);
        }
}
```

operations `sendFlit` that inserts the flit to the local NI and afterwards directly returns. On the other hand the basic `recvFlit` blocks and wait until the appropriate flit has arrived at the NI. Thus, with that implementation it is possible to emulate concurrently executed send and receive operations on complete messages msg^s and msg^r.

Since we do not know when the processes arrive in program execution at an investigated communication, we need to consider all possible cases. Looking at a send-receive communication it is possible that the send and the receive parts of a process happen to completely different times or there is a partial or a complete overlapping of the execution of both parts. With overlapping the time intervals between two receives or two sends of flits are enlarged, compared to separated sending and receiving. Thus, the overlapping delays the receiving of the last flit of a message msg^r as well as the sending of the last flit of a message msg^s. Therefore, we can state that the worst case occurs when sending and receiving the messages is performed concurrently. Hence, we consider that case in the formalism. The principle components of the timing are stated in Figure 5.5.

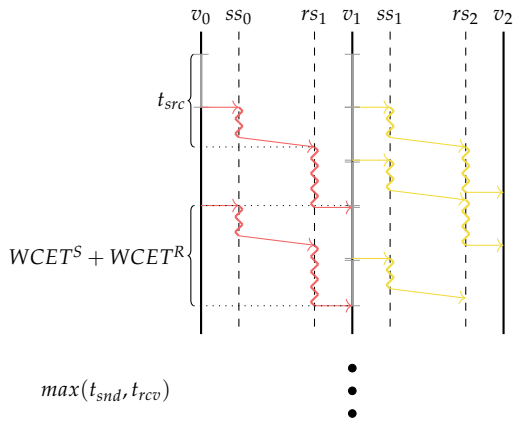


Figure 5.5: The timing characteristics of a send-receive operation. The illustration is in the point of view of the process that is executing the operation. This process is marked with the symbol v_1. The time goes from top to bottom. v_0 to v_2 mark the execution of code while *ss* and *rs* illustrate the time flits stay in send or receive state, respectively. The colours mark different flits. Gray segments illustrate the local computation times for sending or receiving a flit. The sloping lines between two processes mark the traversal of a flit over the NoC and arrows indicate the arrival of a flit at the process's receive buffer.

At first, we must provide several times as building blocks for the final calculation of the end-to-end timing bound of send-receive communications. The first timings to provide are similar to Equation 5.3 and state the times that last between the sending of two flits. Thereby, one has to distinguish between the phase of concurrent sending and receiving and the phase of pure sending. Thus, we state the following equations.

$$WCET_i^{CSR} = WCET_i^S + WCET_i^R \tag{5.6}$$

$$\Delta t_i^{cs} = \max(t_i^a, WCET_i^{CSR}) \tag{5.7}$$

$$\Delta t_i^s = \max(t_i^a, WCET_i^S) \tag{5.8}$$

In Equation 5.6 we define the local computation time that is needed by a process to handle concurrently sending and receiving one flit. The maximum calculations of Equations 5.7 and 5.8 reflect the issues that are described in Figures 5.3a and 5.3b. The difference between both presented equations is the time considered for local computation. Since we need to consider that there is a receive operation executed between two successive sends during the phase of concurrent send and receive, it is necessary to additionally consider the local computation of receiving a flit.

Next to the timing about sending flits there is also the need to obtain the timing for receiving flits. Thereby, the first time interval of interest is the time between the arrival of two successive flits at the receive buffer of a NI, since this time indicates the time when a `recvFlit` operation (see Code example 5.2) can handle that flit at the regarded process. The following equation concretely calculates the time between the $(x-1)$-th and the x-th arrived flit. Please note that we subsequently assume that the processes of a row or ring of

send-receive communications are indexed in ascending order for simplicity.

$$t_i^r(x) = \begin{cases} \Delta t_{i-1}^{cs} & \text{if } x \leq f_{i-1}^{cs} \\ \Delta t_{i-1}^{s} & \text{if } x > f_{i-1}^{cs} \end{cases} \tag{5.9}$$

The time that lasts between the arrival of two flits completely bases on the times when the flits are sent and the time for traversing the NoC. However, since the time for the NoC traversal is the same for all flits, t_i^r is only determined by the time that the sending process exhibits. Thus, it is possible to calculate that time with Equations 5.7 and 5.8. Additionally, we need to refer to f_{i-1}^{cs} of Equation 5.13 (see below) to decide if $t_i^r(x)$ exhibits the timing for concurrent send and receive or for the pure send of the neighboured process v_{i-1}.

Based on Equation 5.9 there are two equations that state the times between the receiving of two successive flits. Thereby, Equation 5.10 targets the timing interval if there are send operations performed concurrently to the considered receives, while Equation 5.11 focuses the case without concurrent send operations.

$$\Delta t_i^{cr}(x) = \max(t_i^r(x), WCET_i^{CSR}) \tag{5.10}$$

$$\Delta t_i^{r}(x) = \max(t_i^r(x), WCET_i^{R}) \tag{5.11}$$

Similar to Equations 5.7 and 5.8 we apply maximum calculations in the Equations 5.10 and 5.11 to refer to the issues presented in Figures 5.3c and 5.3d. Thereby, we use $t_i^r(x)$ to refer to the regarded arrival time and apply $WCET_i^{CSR}$ and $WCET_i^R$ to consider the correct local computations. As with previous equations we again need to distinguish between concurrent send and receive execution and pure receiving for the timing of the local computation part.

Since we are alternating sending and receiving one flit, the local computation between the admission of two flits consists of the timing of one send execution and one receive execution. Hence, as long as there remain flits to send and to receive both times need to be considered. If there are still flits of msg^r to receive after all flits of msg^s have been sent, then this receiving must be considered. On the other hand if sending takes longer than receiving, then the delivery of the last flit of msg^s dominates the timing. To put these characteristics in an equation, we need to calculate the number of flits f_i^{cr} that can be received by the considered process v_i during the period until the last flit of msg_i^s is sent. This calculation is presented in Equation 5.12.

$$f_i^{cr(k+1)} = \min\left(f_i^r, \left\lfloor \frac{f_i^s \cdot WCET_i^S + f_i^{cr(k)} \cdot WCET_i^R}{\Delta t_i^{cr}(1)} \right\rfloor\right), k \in \mathbb{N}_0 \tag{5.12}$$

This number of flits f_i^{cr} can be obtained by considering the time the send part of the operation needs for finishing its local computation ($f_i^s \cdot WCET_i^S + f_i^{cr(k)} \cdot WCET_i^R$) and divide it by the time that lasts for handling one flit with the receive part of the operation $\Delta t_i^{cr}(1)$. Since we are talking about the number of flits in the equation, it makes no sense that there are partial flits within the concurrent execution. Thus, the fraction must be rounded down to an integer. The local computation must regard to the time for sending and receiving flits, since the process alternates in sending and receiving a flit. Since we are looking at the case of concurrent sending and receiving, we need to consider the local computation of the send

operations and the local computations of the receive operations performed in the time until all the flits of a message are sent. These times can be calculated by multiplying the time needed for handling one flit ($WCET_i^S$ or $WCET_i^R$) with the number of flits handled (f_i^s or f_i^{cr}). However, the number of flits received during the phase of concurrent execution of send and receive operations is subject to the current calculation. Thus, we use the mathematical method of a fixed point iteration for the calculation of this value. Thereby, we start with the value $f_i^{cr(0)} = 0$ for $k = 0$ and iterate over ascending values k until there is no difference between the result of $f_i^{cr(k)}$ and $f_i^{cr(k+1)}$. Because of the fixed point operation the value of $f_i^{cr(k)}$ may raise to a larger number than the length of the message received, we upper bound the value of f_i^{cr} to the length of the regarded message by performing a minimum calculation with f_i^r.

Similar to the number of flits that can be received during the period of sending flits, we need to know the number of flits that can be sent before receiving has finished. This number can be different to f_i^s if the sending takes longer than the receiving. Thus, we provide an equation to calculate this number.

$$f_i^{cs} = \min\left(f_i^s, \left\lceil \frac{f_i^{cr} \cdot \Delta t_i^{cr}(1) - f_i^{cr} \cdot WCET_i^R}{WCET_i^S} \right\rceil\right) \tag{5.13}$$

Equation 5.13 is based on the number of flits that can be received before sending is completed (see Equation 5.12). We use this value for calculating the time that is remaining for performing send operations during the phase of concurrent handling of flits sent and received. For that we calculate the time until the last of the concurrently handled flits is received ($f_i^{cr} \cdot \Delta t_i^{cr}(1)$) and subtract the time the process is actively performing receive operations ($f_i^{cr} \cdot WCET_i^R$). The remaining time can be used for performing send operations. Thus, if this time is divided by the time that is needed to send one flit, we get the number of flits that are sent during the phase of concurrent sending and receiving. Similar to Equation 5.12 it makes no sense to provide partial flits. Thus, we round up the result of the fraction to the next integer. The rounding is performed upwards, since we always see the execution of a send operation as the last operation of the phase of concurrent handling of sending and receiving.

In addition to the previously presented calculations we distinguish if sending or receiving of the according messages (msg^s or msg^r) takes longer. This decides if the last flit of a message is handled during the concurrent execution or afterwards. According to the result there are different local computation bounds to regard. Therefore, we provide the following equation that reflect that issue.

$$WCET_i^{SR,r} = \begin{cases} WCET_i^R & \text{if } f_i^r > f_i^{cr} \\ WCET_i^{CSR} & \text{if } f_i^r \leq f_i^{cr} \end{cases} \tag{5.14}$$

As already mentioned the send-receive operation are possibly combined with pure send or receive operations that are executed at the partner processes (cf. Figure 5.4a). Since we have to refer to the timing of the communication partners in some situations, we need to regard to the different timing according to its executed operation. Thus, the following

equation reflects that issue for the time to locally perform a send operation for one flit.

$$W_i^{snd} = \begin{cases} WCET_i^S + t_i^a & \text{if } v_i \text{ executes } \textit{send} \text{ operation} \\ WCET_i^{CSR} + t_i^a & \text{if } v_i \text{ executes } \textit{send-recv} \text{ operation} \end{cases} \tag{5.15}$$

A send-receive operation can logically be split in a sending and a receiving part. With the regarded implementation of alternating sending and receiving flits this holds true for the timing as well. Hence, we provide the timing of each part in a separate equation.

$$W_i^{SR,s} = WCET_i^{CSR} + (f_i^{cs} - 1) \cdot \Delta t_i^{cs} + (f_i^s - f_i^{cs}) \cdot \Delta t_i^s \tag{5.16}$$

In Equation 5.16 the timing bound for sending the flits of a send-receive operation is presented. For the first flit always the time for local computation ($WCET_i^{CSR}$) must be considered. The next part of the equation ($(f_i^{cs} - 1) \cdot \Delta t_i^{cs}$) focuses on the timing of the flits sent concurrently to executed receive operations except for the already considered first flit. Afterwards the remaining flits are handled with considering pure sending of flits ($(f_i^s - f_i^{cs}) \cdot \Delta t_i^s$).

$$W_i^{SR,r} = f_i^{cr} \cdot \Delta t_i^{cr}(x) + (f_i^r - 1 - f_i^{cr}) \cdot \Delta t_i^r(x) + WCET_i^{SR,r} \tag{5.17}$$

Equation 5.17 provides the timing bound for receiving the flits of a send-receive operation. Thereby, this timing bound starts at the point in time when the first flit arrives and the regarded process v_i. Thus, at first we consider all flits that are handled during concurrent execution of the send and receive operations. Afterwards, the timing of the remaining flits except the last one are considered. This last flit always is addressed with the timing for the local computation $WCET_i^{SR,r}$. Hence, it is considered separately in the equation.

After regarding both parts of a send-receive operation it is possible to provide the timing of the operation for a process v_i.

$$W_i^{SR} = \max(W_i^{SR,s}, W_{i-1}^{snd} + t_{i-1,i}^t + W_i^{SR,r}) \tag{5.18}$$

Equation 5.18 obtains the timing upper bound of send-receive operation by calculating the maximum of the timing for the send and receive part. At the receive part we need to consider the time until the first flit arrives at the regarded process v_i and accumulate the time for receiving all flits ($W_i^{SR,r}$). Thereby, the time for receiving the first flit is determined by the time for sending it at the source process (W_{i-1}^{snd}) and the transportation time over the NoC ($t_{i-1,i}^t$).

Typically, send-receive operations form a row or a ring of processes that shift data along its neighboured processes (see Figure 5.4). Subsequently, we define such a structure with a list of processes $\langle v_m, \ldots, v_n \rangle$. Thereby, in a row the first and last process (v_m and v_n) execute a send or a receive operation, respectively. In contrast, in a ring structure the first and last processes of the list perform a send-receive operation like all other processes in a way that the they connect to each other and hence, forming a ring. Therefore, one such row or ring of processes must be seen as a single complex communication. The according worst-case

timing of such communication operations is presented in the following equation.

$$W^{sndrcv}_{\langle v_m,\ldots,v_n\rangle} = \begin{cases} \max\limits_{\forall i:v_i\in\langle v_m,\ldots,v_n\rangle}(W^{SR}_i) & \text{if } \langle v_m,\ldots,v_n\rangle \in CP^{sr,ring} \\ \max\limits_{\forall i:v_i\in\langle v_{m+1},\ldots,v_{n-1}\rangle}(W^{S}_m, W^{SR}_i, & \\ \qquad W^{snd}_{n-1} + t^t_{n-1,n} + W^R_n) & \text{if } \langle v_m,\ldots,v_n\rangle \in CP^{sr,row} \end{cases} \tag{5.19}$$

Equation 5.19 calculates the maximum of the send-receive operation's timing bounds of all processes of the row or ring pattern. Please note that there is the timing of a simple send or receive operation used for the first and last process of a row.

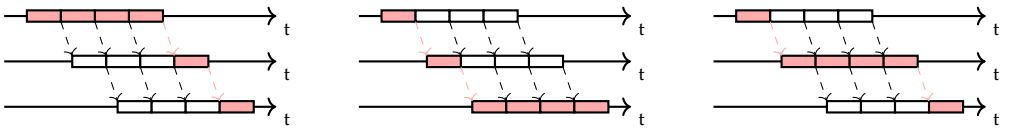
5.2.2 Collective Communication

Collective communication operations target the exchange of data among a group of 2 or more processes. As stated in Section 2.3, there are several different operations available that can be classified in two classes. One class provides communication that uses a central process to collect or spread data from or across all other processes participating in the communication. The other class of operations performs communications that do not have one dedicated process. Rather, all processes behave similar in terms of the data sent and received. Subsequently, we firstly focus on the class of operations with a central process and regard to the operations performing an uniform data exchange afterwards.

Collective Operations based on a Central Process

As stated in Section 2.3 collective communication that is based on a central process typically is realized by forming tree-like communication patterns CP^{tree} (see Figure 2.13). These patterns are build with multiple point-to-point communications based on blocked send and receive operations. Hence, the overall timing of the communications described here directly bases on the timing of the point-to-point communication with single send and receive operations. Subsequently, we present our formalism for calculating end-to-end timing bounds for tree-like communication pattern by firstly providing the simple cases and stepwise extend them to the more complex patterns afterwards. As a first step in calculating the timing of the appropriate communication patterns $cp \in CP^{tree}$ we obtain the timing of a message delivery consisting of multiple flits along a chain like row of nodes (see Figure 5.6). There are several possibilities to accumulate the times of transporting single flits to obtain an end-to-end delivery time. Thus, it needs to be examined which logical path through the message delivery is the longest in terms of time.

There are some general assumptions that facilitate the identification of this path. One assumption is that the low-level code executed for sending and receiving is the same for each flit on a process, which in turn implies that the corresponding WCET bound is the same for each flit. Please note that we provide this timing bound without considering a particular processor or execution state. Thus, we assume the state of the worst case. Further, it should hold true that the transportation time of a specific hop does not vary for all flits. Thus, the delivery of a flit takes the same amount of time no matter if it is the first, last or any flit in the middle of a message.



(a) Path considering the WCET of send operations is the longest one

(b) Path considering the WCET of receive operations is the longest one

(c) Path considering the WCET of send and forward executions of an intermediate node is the longest one

Figure 5.6: Delivery of four flits along a row of three processes. Thereby, each line represents the timing characteristics of one process. The rectangles indicate local code execution for sending or receiving one flit, respectively and the arrows mark flit migration across the NoC. Each figure displays a different logical path considered as critical for timing calculation.

With those assumptions the number of paths which possibly take longest is bound to the number of processes participating in the chain like communication. For one of those paths it is possible to accumulate all times that the root process needs for sending the flits appended by the time the last flit of the message needs for the delivery to the last process of the row including all traversal, sending and receiving times (see Figure 5.6a). In contrast, it is also possible that receiving takes longer than sending. That case is displayed in Figure 5.6b. It causes the summation of all delivery times (from the first to the last process) of the first flit complemented with the receiving times of all remaining flits on the last process. When a process not only has to receive a message but also forward it, its code WCET bound may differ from a simple addition of send and receive WCETs. Therefore, the remaining options focus on the case that one of the intermediate processes is responsible for the maximum timing bound. It is shown in Figure 5.6c and the timing of the first flit is considered until it reaches the designated intermediate process between the first and the last process. Afterwards the receiving and forwarding of all remaining flits is taken into account and finally the delivery of the last flit to the last process is accumulated. In all examined paths the issues for the boundaries between WCET and WCTT like described in Figure 5.3 must be considered. These issues are already addressed in Equations 5.2 and 5.4 and the according timing calculations W^S and W^R are reused for the current calculations. Thereby, t^r is driven by the times for sending the flits of the previous process, as the transportation times are the same for all flits.

The calculation of the timing bound for delivering a message along a row of processes requires the consideration of the timing of processes that receive data from other processes and forwards them afterwards. We refer to the timing of such intermediate processes v_i with the symbol W_i^{RF}. Looking ahead to more complicated communication patterns than a row of processes there may be a different number of flits received compared to the number forwarded. Thus, the local computation of an intermediate process is split in two parts: the WCET estimation for receiving a flit $WCET_i^{RF}$ and the WCET upper bound for sending a flit $WCET_i^{SF}$ at a process v_i. Please note that we choose different symbols than for solely sending or receiving flits ($WCET^S$ or $WCET^R$) to indicate that the timing bounds are part of a local receive and forward operation. Because of that difference their timing may also

differ from a single basic send or receive operation. For both parts the issue of Figure 5.3 is considered and lead to the following equation for the timing of receive and forwarding flits (x incoming and y outgoing flits) at a process v_i.

$$W_i^{RF}(x,y) = (x-1)\cdot \max\left(t_i^r, WCET_i^{RF}\right) + WCET_i^{RF} \\ +(y-1)\cdot \max\left(WCET_i^{SF}, t_i^a\right) + WCET_i^{SF} + t_i^a \tag{5.20}$$

Now we are able to investigate the delivery of one flit from a process v_m along a chain like row of l processes until it reaches a process v_n. Thereby, we state the number of participating processes l as follows.

$$l = |[v_m, v_n]| \tag{5.21}$$

Furthermore, we assume that the indices of the processes ascent along the considered row of processes for simplicity. The calculation of this timing bound ($W^{row}(f=1)$) is a preparing consideration for the issue discussed in Figure 5.6, which in contrast concentrates on the delivery of multiple flits along the same pattern of nodes. The principle procedure is to follow the way covered by the flit transport and accumulate all times arising. A simple accumulation is possible, as the borders between WCET and WCTT bounds are well defined. The timing bounds include the ones caused by nodes (W_i^S, W_i^{RF}, W_i^R) and the transportation times $t_{i,j}^t$ for all traversed communication edges $e_{i,j}^c$. Following the described principle the end-to-end WCET bound of delivering one flit along a row of multiple processes is calculated as follows.

$$W^{row}(f=1) = W_m^S(x=1) + t_{m,m+1}^t + \sum_{i=m+1}^{n-1}\left[W_i^{RF}(x=1,y=1) + t_{i,i+1}^t\right] \\ +W_n^R(x=1) \tag{5.22}$$

Based on Equation 5.22 it is possible to calculate the delivery of messages comprising more than one flit along a row. The issue described in Figure 5.6 mainly drives the timing of this communication. Therefore, the maximum of all described paths must be calculated. There are l different paths possible to be the longest one. They differ in the timing each process needs to perform the communication. The subsequent equations (Equations 5.23 to 5.27) assume that we send a message msg comprising of $f = |msg|$ flits. The timing of the first path (sending all flits at the first process and tracking the last flit, cf. Figure 5.6a) is bounded as follows.

$$W^a(f) = W_m^S(x=f) + t_{m,m+1}^t + \sum_{i=m+1}^{n-1}\left[W_i^{RF}(x=1,y=1) + t_{i,i+1}^t\right] \\ +W_n^R(x=1) \tag{5.23}$$

In the case when receiving takes longest, the delivery of the first flit is tracked and afterwards the time for receiving all flits at the last process is considered (cf. Figure 5.6b).

$$W^b(f) = W_m^S(x=1) + t_{m,m+1}^t + \sum_{i=m+1}^{n-1}\left[W_i^{RF}(x=1,y=1) + t_{i,i+1}^t\right] + W_n^R(x=f) \tag{5.24}$$

The remaining $l-2$ paths track all flits at a certain intermediate process. Thus, in each of those paths the first flit is followed until that particular process is reached and the last flit is

used to go along the rest of the path to the last process (cf. Figure 5.6c). When considering process v_j as the dedicated process, the timing is indicated in the following equation.

$$\begin{aligned} W_j^c(f) &= W_m^S(x=1) + t_{m,m+1}^t + \sum_{i=m+1}^{j-1} [W_i^{RF}(x=1,y=1) + t_{i,i+1}^t] \\ &\quad + W_j^{RF}(x=f,y=f) + t_{j,j+1}^t \\ &\quad + \sum_{i=j+1}^{n-1} [W_i^{RF}(x=1,y=1) + t_{i,i+1}^t] + W_n^R(x=1) \qquad (5.25) \\ &= W_m^S(x=1) + t_{m,m+1}^t + W_j^{RF}(x=f,y=f) + t_{j,j+1}^t + W_n^R(x=1) \\ &\quad + \sum_{\substack{i=m+1 \\ i\neq j}}^{n-1} [W_i^{RF}(x=1,y=1) + t_{i,i+1}^t] \qquad (5.26) \end{aligned}$$

Since we look for the maximum timing bound, the end-to-end WCET upper bound of a delivery of f flits along a row of l processes (from process v_m to process v_n) is displayed in Equation 5.27.

$$W^{row}(f) = \max_{i\in[m+1,n-1]} \{W^a(f), W^b(f), W_i^c(f)\} \qquad (5.27)$$

Next, we extend our formalism to support arbitrary tree-like patterns $cp \in CP^{tree}$ to allow the calculation of the timing bounds of the communication patterns used in MPI collective communication that is based on a central process. In the case that core local WCETs, admission times t^a or transportation times t^t vary for different processes or communication edges, all possible pattern paths of the communication pattern must be investigated to obtain the maximal pattern path in terms of timing. Thus, we need to extend the timing bound W^{row} to a timing bound for trees W^{tree}. Depending on the intended communication operation the communication patterns are built with two different communication directions. In one direction the data are delivered from the root process to the leaf processes (e.g. for broadcast). The other direction begins in the leaf processes and finishes in the root processes (e.g. with gather). For readability we mark in some situations symbols that refer to an aspect for communication from root to leaf with a sign $\downarrow$ and use $\uparrow$ for the opposite direction. At first, we focus on the timing of communications that go from root to leaf processes.

For the timing calculations on such patterns we build on the timing obtained for single processes. The corresponding calculations are given in Equations 5.2, 5.4 and 5.20. However, Equations 5.4 and 5.20 include the time passing between two incoming flits t_i^r, which directly corresponds to the exhibited timing of previous processes in the communication pattern $cp \in CP$. Thus, it is possible to express t_i^r in an equation that relies on the timing of the previous processes. However, since we consider processes that send flits to multiple other processes, we need to encode the correct local computation WCET bound according to the type of a regarded process v_i (root, intermediate of leaf process) first. Thus, we extend Equation 5.3 to fit for all types of processes v_i.

$$t_i^{se} = \begin{cases} |V_i^{cp,out}| \cdot \max(WCET_i^S, t_i^a) & \text{if } |V_i^{cp,in}| = 0 \\ |V_i^{cp,out}| \cdot \max(WCET_i^{SF}, t_i^a) & \text{if } |V_i^{cp,in}| > 0 \end{cases} \qquad (5.28)$$

With the considered tree patterns $cp \in CP_{\downarrow}^{tree}$ it is possible that one flit of a message is sent multiple times, each time to a different node. Therefore, a function is needed, that

encodes the total number of flits sent until a message's last flit. The function is based on the number of already handled data flits from a message msg and the degree of outgoing communication edges of a regarded process v_j. The number of flits that are already sent from a process v_j when it is about to send the last flit of message msg to its child connected via a communication edge e^c is given as follows.

$$f_j^{s,pre}(msg, e^c) = e^c.o^c + \sum_{v_i \in V_j^{cp,out}} snt_j^{cp}(msg, v_i) \tag{5.29}$$

In Equations 5.29 the symbol $e^c.o^c$ represents the position of the communication edge e^c. Thus, there are $e^c.o^c - 1$ edges served with a flit before the current edge is served with that flit. Furthermore, the function snt (see Definition 2.46) is used to provide the number of flits already sent to a process v_i when we are about to send the last flit of message msg. If all outgoing communication edges handle the same number of flits, Equations 5.29 can be simplified as follows.

$$f_j^{s,pre}(msg, e^c) = e^c.o^c + |V_j^{cp,out}| \cdot snt_j^{cp}(msg, v_i) \tag{5.30}$$

Now, we extend Equations 5.23, 5.24 and 5.26 to fit for a pattern path pp of a given communication pattern $cp \in CP_{\downarrow}^{tree}$. Subsequently, we assume for simplicity that the processes of a particular pattern path $pp \in PP^{tree}$ exhibit ascending indices and the path begins at process v_m and ends with process v_n.

Similar to the illustration in Figure 5.6a we firstly consider the case when the local timing of the root process dominates the overall timing bound of a pattern path. For that the timing of all flits sent from the root process is regarded in the following equation and afterwards the timing of the last flit is considered for traversing the rest of the regarded pattern path pp.

$$\begin{aligned} W_{\downarrow}^{pp,a}(msg) = {} & W_m^S(x = f_m^{s,pre}(msg, pp.e_m^c)) + t_{m,m+1}^t \\ & + \sum_{i=m+1}^{n-1} [W_i^{RF}(x = 1, y = e_{i,i+1}^c.o^c) + t_{i,i+1}^t] + W_n^R(x = 1) \end{aligned} \tag{5.31}$$

Since each data flit of the regarded message msg is sent to multiple children of the root process we use $f^{s,pre}$ of Equation 5.29 and 5.30 to obtain the correct number of flits sent. The concrete child that belongs to the regarded pattern path is given by $pp.e_m^c$. After all flits are handled at the root process, the timing of the last flit is considered for the rest of the pattern path to the leaf process. Thereby, we need to consider the child that belongs to the regarded pattern path for each intermediate process. Therefore, the parameter y is set to $e^c.o^c$ for each time $W^{RF}(x, y)$. The parameter x is set to 1, since each process only receives data from one parent process in the regarded patterns.

If the receiver process takes longest for handling a flit, the first flit of the message is followed all the way along the pattern path and then all remaining flits are considered at the according leaf process. This case is targeted in the following equation.

$$\begin{aligned} W_{\downarrow}^{pp,b}(msg) = {} & W_m^S(x = pp.e_m^c.o^c) + t_{m,m+1}^t \\ & + \sum_{i=m+1}^{n-1} [W_i^{RF}(x = 1, y = e_{i,i+1}^c.o^c) + t_{i,i+1}^t] + W_n^R(x = |msg_n|) \end{aligned} \tag{5.32}$$

In contrast to Equation 5.31 the parameter x for the root process's time W_m^S is $e^c.o^c$ to provide the correct child process for the first flit. Furthermore, W^R does not only handle one flit, but as many flits as contained in the regarded received message ($f = |msg_n|$). Please note that the length of the received message msg_n may differ compared to the length of the message sent at the root process msg (cf. a scatter communication for example).

The third part of the calculation focusses on the case that the timing bound for receiving and forwarding a flit at an intermediate process takes longest. Thus, we extend Equation 5.26 to fit for any intermediate process of a pattern path of a communication pattern $cp \in CP_{\downarrow}^{tree}$. Please note that we refer to the message or the part of a message which is received and handled by a process v_j with the symbol msg_j.

$$\begin{aligned} W_{\downarrow,j}^{pp,c}(msg) = {} & W_m^S(x = pp.e_m^c.o^c) + t_{m,m+1}^t \\ & + \sum_{i=m+1}^{j-1} [W_i^{RF}(x = 1, y = pp.e_{i,i+1}^c.o^c) + t_{i,i+1}^t] \\ & + W_j^{RF}(x = |msg_j|, y = f_j^{s,pre}(msg_j, pp.e_{j,j+1}^c)) + t_{j,j+1}^t \\ & + \sum_{i=j+1}^{n-1} [W_i^{RF}(x = 1, y = pp.e_{i,i+1}^c.o^c) + t_{i,i+1}^t] + W_n^R(x = 1) \qquad (5.33) \\ = {} & W_m^S(x = pp.e_m^c.o^c) + t_{m,m+1}^t + W_n^R(x = 1) \\ & + W_j^{RF}(x = |msg_j|, y = f_j^{s,pre}(msg_j, pp.e_{j,j+1}^c)) + t_{j,j+1}^t \\ & + \sum_{\substack{i=m+1 \\ i \neq j}}^{n-1} [W_i^{RF}(x = 1, y = pp.e_{i,i+1}^c.o^c) + t_{i,i+1}^t] \qquad (5.34) \end{aligned}$$

In Equation 5.34 the regarded process v_j provides a timing bound W_j^{RF} that considers the receipt of all flits of the message msg_j and the forwarding of them accordingly to all children of the process. In contrast, all other processes that belong to the regarded pattern path provide a local timing bound W^S, W^{RF}, or W^R that considers the receiving of one flit and the forwarding of that flit to the correct child and all previously served children of that process.

After calculating the timing of the cases that are able to exhibit the highest timing bound, the timing upper bound of the complete pattern path pp is determined by calculating the maximum time of all considered cases.

$$W_{\downarrow}^{pp}(msg) = \max_{i \in [m+1, n-1]} \{W_{\downarrow}^{pp,a}(msg), W_{\downarrow}^{pp,b}(msg), W_{\downarrow,i}^{pp,c}(msg)\} \qquad (5.35)$$

Finally, the worst-case end-to-end timing bound of the regarded communication pattern $cp \in CP_{\downarrow}^{tree}$ is exhibited by considering all possible pattern paths $pp \in PP^{cp}$ of that pattern and determining the maximal timing bound of all pattern paths.

$$W_{\downarrow}^{tree,cp}(msg) = \max_{pp \in PP^{cp}} \{W^{pp}(msg)\} \qquad (5.36)$$

After examining communication patterns that deliver data from the root process to the leaf processes, we subsequently focus on communication patterns that start at the leaf processes and finish their delivery at the root process (cf. gather communication). With such patterns a process may receive flits from different processes and each of those processes is sending a

certain number of flits. Basically, there are two options to implement such a communication. The first option is that each process receives the data from its parents one after another. This allows each process to receive data only from one sender process at a time. The other option is to allow sending of data from each parent process at the same time. Though this option would probably show a more optimized performance, it would also cause an additional synchronisation mechanism that must be implemented in hardware to prevent conflicts in some of the TDM schedules[1] which are caused by receiving flits from multiple processes in the same TDM cycle. Since we see the development of such a synchronisation mechanism as out of scope of this thesis, we concentrate on the analysis of the first option.

The general procedure of the analysis of communication patterns that forms a tree starting at the leaf processes and finishing at the root process is very similar to the analysis for tree patterns with an opposite direction. We still need to regard to the issues presented in Figures 5.3 and 5.6. Furthermore, similar to broadcast like communication patterns ($cp \in CP_{\downarrow}^{tree}$) the concept of pattern paths is applied to consider each possible path of a pattern $cp \in CP_{\uparrow}^{tree}$. Thus, we are searching for the pattern path with the highest timing bound. When looking at a concrete pattern path pp we always consider the three cases illustrated in Figure 5.6. Thereby, we refer to case *a* if the pure sending of data dominates the timing, case *b* if pure receiving takes longest and case *c* assumes that the receive and forward of an intermediate process is the dominating factor. According to these cases a flit is followed along the pattern path and at a dedicated process v_i all flits of the message are taken into account. For each investigated case the handling of this dedicated process is assumed as the dominating factor of the communication. Thus, we assume the delivery of the regarded flit as not delayed by other parts of the communication before it arrives at that particular process and in addition consider the timing of all flits that must be handled after the arrival of the current flit at the regarded process. Moreover, since there is an order for the receiving of data at a process, all flits from the subsequently handled processes must be regarded for all processes that are located on the current pattern path pp between the process v_i and the root process. Please note that we must correspond to delayed messages and flits not only for one process but for all processes, because those messages originate from different pattern paths and meet the currently investigated pattern path at different processes. However, we can assume the delivery from the leaf process the dedicated process v_i as non-delayed, because any kind of delay would be regarded by the consideration of an other pattern path or the consideration of another case *a*, *b* or *c*.

For the subsequent equations we use the symbol msg in two different ways. The symbol in combination with a subscripted index (msg_i) denotes the message or part of the message that is handled by a process v_i. On the other hand the symbol with a superscripted index (msg^i) is used when considering a set of parent processes of the currently regarded process v_j. Thereby, the symbol refers to the message of the parent process that is the i-th parent to be handled by process v_j. Hence, if the communication edge $e_{i,j}^c$ connects both processes, its order is i ($e_{i,j}^c.o^c = i$). To facilitate the presentation of the following formalism we provide the number of flits that remain for receiving by a specific process v_i in the communication pattern after a particular communication edge $e_{j,i}^c$ is served. We refer to this number with the symbol

[1] especially for the One-to-One, All-to-One, Alternate and Triplet schedules

$f_i^{r,post}(e^c)$ where i indicates the regarded process v_i and e^c stands for a communication edge that points to the process v_i.

$$f_i^{r,post}(e^c) = \sum_{j=e^c.o^c+1}^{|V_{e^c.v_{src}}^{cp,in}|} |msg^j| \tag{5.37}$$

Equation 5.37 iterates over all incomming communication edges of process v_i that are located after the currently regarded edge e^c, which means that for each edge $e_{k,i}^c$ it holds $e^c.o^c < e_{k,i}^c.o^c$. Thereby, it accumulates the number of flits to receive $|msg_j|$ from each of the considered communication edges.

Similar to the communication patterns $CP_{\downarrow}^{tree}$ we base on the Equations 5.23 to 5.26 refer to the different cases of each pattern path pp. Furthermore, we assume for simplicity that the processes of a particular pattern path $pp \in PP^{tree}$ exhibit ascending indices and the path begins at process v_m and ends with process v_n.

We firstly consider the case that pure sending at a leaf process dominates the timing of the examined pattern path. According to the procedure described above the following equation is derived for that case.

$$\begin{aligned} W_{\uparrow}^{pp,a}(msg) = {} & W_m^S(x = |msg_m|) + t_{m,m+1}^t \\ & + \sum_{i=m+1}^{n-1} \Bigg[W_i^{RF}(x = 1, y = 1) \\ & \quad + \sum_{j=e_{i-1,i}^c.o^c+1}^{|V_i^{cp,in}|} \Big(W_i^{RF}(x = |msg^j|, y = |msg^j|) \Big) + t_{i,i+1}^t \Bigg] \\ & + W_n^R(x = 1 + f_n^{r,post}(pp.e_{n-1,n}^c)) \end{aligned} \tag{5.38}$$

The first line of Equation 5.38 considers all flits that are sent from the leaf process v_m of the regarded pattern path pp and also regards to the transportation time $t_{m,m+1}^t$ of the last flit of msg_m to the next process in pp. In the next two lines the sum over i accumulates the timing bounds of all intermediate processes of pp. Since we see in this case the leaf process as dominating factor, all flits that can only be handled after processing the flits from that leaf process must be regarded as well. The reason is that the handling of those flits is delayed until all flits of msg_m are handled. Therefore, the processing of each intermediate process is two-parted. One part is the receiving of the last flit from the previous process and the time for sending it to the next process (see line two) and the other part is the processing of all flits that are delayed by the previously handled flits. This second part is stated in line three where we iterate over all communication edges with an higher order $e^c.o^c$ than the one of the currently regarded pattern path (see sum over index j). For each of the considered edges e^c the timing bounds for the handling of the corresponding messages is accumulated and finally the transportation time of the last of all processed flits to the next process of the pattern path is taken into account. The last line of the equation cares about the receiving of the pattern path's last flit at the root process v_n as well as all flits from other pattern paths delayed by the current one.

After regarding to pure sending at a leaf process as dominating factor, we take a look at the situation when receiving at the root process dominates. This situation is described in the

following equation.

$$
\begin{aligned}
W_{\uparrow}^{pp,b}(msg) = {} & W_m^S(x=1) + t_{m,m+1}^t \\
& + \sum_{i=m+1}^{n-1} \left[W_i^{RF}(x=1,y=1) + t_{i,i+1}^t \right] \\
& + W_n^R(x = |msg_{n-1}| + f_i^{r,post}(pp.e_{n-1,n}^c))
\end{aligned} \tag{5.39}
$$

Equation 5.39 regards to the first flit sent at the leaf process v_m and follows it along the pattern path without considering any delay until it reaches the root process v_n. At the root process the receiving of all flits that are handled after the processing of the regraded flit are considered for receiving. In this equation the first line considers the send procedure at the leaf process, the second line accumulates the timing of all intermediate processes and the last line is responsible for the receiving at the root process. There is no delay considered for the traversal along the pattern path because such delays are regarded by other cases of this pattern path or by other pattern paths.

The last case focuses on an intermediate process as the dominating process. It is based on Equation 5.26 and regards to the dominating process as v_j. Its calculation is presented in the following equation.

$$
\begin{aligned}
W_{\uparrow,j}^{pp,c}(msg) = {} & W_m^S(x=1) + t_{m,m+1}^t \\
& + \sum_{i=m+1}^{j-1} [W_i^{RF}(x=1,y=1) + t_{i,i+1}^t] \\
& + \sum_{k=e_{j-1,j}^c.o^c}^{|V_j^{cp,in}|} \left(W_j^{RF}(x=|msg^k|, y=|msg^k|) \right) + t_{j,j+1}^t \\
& + \sum_{i=j+1}^{n-1} \Bigg[W_i^{RF}(x=1,y=1) \\
& \quad + \sum_{k=e_{i-1,i}^c.o^c+1}^{|V_i^{cp,in}|} \left(W_i^{RF}(x=|msg^k|, y=|msg^k|) \right) + t_{i,i+1}^t \Bigg] \\
& + W_n^R(x = 1 + f_n^{r,post}(pp.e_{n-1,n}^c))
\end{aligned} \tag{5.40}
$$

In Equation 5.40 the first flit of a message is followed along the regarded pattern path until it reaches the process that is considered as the dominating process (see lines one and two). For this process in line three all flits that must be handled for the pattern path as well as all flits of subsequent handled messages are considered. The remaining pattern paths is followed to the root process and in each passed process all flits that are handled after the flits originated in the regared pattern path are considered for the worst-case timing bound. This is done in lines four and five and is regarded in the same way as in Equation 5.38[2]. The timing of the root process considers all flits that are delivered along the regarded pattern path and the ones that are delayed by the execution of this path.

At that point the timing bounds of the cases that are able to exhibit the highest timing bound of a pattern path are calculated. Next, the timing upper bound of the complete

[2]Please note the different start values for k in line three and five of Equation 5.40

pattern path pp is determined by calculating the maximum time of all considered cases.

$$W_{\uparrow}^{pp}(msg) = \max_{i \in [m+1, n-1]} \{W_{\uparrow}^{pp,a}(msg), W_{\uparrow}^{pp,b}(msg), W_{\uparrow,i}^{pp,c}(msg)\} \tag{5.41}$$

The total worst-case timing upper bound is the highest bound of all examined pattern paths of the communication pattern. Hence, the maximum of all pattern paths' timing bounds must be calculated to reveal the end-to-end timing upper bound of the communication pattern.

$$W_{\uparrow}^{tree,cp}(msg) = \max_{pp \in PP^{cp}} \{W_{\uparrow}^{pp}(msg)\} \tag{5.42}$$

Collective Communication based on Uniform Data Exchange

The communication patterns used to execute a collective communication that is based on uniform data exchange is presented in Section 2.3. These patterns are dominated by performing multiple send-receive operations. Thereby, the patterns can be subdivided in multiple steps and in each step all participants execute a send-receive operation. The main differences between the patterns are their number of performed communication steps and the communication partners each process has in a send-receive communication.

Thus, for the calculation of the timing upper bounds one has to accumulate the timing bounds of the send-receive operations performed in each step. Therefore, we can directly build on the procedure presented for calculating the timing upper bound of a single send-receive operation (see Equation 5.19). We need to consider as many send-receive communications as there are steps in the communication pattern and additionally consider the timing for combining the steps. Hence the end-to-end timing of collective communication with uniform data exchange can be calculated as follows.

$$W^{uni}(steps) = \sum_{i=0}^{steps} (W_i^{sendrecv} + WCET^{local}) \tag{5.43}$$

Please note that main parts of the formalism are already presented in Section 5.2.1 where we defined the calculation of the timing upper bound of the concurrent send-receive communication.

5.3 Evaluation

The communication patterns of the MPI communication are investigated in detail with respect to their real-time behaviour. For collective communication a representative is selected for each class of patterns presented in Section 2.3 to enable the examination of the patterns' concrete timing. MPI_Broadcast is chosen for tree like communication patterns. This collective sends data from one node to each other participating node in a way that all nodes hold the same data at the end of the communication. The communication patterns forming uniform data exchanges are represented by MPI_Allgather. This collective logically collects data from all participating nodes and makes the complete set of data available to all nodes.

As the evaluation needs to regard a specific hardware architecture, an implementation must be chosen that fulfils the requirements of Sections 2.1 and 2.2. In this work we consider

the *reduced complexity many-core (RC/MC)* architecture [Mis+17] as hardware platform for timing analysis. It is an homogeneous many-core architecture consisting of RISC-V cores where each core has a separate private scratchpad memory. The cores are connected via a NoC forming an unidirectional two-dimensional torus. Communication through this NoC is done via explicit message-passing from core to core without any additional techniques such as DMA involved. Thereby, each flit is treated as a separate message that contains 8 bytes of data. The sending and receiving is executed by calling specific assembler instructions which are responsible for sending or receiving one flit, respectively. The NI mainly consists of one send and one receive buffer, which both use a first in first out policy.

The architecture facilitates the usage of TDMA and is developed to be timing-analysable. Thereby, the tool OTAWA [Bal+11], which supports WCET analysis on function level, identifies the timing of the collective's core-local executed code and the timing behaviour of TDM which is described in Chapter 4 is applied for calculating the WCTTs. Afterwards, the local code WCET and the timing bounds for the flit traversal are combined according to the formalism presented in Section 5.2.

The MPI operations are ported from OpenMPI [Gab+04] to run on RC/MC. Thereby, the different variants described in Section 2.3 are provided. The ported communication operations utilize the instruction set extension of RC/MC [Mis+17] and hence, each flit is sent as a separate message.

The platform presented in [Mis+17] does not assume a specific hardware flow control. However, without any synchronisation the receive buffers can overflow as described in Section 2.1.1 (see Figure 2.3). Hence, a mechanism is needed to prevent such behaviour. For the remainder of this section we assume an synchronisation mechanism that is similar to the one of Walter [Wal19] which is based on the ready synchronisation of Frieb [Fri+18; Fri19]. With that mechanism the receiver node must indicate the sender node that it is ready to receive data before data can be send. For this purpose the receiver sends one special synchronisation flit to the sender confirming its readiness for receiving data. Afterwards, all flits are sent and there is no need for further acknowledges, as both communicating nodes exactly know the message's length.

As Walter states there is a need for an additional synchronisation NoC for the delivery of the synchronisation flits. This is because in One-To-One, All-To-One and Triplet TDM schedules it must be ensured that within one period only one flit is sent to each core. This cannot be guaranteed for the delivery of synchronisation flits. Hence, the synchronisation flits must use a schedule that allows such situations. Walter proposes the usage of the One-to-All schedule for the synchronisation NoC to ensure timing predictability of the synchronisation. Furthermore, with the assumed hardware support the synchronisation flits are not treated in the same way like data flits. Instead of inserting them to the receive buffer in the NI an additional bit array is provided in the NI that stores if an unhandled synchronisation flit has been arrived from a specific node of the NoC. Thereby, each bit in the array stands for a specific node of the NoC. A sender node asks for readiness for a receiver node by checking the bit array. If the bit of that node is set it assumes readiness for this node and clears the bit to 0 to enable the handling of further synchronisation flits from that node. Afterwards, all flits are sent and there is no need for further acknowledges, as both communicating nodes exactly know the message's length.

If a specific platform and TDM schedule is utilized, it is feasible to define exact values for the admission and transportation time t^a and t^t. We assume a quadratic two-dimensional torus and thus, the parameters which adapt the calculation to a specific application are the NoC size $N = n \cdot n$, the number of flits to send f and the number of processes participating in the communication χ. That means if a platform with e.g. $n = 8$ is given there are $8 \cdot 8 = 64$ nodes on the chip.

If not stated otherwise we utilize in this section the Triplet schedule as it provides acceptable performance in all investigated situations and is competitive to all other schedules (see Section 4.3). The schedule exhibits an admission time of $t^a = 2n \cdot min(\chi, n)$ cycles per TDM cycle (see Section 4.2.2 for details). This time directly depends on the target hardware, as n indicates the dimension of the NoC. The architecture needs a latency of one cycle to forward a flit from one router to the next. Aka the hop latency L^{hop} is equal to all physical links and is equal to the slot latency.

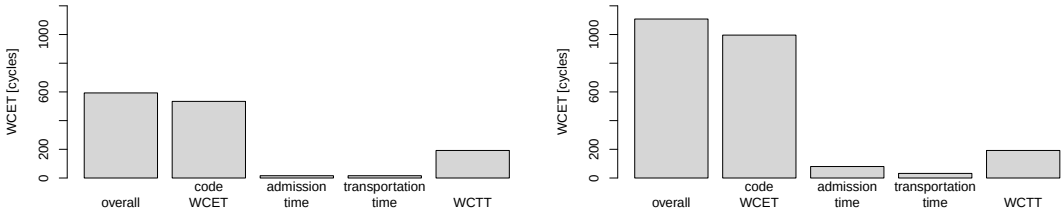
$$\forall i, j \in [0, n] : L^{hop}_{i,j} = L^{slot}_{i,j} = 1 \; cycle \tag{5.44}$$

The transportation time is upper bounded to $t^t = 2 \cdot n$ cycles for all paths of the NoC. While the NoC size influences t^a and t^t directly, the other parameters (f and χ) mainly influence the timing through the formalism.Thereby, f can be inserted directly and χ defines together with the applied communication patterns the values of the direct communication partners.

Another considerable parameter for the evaluation is the optimization level used to compile the MPI code. As the WCET analysis of code compiled with a higher optimization level than O0 is complex and causes a high amount of manual effort, the analysis with these optimization levels is out of scope of this work. Instead, we identify the range of influence of the code WCET by providing values for the upper and the lower bound. The optimization level O0 is applied to obtain the worst-case influence of code on timing behaviour. On the other hand we compare this timing to the theoretical optimum due to optimization. Hence, we calculate the timing bounds for the theoretical case that code is optimized to an execution time of 0 cycles. Please note that the theoretical optimum is not reachable in practice. Instead, it is provided to exhibit a range in that the timing bound lies depending on the provided optimization level for compiling the code.

The subsequent discussion covers the investigation of the building blocks of each communication according to their worst-case timing. The overall WCET is plotted, once considering the compilation with optimization level O0 and once considering the theoretical optimal code WCET of 0 cycles. The theoretical values only consider admission and transportation times and are labelled with "WCTT". If not stated otherwise the platform utilized for analysis is fixed to a size of $n = 8$, which means that it contains 64 cores. The analysis assumes to deliver a message consisting of $f = 6$ flits or 48 bytes, respectively. We also examined different message sizes including lengths up to 100 flits. But as the results for the building blocks are similar for all lengths, we restrict this part of the evaluation to one message length that facilitates the presentation.

The building blocks do not directly sum up to the overall WCET with O0 optimization. This is because overlapping of times is already taken into account. Thus, if for example admission time and execution of code has to be considered concurrently this time interval is assigned to the time for code execution and does not occur in admission time. There



(a) Send → Recv (b) concurrent MPI_Sendrecv

Figure 5.7: The building blocks of the overall WCET of point-to-point communication. Each bar represents one building block or the overall timing bound, respectively. Please note that the admission and transportation times only present the times that are not covered with other building blocks. In contrast, the WCTT block shows the time without the code WCET which reveals the part of the timing that is covered by code execution.

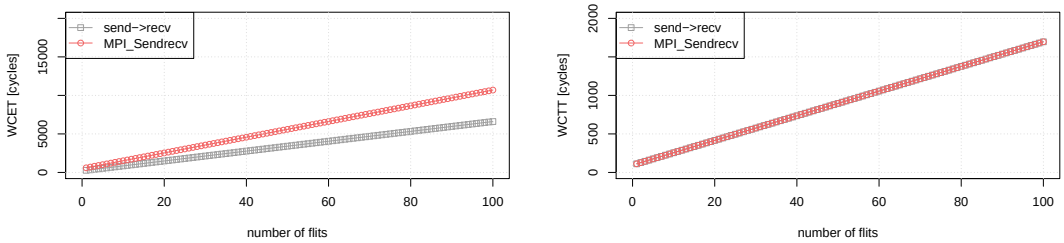
is a strict order for overlapping the building blocks regarding the presented times. The admission time overlaps transportation time, which in turn is covered by code WCET. The time for code execution cannot be overlapped.

After investigating the building blocks, the communication patterns are compared to each other. Thereby, the comparison is performed for different group sizes and message sizes to cover a large amount of possible application scenarios.

5.3.1 Point-to-Point Communication

Figure 5.7 shows the building blocks of the end-to-end WCET bounds of all point-to-point communications. Thereby, Figure 5.7a targets communication with a simple send and receive operation, while Figure 5.7b focusses on a concurrent sendrecv operation. In both diagrams the investigation exhibits that the contribution of the transportation time to the WCET is nearly negligible. The same holds true for the admission time, that only contributes slightly more than the transportation time. However, the WCTT bound without considering the code shows a higher time than the accumulation of the admission and the transportation time. This is caused by the previously described overlapping and indicates that it makes nearly no sense to optimize admission and transportation time as long as the code WCET is large compared to these times. It is also to note that the code is compiled with O0 optimization which leads to high WCETs. The range between both overall timing bound and the timing bound for WCTT describes the potential for optimizing the timing of code execution either through optimizing code, its execution or better analysis methods for local code execution. Another appropriate option would be to support the communication by providing special hardware. A DMA for example would cause the sending of each flit in hardware instead of performing it in software.

The timing of all components is constant for different group sizes, since in an explicit send to receive communication there are always two processes involved and in a send-receive communication there are three processes involved directly no matter how large the group of the applied communicator is. Focusing on explicit send to receive communication, the diagram of Figure 5.7a shows that the execution of code is more than double of the time



(a) Overall WCET of communication

(b) WCTT of communication

Figure 5.8: Comparison of the different point-to-point communications. The diagrams show the timing upper bounds as a function of the group size. Please note that the timing bounds are given in cycles and the scale between both diagrams varies with an order of magnitude.

than the overall WCTT. Additionally, the admission and transportation times are almost completely covered by the code WCET bound. The same holds true for the concurrent sendrecv communication (see Figure 5.7b). However, with this communication the code WCET is even more dominant than with the simple send and receive communication (about five times as large).

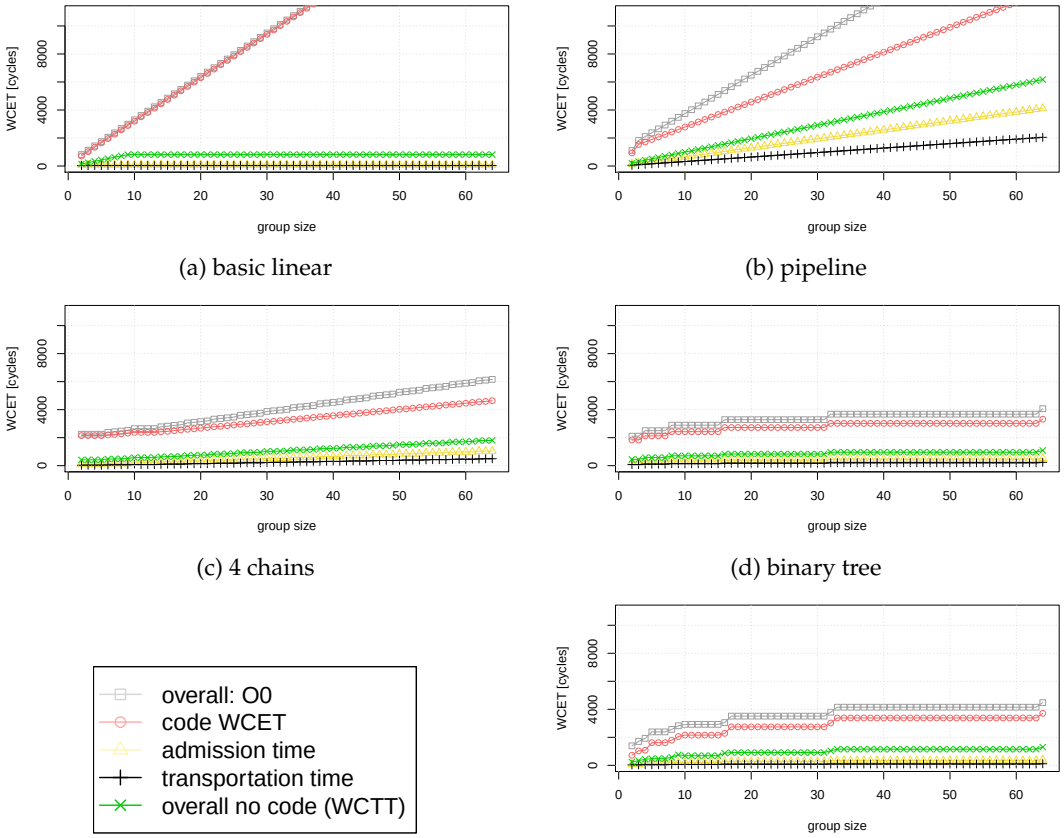
The comparison of the different send and receive operations is displayed in Figure 5.8. Thereby, it is performed once with the overall worst-case timing (see Figure 5.8a) and once considering only the WCTT (see Figure 5.8b).

Since the explicit send to receive communication has less work to do than the other communication, it is clear that it exhibits the lower timing bound for the overall WCET. The concurrent send-receive communication always exhibit a higher timing bound than the simple send to receive communication. Moreover, the send-receive operations shows a higher gradient in relation to the message size sent. However, when considering the WCTT the timing of both communication is similar. Though the sendrecv communication must perform a send and a receive operation in the same process, there are no additional costs for the WCTT of the concurrent sendrecv communication. The reason is that the receive operation only causes additional effort for the code part of the operation, while admission and transportation time of both communication parts (send and receive) overlap each other.

5.3.2 Collective Communication based on a Central Process

As already mentioned we apply the WCET analysis to the MPI_Broadcast operation as a representative of the communications that are based on a central process. Thereby, we assume that a message with f flits is sent to $\chi - 1$ receiver processes in a NoC with $N = n \times n$ nodes. Thus, in the end each participating process owns the same message with a length of f flits. If not stated otherwise the standard values for the parameters are $n = 8$, $f = 6$ and $\chi = 64$.

Figure 5.9 shows the building blocks of the end-to-end WCETs of all investigated communication patterns. The diagrams clearly show that the execution of code covers the highest amount of the overall WCET. Furthermore, the investigation exhibits that the contribution of the transportation time to the overall WCET is nearly negligible for all communication



(a) basic linear

(b) pipeline

(c) 4 chains

(d) binary tree

(e) legend

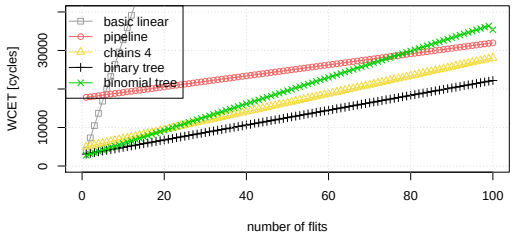
(f) binomial tree

Figure 5.9: WCETs of the building blocks that form the patterns' end-to-end WCET of MPI_-Broadcast. The diagrams are plotted as a function of the group size and each one illustrates the building blocks of another communication pattern. The legend for all graphs is displayed in Figure 5.9e.

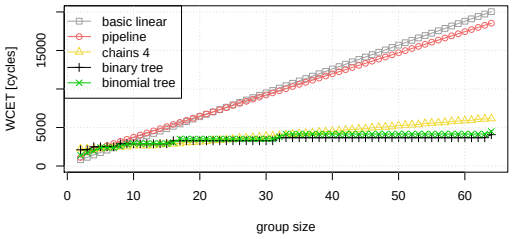
patterns except for the pipeline pattern. The same holds true for the admission time, that only contributes slightly more than the transportation time. Similar to the point-to-point communication the WCTT without considering the code shows a higher time than the accumulation of the admission and the transportation time. This is again subject to the previously described overlapping of timing. It is also to note that the code is compiled with O0 optimization which leads to high WCETs. As already mentioned in the previous section a better optimization of the code execution, a better analysis method for the local execution or hardware support may reduce or even eliminate the dominance of the code WCET.

Figure 5.10 provides a comparison of all communication patterns applied for MPI_-Broadcast. The WCET bounds are plotted as a function of the group size χ and as a function of the message length f.

The WCET of the basic linear pattern increases very fast with the number of flits sent (see Figure 5.10a). Compared to basic linear the other patterns are relatively stable in terms of message size, though there are some differences. The influence of the message size rises for the chains pattern with the number of applied chains. This is revealed by the graphs for 4

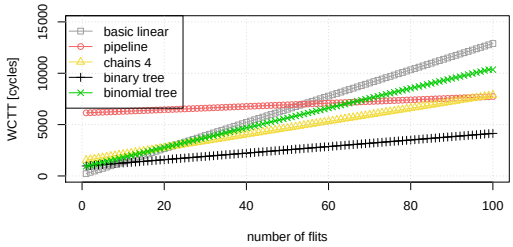


(a) WCET as a function of message length f. The group size χ is set to n^2, which means that all 64 cores are participating in the collective.

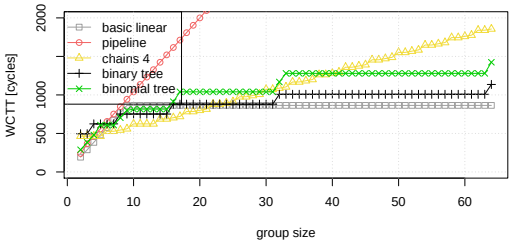


(b) WCET as a function of group size χ. The message length f is set to 6 flits which cause a message size of 48 bytes.

Figure 5.10: Overall WCETs of the communication patterns for MPI_Broadcast. Please note that the scale of WCET is different for both diagrams.



(a) WCTT as a function of message length f. The group size χ is set to n^2, which means that all 64 cores are participating in the collective.



(b) WCTT as a function of group size χ. The message length f is set to 6.

Figure 5.11: WCTTs of the communication patterns for MPI_Broadcast. Please note that the scale of WCTT is different for both diagrams.

chains and the pipeline pattern, which is in fact a pattern with 1 chain. The gradient of the binomial tree pattern raises even faster than the one of the 4 chains pattern. In comparison the one of the binary tree pattern rises not that fast. Additionally to the gradient they differ in the absolute time for delivering a message as well. Here a higher number of applied chains has a positive influence. But again the binary tree seems to behave best of all compared patterns except for the binomial tree.

In terms of group size (see Figure 5.10b) there is a linear growth of WCETs with a rising number of participating nodes for the basic linear, pipeline and chains patterns. In case of the binary and the binomial tree the WCETs raises in step function where the location of the steps are related to the height of the trees. The pipeline and basic linear pattern perform worst for large group sizes while the binary tree exhibits the best results for those groups. For small groups the basic linear pattern performs best but in those cases all pattern show a similar timing bound. However, the good performance of basic linear is facilitated by the relatively short message (6 flits) considered in this evaluation.

In Figure 5.11 we present a comparison of the different communication patterns of MPI_-Broadcast in terms of WCTT. This comparison is provided in addition to the one in Figure 5.10 to observe the timing behaviour of the communication patterns when they are not dominated with the code WCET. Thus, this investigation is of special interest if one targets

to use for example hardware extensions like a DMA to reduce the dominance of the code execution.

The first observation is that the total values of the timing bound are only about half of the values of the overall WCET. In Figure 5.11a the WCTT is investigated as a function of the message size. Similar to the overall WCET the timing bound of the basic linear pattern increases with the highest gradient. However, this time it is much more competitive. It even behaves best for small message sizes. On the other hand the pipeline pattern exhibits the lowest gradient. However, it exhibits a high WCTT even for small messages. Thus, it becomes competitive only for large messages. For the largest message investigated it is competitive to all observed patterns except for the binary tree. This communication pattern behaves best for message sizes larger than 8 flits. However, as its gradient is larger than the one of the pipeline pattern it probably will be exceeded at a certain message length.

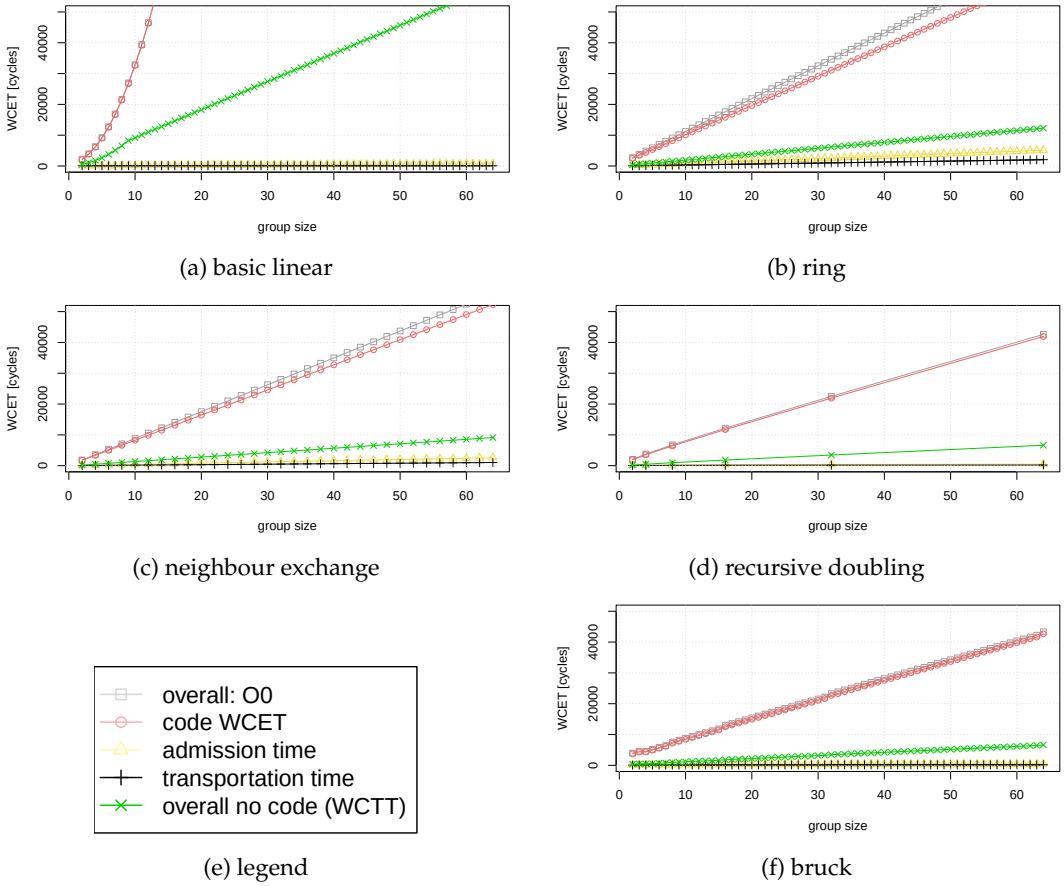
Figure 5.11b shows the comparison of the WCTT as a function of the group size. For this evaluation the message length is set to 6 flits, which is short enough that the basic linear pattern is still competitive to the other patterns. Thus, in this investigation it behaves best when more than 31 processes participate in the communication. The reason is that with this pattern and applied schedule each flit of the message can be delivered to all participating processes within one TDM cycle (see Section 4.1). All other patterns increase their WCTT stepwise with the number of applied participants. Thereby, the location of the steps is determined by the changes of the heights of the communication patterns. As the height of pipeline pattern increases with each additional process it exhibits a linear behaviour. In the range of 16 to 31 participants the binary tree pattern behaves equal to the basic linear pattern. However, they are exceeded by the chains pattern with 4 chains for group sizes lower than 22 and for groups with less than 4 processes the basic linear pattern again exhibits the best timing.

Please note that theses results consider that there is no time needed for code execution at all. As this is not possible in reality the timing behaviour of real systems will probably be located in between the two investigations of Figure 5.10 and 5.11.

The simple basic linear pattern exhibits good timing bounds for short messages and small group sizes. On the other hand it is outperformed by the competing patterns very soon if one of these parameters rises. Especially for long messages this pattern is not feasible. The pipeline and the chains patterns show situations where they are competitive to other schedules but also reveal situation where they show relatively high timing upper bounds. The binary and binomial tree patterns perform similar in terms of varying group and message size. But in both terms the binary tree is slightly better than the binomial tree, especially for large messages.

5.3.3 Collective Communication based on Uniform Data Exchange

In this section we apply the WCET analysis to the MPI_Allgather operation as a representative of the communications that are based on uniform data exchange. Thereby, we assume that at the beginning each participating process owns a message of f flits. The communication distributes all messages to each other process. Therefore, in the end each of the χ participating processes owns a composed message with a length of $\chi \cdot f$ flits. The size



(a) basic linear

(b) ring

(c) neighbour exchange

(d) recursive doubling

(e) legend

(f) bruck

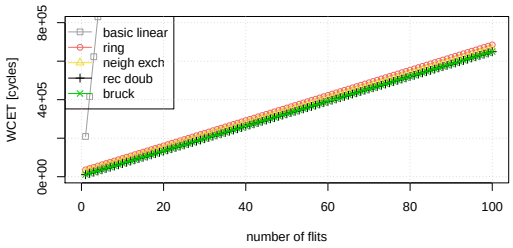
Figure 5.12: WCETs of the building blocks that form the patterns' end-to-end WCET of MPI_Allgather. The diagrams are plotted as a function of the group size and each one illustrates the building blocks of another communication pattern. The legend for all graphs is displayed in Figure 5.12e.

of the NoC is assumed as $N = n \times n$ nodes. If not stated otherwise the standard values for the parameters are $n = 8$, $f = 6$ and $\chi = 64$.

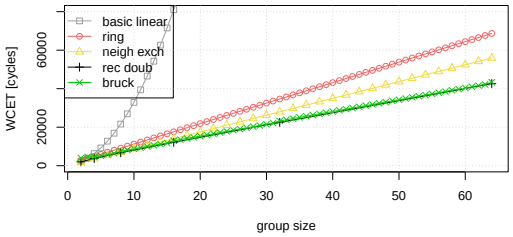
Figure 5.12 shows the building blocks of the end-to-end WCET of all investigated communication patterns for MPI_Allgather. Please note that the examined version of the *neighbour exchange* pattern needs groups of even size (see Figure 5.12c) and the examined version of the *recursive doubling* pattern is only feasible for group sizes with a power of 2 (see Figure 5.12d). As with MPI_Broadcast the overall WCET bounds are clearly driven by the code execution and admission as well as transportation time have nearly no impact in most patterns. Again this statement must be seen in the light that these times are partly covered. A comparison of the displayed overall WCETs and the WCTTs obtains a high potential to effectively reduce the overall WCET by optimizing the code execution, compilation or analysis.

Figure 5.13 compares the communication patterns for MPI_Allgather. The WCET bounds are plotted as a function of the group size χ and as a function of the message length f.

A close look on Figure 5.13a shows that all regarded patterns except the basic linear pattern behave quite similar in terms of varying message size. They exhibit the same gradient and

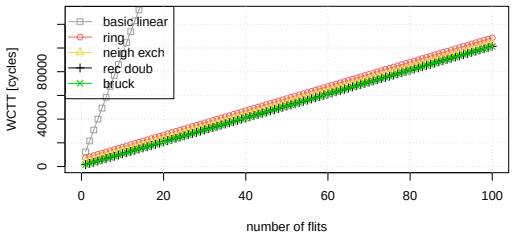


(a) WCET as a function of message length f. The group size χ is set to n^2, which means that all 64 cores are participating in the collective.

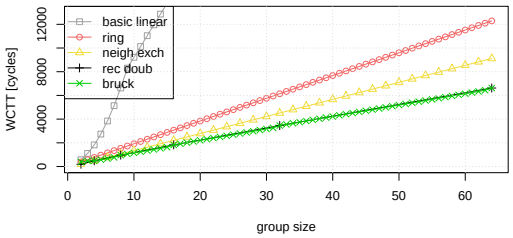


(b) WCET as a function of group size χ. The message length f is set to 6.

Figure 5.13: Overall WCETs of the communication patterns for MPI_Allgather. Please note that the scale of WCET is different for both diagrams.



(a) WCTT as a function of message length f. The group size χ is set to n^2, which means that all 64 cores are participating in the collective.



(b) WCTT as a function of group size χ. The message length f is set to 6.

Figure 5.14: WCTTs of the communication patterns for MPI_Allgather. Please note that the scale of WCTT is different for both diagrams.

the absolute WCET bounds differ only in a small range. Although the differences between the patterns seem marginal, it can be observed that the recursive doubling and the bruck pattern perform best. In contrast, the basic linear pattern exhibits a much higher gradient and absolute timing bound. Therefore, basic linear is not feasible for any message length.

In Figure 5.13b the WCET bounds of MPI_Allgather patterns are given in terms of the group size. Here, except for recursive doubling and bruck, a clear performance difference can be identified between the patterns. While the timing bounds are quite similar for small group sizes, they tend to drift apart with an increasing number of participants. Analogous to a varying message length the timing behaviour of the recursive doubling pattern and the bruck pattern are nearly the same. These two patterns also mark the lowest timing bounds for all examined group sizes.

In Figure 5.14 we present a comparison of the different communication patterns of MPI_-Allgather in terms of WCTT. This comparison is provided in addition to the one in Figure 5.13 to observe the timing behaviour of the communication patterns when they are not dominated with the code WCET. Thus, like with the examination of the WCTTs in Section 5.3.2 this investigation is of special interest if one targets to use for example hardware extensions like a DMA to reduce the dominance of the code execution.

A comparison between Figure 5.13 and 5.14 exhibits the large difference between the

overall WCET and the WCTT which is nearly an order of magnitude. In Figure 5.14a we investigate the WCTT as a function of the message size. With this examination the WCTTs show a similar behaviour as with the overall WCET. Except for basic linear all patterns show nearly the same timing behaviour. They exhibit nearly the same timing bounds for equal message sizes and increase linearly with the same gradient. Though the basic linear pattern only exhibits slightly larger timing bounds than the other patterns for very small messages (1 flit) the WCTT raises very fast with the number of delivered flits. Hence it cannot be seen competitive to the other patterns for any message size. Similar to the investigation of the overall WCET the recursive doubling and the bruck pattern perform best. However, there is only a small gap between those patterns and the ring and neighbour exchange pattern.

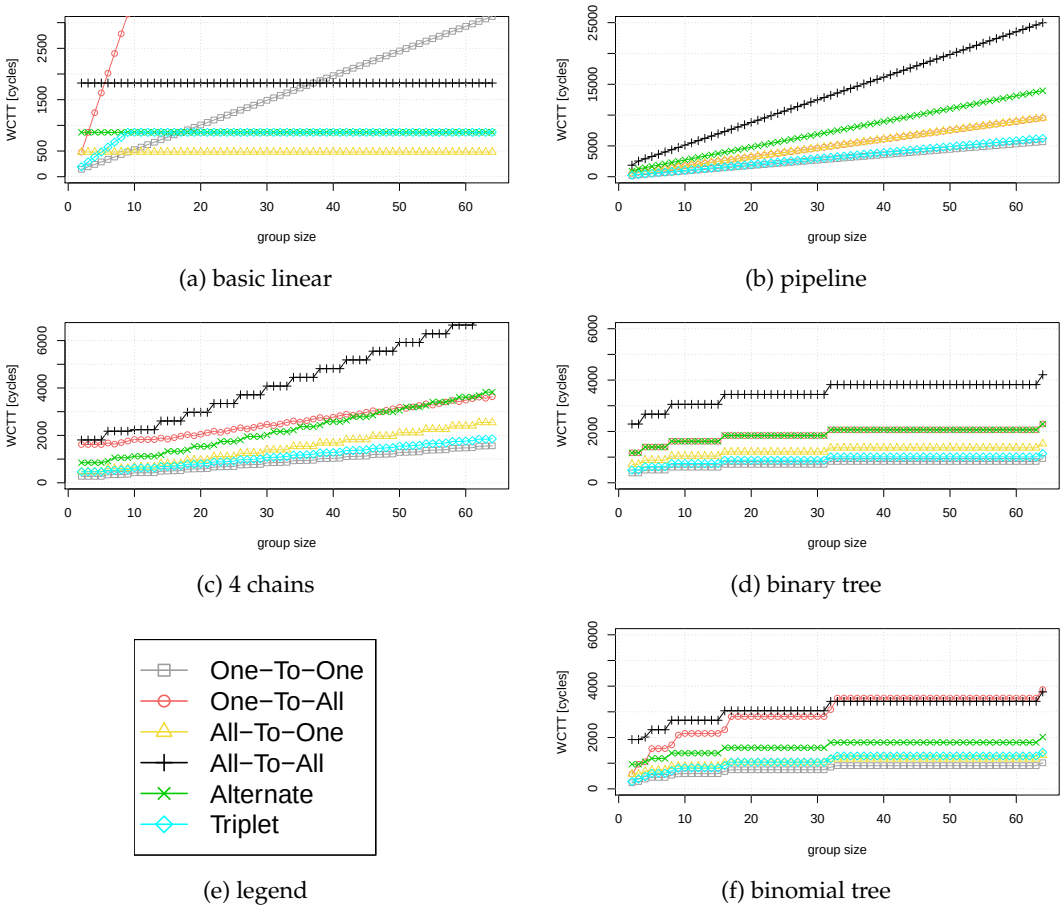
In Figure 5.14b the comparison of the communication patterns' WCTT is displayed as a function of the group size. This examination exhibits a similar behaviour as for the overall WCET. Except for extremely small group sizes the basic linear pattern is clearly not competitive to the other patterns. The guaranteed performance of the other patterns drift apart with an increased number of participating processes. While the ring pattern show the highest timing bounds of the remaining patterns, the chains pattern provides slightly better guarantees. As with the comparison with different message length the best WCTTs exhibit the recursive doubling and the bruck pattern. However, the recursive doubling is only suitable for group sizes with a power of 2.

Altogether basic linear is not competitive to the other patterns. The ring and the neighbour exchange patterns perform on the one hand much better than the basic linear pattern but exhibit higher WCET bounds than the recursive doubling or bruck pattern. The patterns with the lowest timing bounds in all situations are recursive doubling and bruck. In terms of timing there is no reason to prefer one pattern but the investigated version of recursive doubling only provides group sizes which are a power of 2. Thus, to provide all numbers of participating nodes it is recommended that the bruck pattern is applied for data delivery.

5.3.4 Influence of TDM Schedules

This section provides a comparison between the TDM schedules presented in Chapter 4 if the communication patterns of the MPI operations are applied in software implementations of the communication operations. To reveal the full picture we examine the two classes of group communication by applying their representatives MPI_Broadcast and MPI_Allgather. Thereby, there is a comparison of the schedules for each available communication pattern and each comparison is done once as a function of the group size and once as a function of the message length. As already mentioned in the previous sections the code execution dominates the overall WCET guarantees. Since the optimization to reduce this dominance is possible (e.g. hardware extensions) but out of scope of this thesis, we concentrate on the WCTT guarantees for the applied communication to give a reasonable examination.

Figure 5.15 presents the comparisons of the schedules for the communication patterns suitable for the broadcast communication. When examining the basic linear pattern (see Figure 5.15a) the graphs exhibit the same shape as the ones in Section 4.3.1 (see Figures 4.12a and 4.12b). This similarity is because the basic linear pattern directly maps to the 1:N communication investigated in Section 4.3.1. The only difference is that in the current



(a) basic linear

(b) pipeline

(c) 4 chains

(d) binary tree

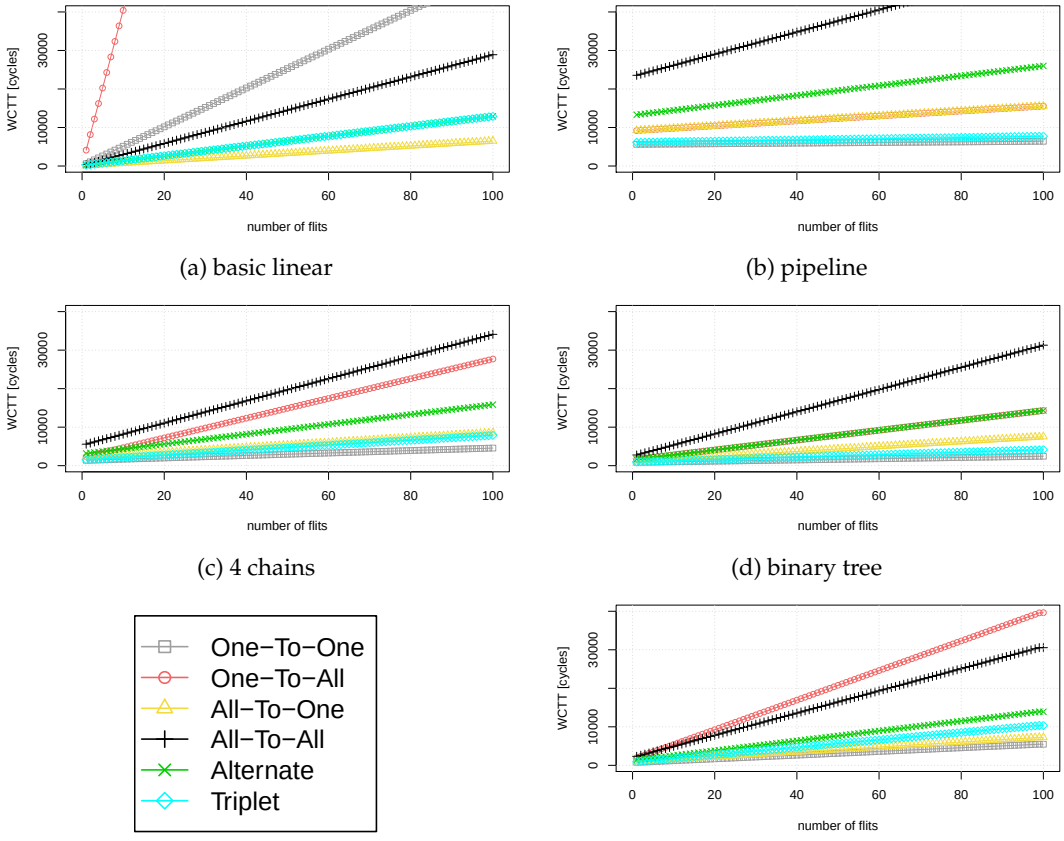
(e) legend

(f) binomial tree

Figure 5.15: WCTTs as a function of the group size of the communication patterns for MPI_-Broadcast. The diagrams are plotted as a function of the group size and each one illustrates the TDM schedules for another communication pattern. The legend for all graphs is displayed in Figure 5.15e. Please note that the scales of the WCTT vary for the different diagrams.

evaluation we use a different set of fixed parameters ($n = 8$ and $f = 6$) which causes a difference in the total numbers of the timing upper bounds. According to this similarity the discussion for this communication pattern is very similar to the discussion in Section 4.3.1. We see that except for group sizes smaller than n the All-to-One schedule exhibits the lowest timing bound. However, this result only holds for the broadcast communication and for a gather communication, which is related to a N:1 communication. When neglecting the differences about the different amount of data sent in total for both communication operations, the All-to-One and the One-to-All schedule would exchange their behaviour with the gather communication. For the small groups the One-to-One schedule performs best but this schedule suffers when it is applied to large groups. As already mentioned in Section 4.3.1 though the Triplet schedule never provides the lowest timing bounds it shows WCTTs that are competitive to the best schedules for each group size.

The One-to-One schedule exhibits the lowest timing upper bounds for all other schedules



(a) basic linear

(b) pipeline

(c) 4 chains

(d) binary tree

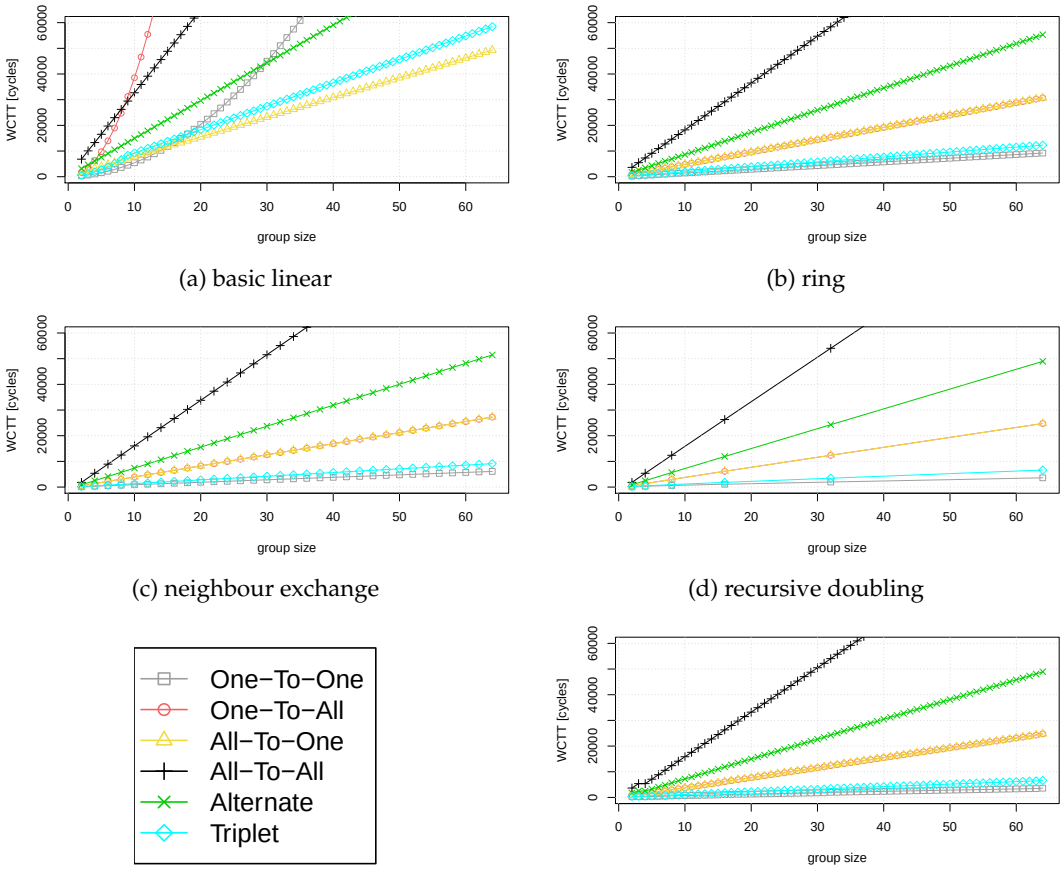
(e) legend

(f) binomial tree

Figure 5.16: WCTTs as a function of the number of flits of the communication patterns for MPI_Broadcast. The diagrams are plotted as a function of the message size and each one illustrates the TDM schedules for another communication pattern. The legend for all graphs is displayed in Figure 5.16e.

in Figure 5.15. However, the Triplet schedules typically performs only slightly worse than the One-to-One schedule. Thereby, the difference between both TDM schedules ranges from 48 cycles to 192 cycles for a group size of 2 and from 192 cycles to 552 cycles for a group size of 64 depending on the according communication pattern. All other schedules exhibit significant higher timing upper bounds, except for the All-to-One schedule with the binomial tree and binary tree pattern. However, as mentioned before this schedule suffers when other communication operations than the presented one are applied (e.g. gather).

Figure 5.16 compares the TDM schedules for the broadcast communication as a function of the message length. Thereby, we assume a NoC size of $n = 8$ and a group size of $\chi = 64$. All the WCTT upper bounds of all schedules increase linearly with the number of flits sent for each communication pattern. As already indicated in Figure 5.15 the All-to-One schedule exhibits with the basic linear pattern the lowest timing upper bound for all message sizes. The second best schedules for the pattern are the Triplet and the Alternate schedules. They exhibit exactly the same timing bounds for this configuration and timing pattern. The One-to-One schedule provides the second highest WCTT and shows even a higher timing



(a) basic linear

(b) ring

(c) neighbour exchange

(d) recursive doubling

(e) legend

(f) bruck

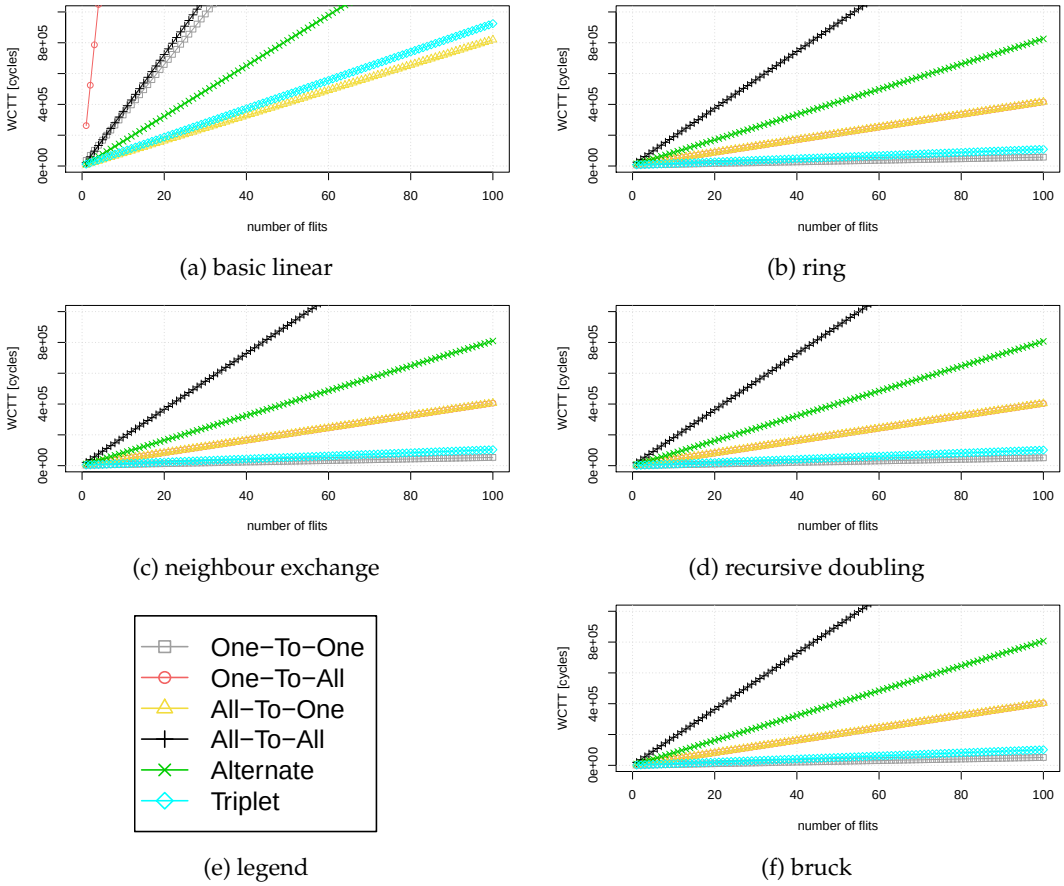
Figure 5.17: WCTTs as a function of the group size of the communication patterns for MPI_-Allgather. The diagrams are plotted as a function of the group size and each one illustrates the TDM schedules for another communication pattern. The legend for all graphs is displayed in Figure 5.17e.

bound than the All-to-All schedule.

However, similar to the behaviour with varying group sizes the One-to-One schedule behaves best for all communication patterns except for the basic linear pattern. On the other hand the Triplet schedule exhibits timing upper bounds that are mostly relative near to the timing bounds of the One-to-One schedule and also the All-to-One schedule shows competitive results for this configuration. The other schedules typically exhibit significant higher WCTTs. Thereby, for those other schedules the Alternate schedule shows the best performance.

After examining the communication patterns for the broadcast communication we take a closer look on the communication patterns for the allgather communication. At first, we concentrate on the WCTT upper bounds as a function of the group size which are displayed in Figure 5.17. For that the message size and the NoC size are kept on fixed values ($f = 6$ and $n = 8$).

A closer look on Figure 5.17a reveals that the One-to-All as well as the One-to-One



(a) basic linear

(b) ring

(c) neighbour exchange

(d) recursive doubling

(e) legend

(f) bruck

Figure 5.18: WCTTs as a function of the number of flits of the communication patterns for MPI_Allgather. The diagrams are plotted as a function of the message size and each one illustrates the TDM schedules for another communication pattern. The legend for all graphs is displayed in Figure 5.18e.

schedule increase with a quadratic like function. The reason for this behaviour is that the total amount of data sent in an allgather communication raises with the number of group members since we assume that each process send f flits to each other group member. Because of this behaviour the One-to-One schedule exhibits the lowest timing bounds for small group sizes but raise to be the second worst schedule for large groups. The All-to-One schedule is the best in terms of guaranteed WCTTs for large group sizes. While the One-to-All, the All-to-All and the Alternate schedules are only competitive for very small groups the Triplet schedule shows results that are rather near to the values of the schedules with the lowest WCTT in each case.

Similar to the broadcast communication the One-to-One schedules shows the lowest timing bounds for all communication patterns except for basic linear. Furthermore, in each case the Triplet schedule is near the best exhibited timing bound and the other schedules show significant higher WCTTs bounds. Thereby, the All-to-All schedule behaves worst, the Alternate exhibits the second highest timing bounds and the One-to-All and All-to-One schedules show an equal timing and are better than the Alternate schedule.

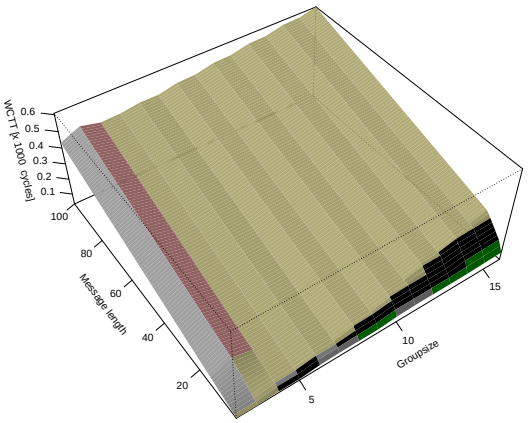
Figure 5.18 presents the TDM schedules for the allgather communication as a function of the message length. In the diagrams of this figure each schedule increases linearly with the number of flits for all communication patterns. Like also observed for varying group sizes the All-to-One schedule shows the lowest WCTT upper bounds for varying message length when applying the basic linear pattern (see Figure 5.18a). The Triplet schedule is slightly worse than the All-to-One schedule and the other schedules are not competitive for this communication pattern.

Like it is the case in all other investigations the One-to-One schedule behaves best for all communication patterns except for the basic linear pattern. Another similarity to the other examinations is the behaviour of the Triplet schedule. It performs only slightly worse than the One-to-One schedule in all situations. Like with varying group sizes the One-to-All and the All-to-One schedule show an equal performance for all these patterns and are the third best option. The All-to-All schedule behaves worst and the Alternate schedule exhibits the second highest timing bounds.

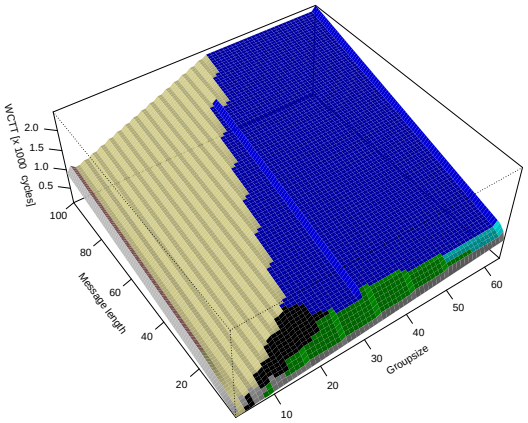
Altogether, the One-to-One schedule performs best for all communication patterns (for the broadcast and the allgather operation) except for the basic linear communication patterns. The reason why the One-to-One TDM schedule performs that good for all group sizes with the different communication patterns, is located in the structure of the patterns. If regarding a single process that must send or forward a message to other processes typically this schedule exhibits a higher WCTT with an increasing number of receiving processes that must be served. However, the communication patterns limit the number of processes that must be served for each process forcing the processes to a situation that is similar to the one for the basic linear pattern with a small group size. Each process in the binary tree pattern owns at maximum only two child processes for example. In such situations the One-to-One schedule can shine with a low WCTT bound and this performance benefit also outperforms the disadvantage of sending a message along a row of processes instead of sending it directly to the target process.

However, there remains the question which combination of a specific schedule and communication pattern performs best. According to the results revealed previously in this section we only regard to a subset of TDM schedules to answer this question. Due to the restriction of providing low timing bounds only for one communication direction (either 1:N or N:1) we reject the One-to-All as well as the All-to-One schedule. Furthermore, the All-to-All schedule is discarded because of its poor performance in most situations. The Alternate schedule is in no situation better than the Triplet schedule. Hence, we also reject the Alternate schedule for the following discussion. After those considerations it remains the One-to-One schedule as the most promising schedule for a large variarity of communications and the Triplet schedule which provides a good performance for the basic linear patterns.

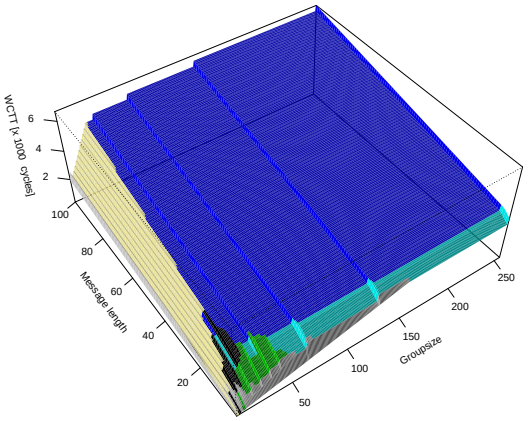
For the targeted comparison we provide an overview to exhibit which pattern shows the lowest WCTT in which situation for both regarded schedules in Figures 5.19 to 5.22. Please note that these figures only present an overview of which communication pattern behaves best in terms of WCTT in each situation, while Figures 5.11 and 5.14 can be used to extract the actual differences of the timing bounds for different communication patterns.



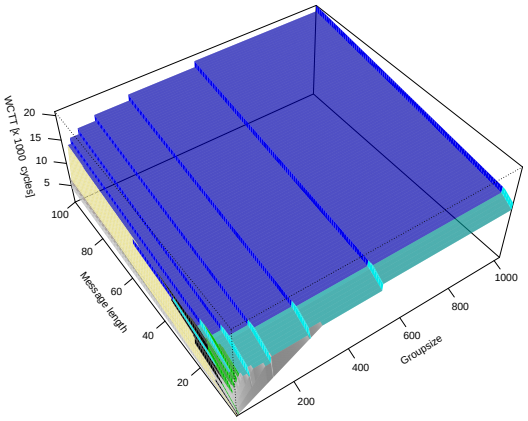
(a) Minimal WCTTs in a NoC with the size $N = 4x4$



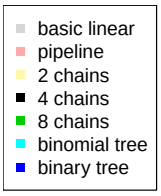
(b) Minimal WCTTs in a NoC with the size $N = 8x8$



(c) Minimal WCTTs in a NoC with the size $N = 16x16$



(d) Minimal WCTTs in a NoC with the size $N = 32x32$



(e) legend

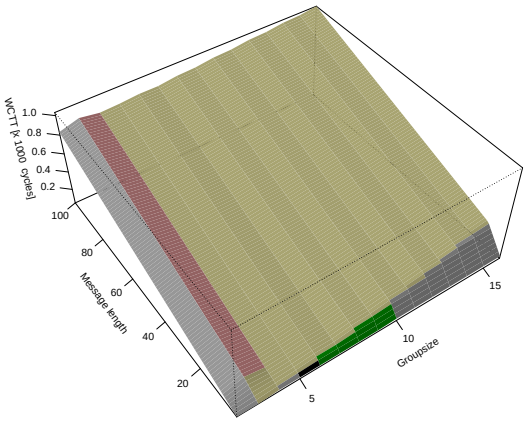
Figure 5.19: The minimal WCTTs exhibited for any communication pattern of the broadcast communication when applying the One-to-One schedule. The results are given as a function of the group size and the number of flits and each diagram displays the times for a different size of the NoC. The colours mark the communication pattern that is used to obtain the minimal WCTT. The legend for all graphs is displayed in Figure 5.19e.

Figure 5.19 shows the minimal exhibited WCTT bounds for a broadcast communication when applying the One-to-One TDM schedule and indicates which communication pattern is utilized to obtain that minimal timing bounds. Though there are some differences in detail it is possible to reveal some general trends that hold for all presented sizes of the NoC. The basic linear pattern is relevant for a group size of 2 (which in fact is a simple point-to-point communication) and for short messages with various group sizes. However, the concrete configuration for that basic linear behaves best is dependent on the NoC size. The group size of 3 is the only size where the pipeline pattern shows the best performance of all patterns. The chains communication pattern with 2 chains is the best option for communications with group sizes larger than 3 and up to about 20 to 40 members in the group. Thereby, the concrete group sizes depend on the message length and the size of the NoC.

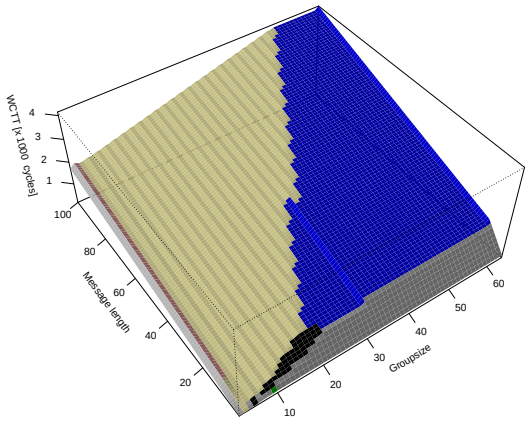
For short messages there may be other communication patterns that are more suitable. The binary tree pattern is typically suitable if the applied communications combine intermediate or long message sizes with intermediate or large group sizes. However, this statement does not hold for small NoCs like the one in Figure 5.19a because with those NoCs there are not enough nodes available to configure group sizes where the binary tree pattern performs best. Chains patterns with more than 2 chains show their benefits with small to intermediate message and group sizes. Again there are small differences between different NoC sizes but in general the application area of these schedule is similar for all NoCs. The binomial tree pattern exhibits the best results for intermediate and large NoC sizes. Thereby, it is restricted to small message sizes in combination with intermediate to large group sizes.

An interesting observation in Figure 5.19 is the border between the 2 chains pattern and the binary tree pattern. Though this border shows a stepwise shape it can be approximated to a linear border for message and group sizes that are large enough. While in the chains pattern only the root process has to serve 2 children, this is the cases for all processes in the binary tree pattern. Thus, each flit on the worst-case path of the binary tree experiences a larger delay at each process (except for the root process) than on the worst-case path of the 2 chains pattern. However, the height of the chains pattern grows faster than the one of the binary tree pattern. These two aspects are probably the reason for the shape of the border between the two patterns. Furthermore, the revealed steps show a difference of the group size of 2 (see Figure 5.19b) which is probably caused by the fact that the height of the 2 chains pattern grows with every second additional process participating in the communication.

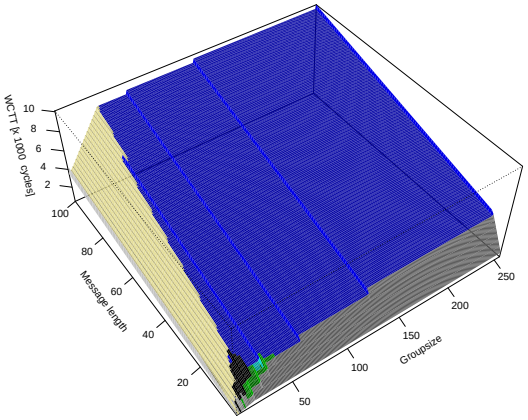
Figure 5.20 shows the minimal exhibited WCTT bounds for a broadcast communication when applying the Triplet schedule and indicates which communication pattern is utilized to obtain that minimal timing bounds. The exhibited results are similar to the ones with the One-to-One schedule, but there are some differences especially for the consideration of small message sizes. With the One-to-One schedule in this area of configurations there are different patterns (like various chains patterns and the binomial tree) that show minimal timing bounds. In contrast, when regarding the Triplet schedule the areas where those patterns show the best results are reduced. Instead, the basic linear pattern occupies those configurations. Hence for small message sizes mostly the basic linear pattern shows the best results of all communication patterns. The other aspects of this consideration are quite similar to the consideration with the One-to-One schedule except for the 2 chains pattern which occupies a slightly larger area than in the examination with the One-to-One schedule.



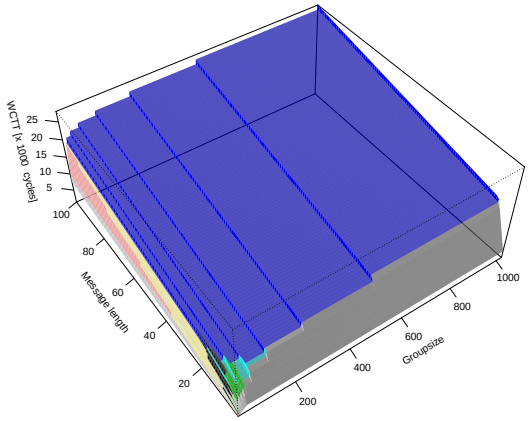
(a) Minimal WCTTs in a NoC with the size $N = 4x4$



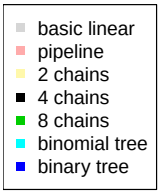
(b) Minimal WCTTs in a NoC with the size $N = 8x8$



(c) Minimal WCTTs in a NoC with the size $N = 16x16$



(d) Minimal WCTTs in a NoC with the size $N = 32x32$



(e) legend

Figure 5.20: The minimal WCTTs exhibited for any communication pattern of the broadcast communication when applying the Triplet schedule. The results are given as a function of the group size and the number of flits and each diagram displays the times for a different size of the NoC. The colours mark the communication pattern that is used to obtain the minimal WCTT. The legend for all graphs is displayed in Figure 5.20e.

The reason for the good performance of the basic linear pattern for small messages and intermediate to large group sizes is located in the One-to-All and All-to-One parts of the Triplet schedule. When these parts are used, all participants of the communication can be served with one flit within one TDM cycle. However, the period of this TDM cycle is larger than the one if the One-to-One part of the schedule is used which is possible for the application of the binary tree pattern. Moreover, the number of needed TDM cycles is related to the number of participating processes. These aspects seem to be the driving forces for the shape of the border between the binary tree and the basic linear pattern.

Figure 5.21 displays the minimal WCTT bounds for an allgather communication when applying the One-to-One schedule. Additionally, the colours indicate the communication pattern that produces the minimal timing bound for a given configuration. For all presented NoC sizes the same communication patterns produce the best timing bounds. Thereby, for most configurations the bruck pattern performs best. Only for group sizes that are a power of 2 the recursive doubling pattern is displayed as the one with the minimal WCTT bound. However, in those cases the recursive doubling pattern only reaches an equal bound as the bruck pattern but is not better than bruck. The reason is that we are focusing on pure traversal times which are the same for both patterns and recursive doubling can only be applied to group sizes with a power of 2 in the considered version. For group sizes lower than 5 the neighbour exchange communication pattern exhibits the best results. However, since this pattern can only be applied for even group sizes in Figure 5.21 ony the groups with size 2 and 4 are marked with the neighbour exchange pattern.

Similar to the WCTTs of allgather with the One-to-One TDM schedule the lowest WCTT bounds for the Triplet schedule are displayed in Figure 5.22. Though, the timing bounds seem to be higher for the Triplet schedule than for the One-to-One schedule, the shape of the exhibited results is very similar. Like it is revealed for the One-to-One schedule the bruck communication pattern shows the lowest timing bounds for most of the depicted configurations. Only the recursive doubling pattern exhibits an equal performance for some group sizes that are a power of 2. The neighbour exchange pattern is again the best pattern for small even group sizes. Finally, one can say that those observations hold true for all investigated sizes of the NoC.

Subsequently, it follows the comparison of the One-to-One and the Triplet TDM schedules. Thereby, we investigate the difference in the revealed minimum WCTT bounds for each schedule. Hence, we regard for each group size and the message length to the communication pattern that performs best for the considered configuration and schedule. Figure 5.23 displays the differences for the broadcast communication and Figure 5.24 focuses on the allgather communication.

In Figure 5.23 we compare the best exhibited values of the One-to-One and the Triplet schedule for the broadcast communication. Thereby, areas with positive values (marked grey) display the configurations where the One-to-One schedule shows lower timing bounds than the Triplet schedule and negative values (marked red) indicate the other case. As we can see the One-to-One TDM schedule outperforms the Triplet schedule for most configurations. Only for small messages that are sent to an intermediate or large group of processes the Triplet schedule shows better results. Furthermore, the overhead of the Triplet schedule to the One-to-One schedule is quite high compared to the total timing bounds for small NoC

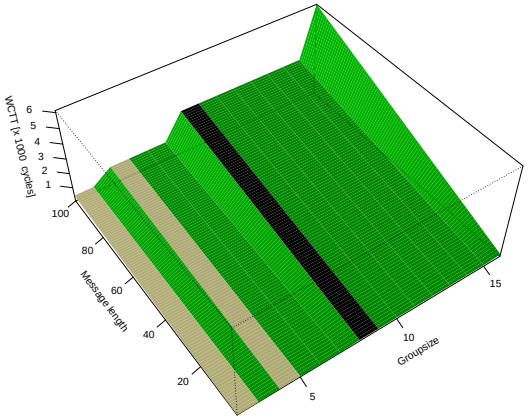
sizes. However, this effect reduces with a growing size of the NoCs and the savings in the configurations where Triplet wins increase.

Figure 5.24 compares the best exhibited values of the One-to-One and the Triplet schedule for the allgather communication. Thereby, grey areas mark the configurations where the One-to-One schedule shows lower timing bounds than the Triplet schedule and red areas indicate the other case. The examination of this aspect for the allgather communication reveals that the One-to-One schedule is able to outperform the Triplet schedule for each configuration investigated when an allgather communication is applied. This observation is caused by the structure of the according communication patterns which are all (except for basic linear) patterns that perform multiple steps of concurrent sendreceive communication (see Sections 2.3 and 5.2) and the One-to-One schedule is optimized for such communications.

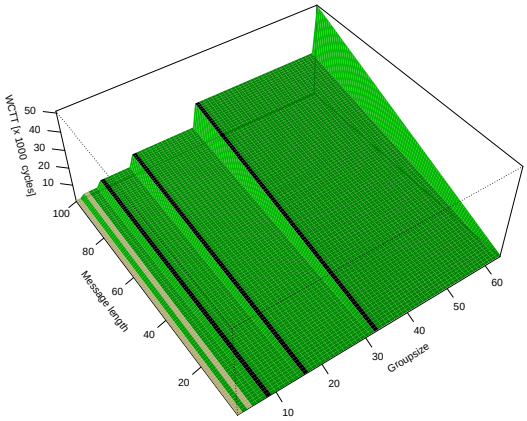
Altogether, the One-to-One schedule is the best option for most of the investigated configurations of different message sizes, group sizes and NoC sizes. The Triplet schedule outperforms the All-to-All, All-to-One, One-to-All and the Alternate schedule and exhibits results that are often near the One-to-One schedule. However, if one is free to apply any of the presented communication patterns for a specific communication the Triplet schedule can only beat the One-to-One schedule in special situations where short messages must be sent from one process to a large number of other processes. Nevertheless, for both schedules the software implementation of the communication must be carefully tuned to use the correct communication pattern to reveal the best results. With regard to the previously presented results this might be slightly easier for the Triplet schedule.

Furthermore, the presented results base on the assumption that it is possible to implement the investigated communication patterns. In the case that this assumption cannot be hold all messages must be send separately which naturally maps to the basic linear communication pattern. An example for such a case could be that intermediate processes are not allowed or not able the handle the data of other processes (cf. for example the intermediate processes of gather and scatter communications with tree patterns). Another example would be the case that there are multiple processes running on the same node and are independently communicating to processes on other nodes. Because of this independence it is probably not possible to combine them to build a common communication pattern. In those cases the Triplet schedule clearly outperforms the One-to-One schedule for large group sizes (cf. Figures 5.15 to 5.18). However, examination of such cases are out of scope of this thesis and are subject to future work.

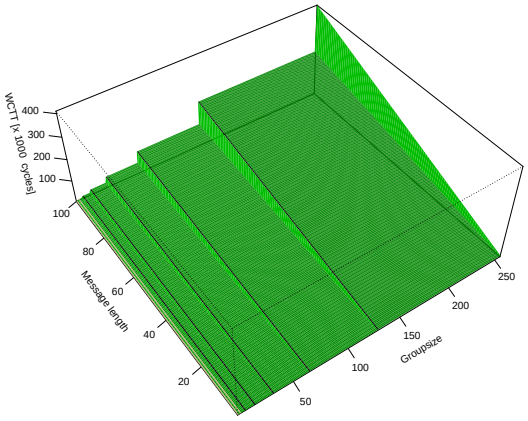
It is possible to utilize the One-to-All part of the Triplet schedule for the synchronisation of data transfers to save the hardware costs for an additional synchronisation NoC as needed for the One-to-One schedule. However, the investigation of this advantage of the Triplet schedule is subject to future work.



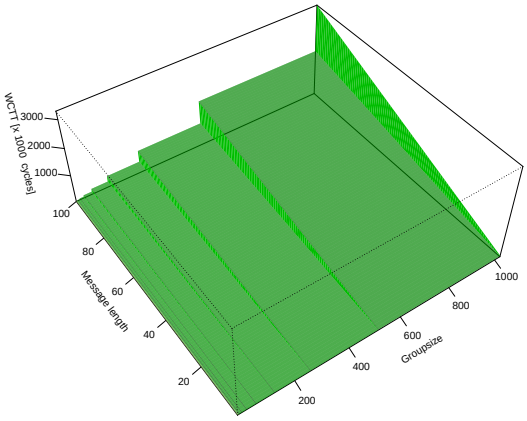
(a) Minimal WCTTs in a NoC with the size $N = 4x4$



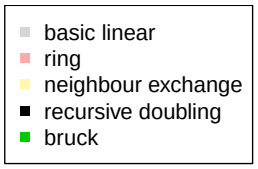
(b) Minimal WCTTs in a NoC with the size $N = 8x8$



(c) Minimal WCTTs in a NoC with the size $N = 16x16$

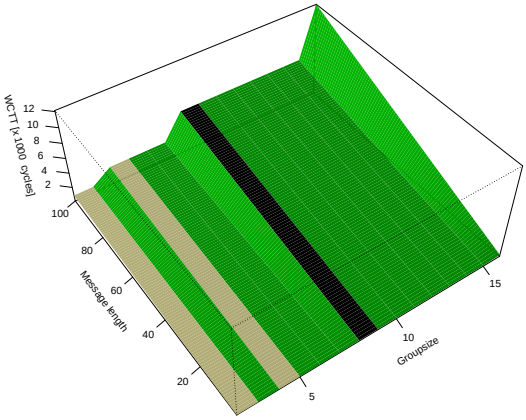


(d) Minimal WCTTs in a NoC with the size $N = 32x32$

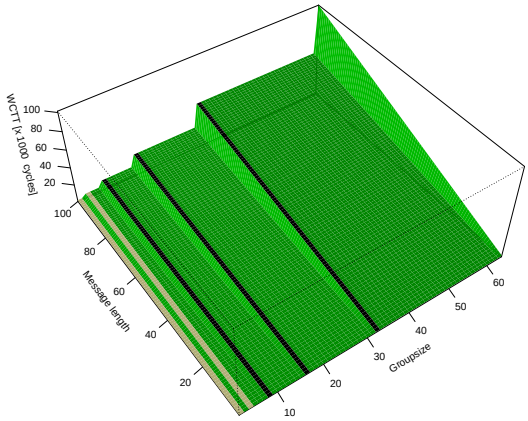


(e) legend

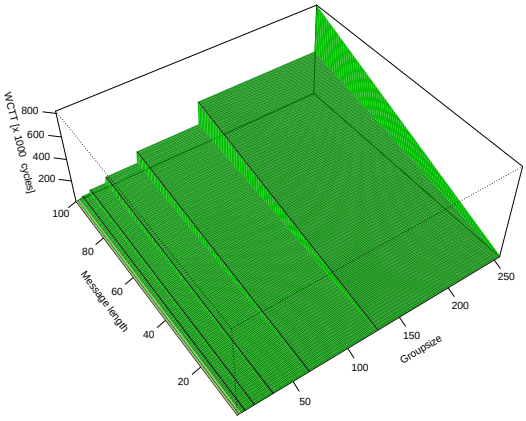
Figure 5.21: The minimal WCTTs exhibited for any communication pattern of the allgather communication when applying the One-to-One schedule. The results are given as a function of the group size and the number of flits and each diagram displays the times for a different size of the NoC. The colours mark the communication pattern that is used to obtain the minimal WCTT. The legend for all graphs is displayed in Figure 5.21e.



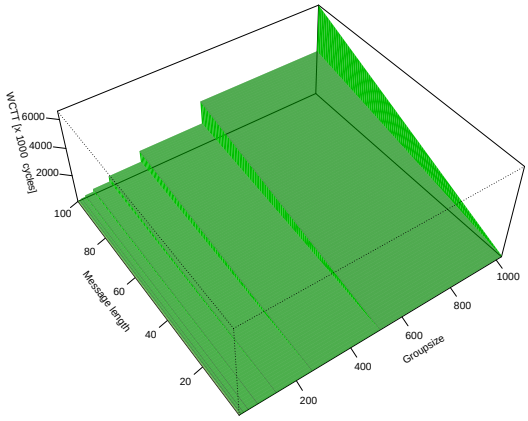
(a) Minimal WCTTs in a NoC with the size $N = 4x4$



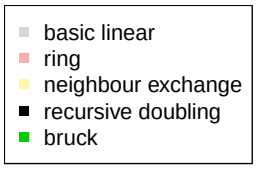
(b) Minimal WCTTs in a NoC with the size $N = 8x8$



(c) Minimal WCTTs in a NoC with the size $N = 16x16$

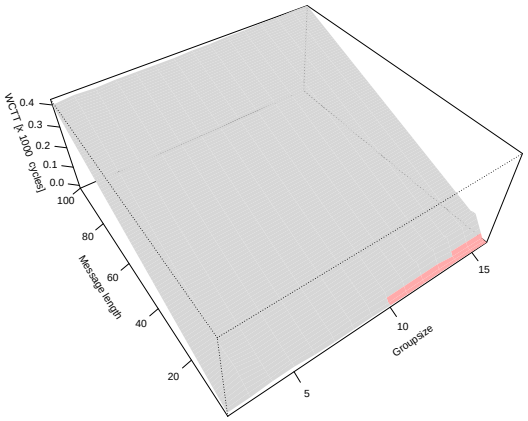


(d) Minimal WCTTs in a NoC with the size $N = 32x32$

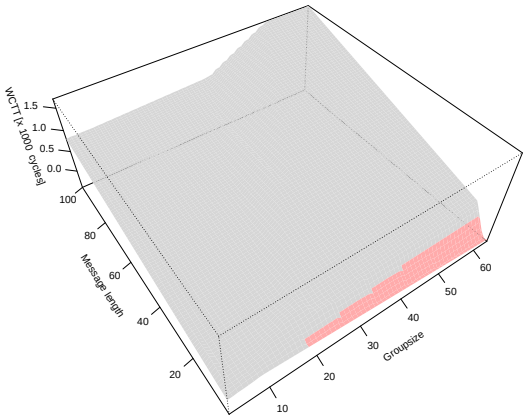


(e) legend

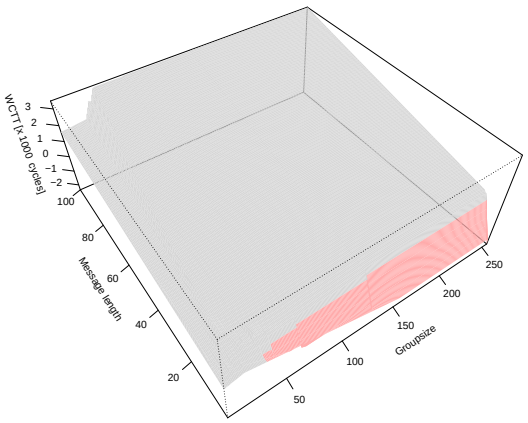
Figure 5.22: The minimal WCTTs exhibited for any communication pattern of the allgather communication when applying the Triplet schedule. The results are given as a function of the group size and the number of flits and each diagram displays the times for a different size of the NoC. The colours mark the communication pattern that is used to obtain the minimal WCTT. The legend for all graphs is displayed in Figure 5.22e.



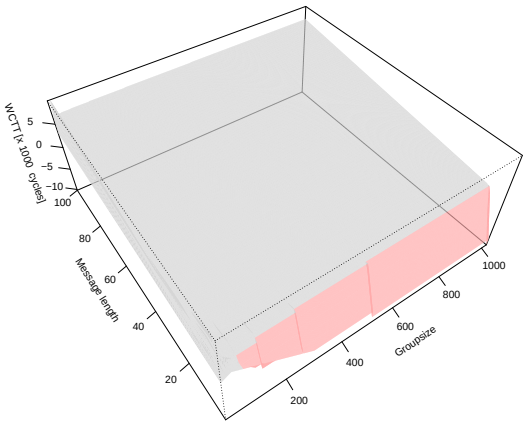
(a) Minimal WCTTs in a NoC with the size $N = 4x4$



(b) Minimal WCTTs in a NoC with the size $N = 8x8$

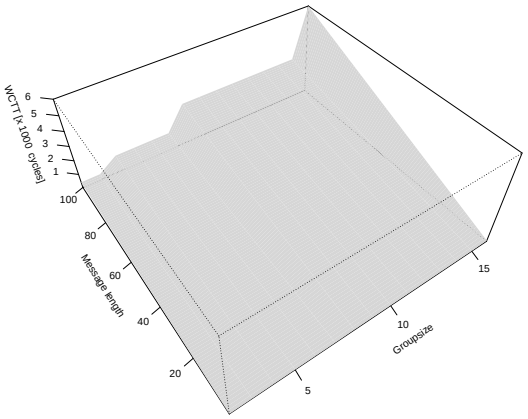


(c) Minimal WCTTs in a NoC with the size $N = 16x16$

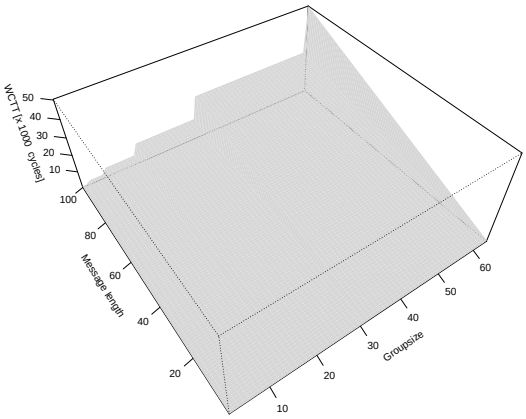


(d) Minimal WCTTs in a NoC with the size $N = 32x32$

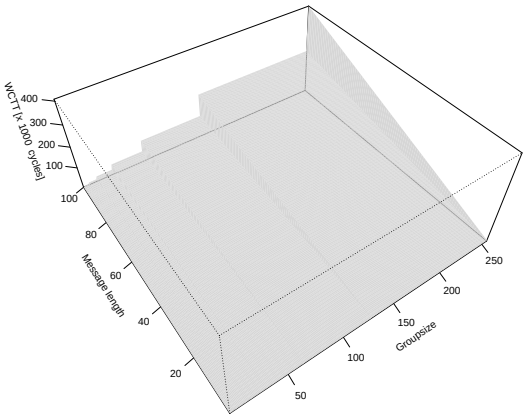
Figure 5.23: The difference in terms of WCTTs between the Triplet and the One-to-One TDM schedule for the broadcast communication. The results are given as a function of the group size and the number of flits and each diagram displays the differences for a different size of the NoC. Thereby, for each value the best time revealed for the One-to-One schedule is subtracted from the best time of the Triplet schedule. Hence, positive numbers indicate a lower WCTT for the One-to-One schedule while negative numbers show a better performance for the Triplet schedule. To support readability positive numbers are highlighted in grey colour, while negative values are indicated with red.



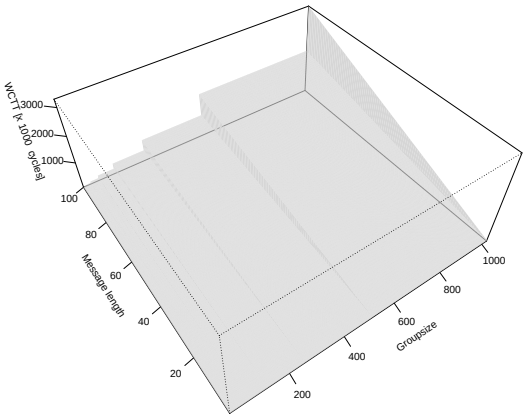
(a) Minimal WCTTs in a NoC with the size $N = 4x4$



(b) Minimal WCTTs in a NoC with the size $N = 8x8$



(c) Minimal WCTTs in a NoC with the size $N = 16x16$



(d) Minimal WCTTs in a NoC with the size $N = 32x32$

Figure 5.24: The difference in terms of WCTTs between the Triplet and the One-to-One TDM schedule for the allgather communication. The results are given as a function of the group size and the number of flits and each diagram displays the differences for a different size of the NoC. Thereby, for each value the best time revealed for the One-to-One schedule is subtracted from the best time of the Triplet schedule. Hence, positive numbers indicate a lower WCTT for the One-to-One schedule while negative numbers show a better performance for the Triplet schedule. To support readability positive numbers are highlighted in grey colour, while negative values are indicated with red.

6

Timing Analysis of MPI-based Parallel Programs

In previous sections we presented techniques and formalisms to enable the calculation of an end-to-end WCET bound for MPI operations. As these operations mark all communication among the processes of a parallel MPI program, they can be seen as building blocks of a parallel program. Thus, we can base on the presented mechanisms to provide an end-to-end worst-case timing upper bound for a complete MPI program. Thereby, we use the calculations of the operations' timing bounds in a modular way. We assume each process participating in a communication needs the time that is returned by the formalism for executing the communication. We also show the application of the procedure described below by providing a timing analysis at a case study.

In this chapter we firstly state the assumptions that we use to build on the analysis and present the overall workflow of the analysis of a parallel MPI program. Afterwards, the procedure is presented that provides the calculations to obtain an program's timing upper bound. Finally, there is a case study that illustrates the application of the presented workflow and procedure and evaluates the impact of design choices of the previous chapters.

6.1 Assumptions and Workflow for Parallel Programs

As stated in Figure 6.1 the analysis of a MPI program is partitioned in four different challenges. The first of those challenges is the separation of parts with pure local computation and parts with pure communication tasks. The next challenges obtain the timing bounds of each of the identified tasks in the computation and communication parts and the last challenge combines the obtained timing to a full end-to-end timing upper bound for the parallel MPI program.

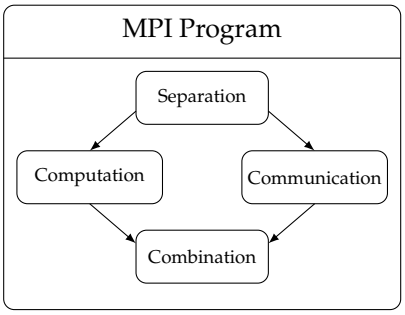


Figure 6.1: Workflow for timing analysis of parallel MPI programs. The workflow is split in four different phases. The rectangles mark these phases while the arrows mark precedence constraints.

code section id	starting point	end point
1	/* A */	/* B */
2	/* C */	/* D */
3	/* E */	/* D */
4	/* E */	/* F */

Table 6.1: Separation of code instances on application level. Starting points and end points refer to comments of Code example 6.1.

When putting the focus to the separation of computation and communication, the call and execution of a MPI operation corresponds directly to one communication phase. As illustrated in Code example 6.1 and Table 6.1, the transition between the phases is directly before calling such a MPI function and directly after returning from it. In that way local computation sections start either at the beginning of the program execution or directly after returning from a MPI function and end either at the end of the program or before a call to a MPI function. Besides defining the points of transition, there is a need to identify all processes participating in the communication that is instantiated by the regarded operation. The information about the participants is needed to calculate the communication's timing behaviour (see Chapter 5) and to put the timing of the operation in the context of the whole program under analysis. The participants can be obtained once from the communicator that is handed to a communication operation and by the control flow of each process. For collective communications the communicator handed to the operation directly leads to the group of processes that is participating in that communication. When regarding to point-to-point communication solely considering the communicator is not sufficient. The reason is that there may participate only a subset of the communicator's group of processes in the actual communication. Hence, the control flow of the execution of each of the processes must be considered to reveal the processes that are actually taking part in the communication. Thereby, each process that is executing the according MPI function performs a communication and is a participant of the communicator that is given as parameter of the function.

After the examination of local computation of the processes and their communication

Code example 6.1 Small pseudocode example for illustration of code section separation on application level. The comments mark points of separation.

```
example_program() {
        /* A */ ...
        /* B */
        MPI_Bcast(...);
        /* C */ ...
        for(all iterations) { ...
                /* D */
                MPI_Gather(...);
                /* E */ ...
        }
        ... /* F */
}
```

behaviour among each other, we must reveal the worst-case timing bound of each part of the program. In case of local computations we assume that state-of-the-art techniques and tools exist that enable the calculation of the WCET of the corresponding part of the program execution (cf. Section 2.4). Thus, we do not further regard how the WCETs for local computation parts are reveal but assume that they are available for the analysis of the MPI program.

While there are tools that support the calculation of the local computation's timing, the timing for communication must be calculated according to the formalism presented in Chapter 5. The formalism can be applied as a function of different characteristics of the hardware architecture and the application. Examples for characteristics regarding the application are the number of participating processes and the size of the transferred message. On the other hand, an example for a characteristic regarding the hardware is the number of nodes in the considered NoC. These parameters must be identified to obtain the actual timing of the communication operation. Typically, the hardware parameters can easily be identified by considering the targeted hardware platform and are equal for all communication phases and the considered architecture. In contrast, the application characteristics may be different for each execution of a communication operation. Thus, they must be identified separately for each communication phase by investigating the application's source code structure and its control and data flow.

After estimating the timing of the local computation and the communication parts it remains the correct assembling of all times to gain the end-to-end timing of the analysed parallel program. Thereby, the procedure bases on the revealed timing bounds of the sections of local computation and communication among the processes. Details about that procedure are provided in the subsequent Section 6.2.

Furthermore, all timing bound parts are statically analysed and assure the worst-case preconditions and prestates. Thus, two equal code cunks that are executed in different processes still exhibit the same static timing upper bound. Moreover, all processes participating in the analysed communication are assumed to begin their execution of the operation in the same point of time. This is a logical assumption to ensure the consideration of the worst case and leads to consequences when applying the analysis of a complete parallel program.

As the processes are typically reaching a MPI operation not exactly at the same time, it must be assumed that a collective starts only at the time when all participants have finished their previous work. Thus, waiting times are assumed in the analysis, that actually may not be present. We see it as future work to reduce this kind of overestimation.

6.2 Assembling Phase

In the first phase we separated the computation from the communication parts and also determined which processes take place in which communications. Hence, after this phase we have a parallel execution structure of the program under analysis.

According to that execution structure it is possible to calculate the end-to-end timing upper bound of the parallel program. Thereby, the principle procedure considers the timing of each process separately and uses the end-to-end timing of the process that is maximal for all processes as result for the complete program. The calculation of the time for each process is performed according to the execution structure. Thereby, we provide a set of components (each component is a local computation or a communication) that is ready to consider for accumulating it to the timing of their participating processes. A component is ready if all participants of that component have accounted all previous components that influence the timing of the regarded participant process. Then considering a component is done by adding up the WCET of the component to the timing bound of each participating process. At that point, the handling is completed and the set of components that are ready for being handled must be updated. After all components are regarded, the maximum of the timing bounds of all processes marks the end-to-end WCET of the MPI program.

According to the described procedure, we provide an algorithm in pseudocode to illustrate the principles of this procedure. Thereby, we present the algorithm in three parts. At first, the applied data structures and function prototypes are presented. Afterwards we show the algorithm for the actual calculation and finally the update process is described that manages the list of components which can be applied next in the calculation algorithm.

Each component of a parallel program is assigned with several characteristics as indicated in Code example 6.2. At first, the data structure of a component must encode the analysed execution structure of the parallel program. Hence, each component owns a list of process ids named `participants` which includes each process that executes the current component. Thereby, this list can comprise one or more processes depending if it is a local computation part or a communication part. Next to the processes participating in the component there is a list of other components that are executed directly before the execution of the currently regarded component. Similar to the participants this list may include one or more elements according to the type of the current component. The third element is the WCET upper bound of the component. This value is obtained by applying the analyses in previous chapters or by running a tool for obtaining the WCET bound of sequentially executed code. Finally, the last two elements are flags for management in the subsequently presented algorithm. They indicate if the regarded component is ready to be considered in the calculation or if it is already handled. In addition to the data structure of a component we need the number of processes executing the parallel program and a list of all components of the program. Furthermore, the list of all active components is hold in the variable `active_comps`.

Code example 6.2 The data structures, global variables and function prototypes of the presented algorithm. The data definitions comprise the type definition of a component, the list of all components of the investigated MPI program and a list of currently active components. The prototypes list a function for the overall calculation of the timing upper bound and a function for updated the set of active components.

```
struct component {
        int *participants;
        comp_t *pre_components;
        int wcet;
        bool is_active;
        bool is_done;
} comp_t;

int numProcs;
comp_t components[] = { all components of application };
list<comp_t> active_comps;

int calcProgWCET(void);
void update_actives(void);
```

The execution flow of the algorithm is presented in two functions. The first function is called `calcProgWCET()` and performs the actual calculation of the end-to-end timing upper bound and function `update_actives()` manages the list of currently active components. Thereby, we see an active component as a component that can be regarded for the timing calculation next.

The function `calcProgWCET()` of Code example 6.3 provides the WCET upper bound of each process of the parallel program in an array `wcet[]`. Before the actual calculation starts there is an initialization phase. Within this phase the initial elements of the list of active components is set. The components that have an empty list of previous components belong to this list. Thereby, previous components mean that those components are executed directly at the beginning of the program's execution.

After the initialization there is a loop that runs as long as there are elements in the list of active components. Within one iteration the algorithm takes one active element (in Code example 6.3 it is the first element) and handles this element for the timing analysis. A component is handled in that the WCET values of all participant processes is set to the maximum time of this set of processes. Afterwards, the WCET of this component is added to the WCET of each participating process (stored in the array `wcet[]`). After the handling the component is removed from the list of active components and the component's flags are set accordingly. Finally, the list of active components must be updated, since there may be some components that become active after the handling of the current component. As already mentioned the algorithm terminates as soon as there are no more active components left in the list. Assuming that the program under analysis is correct and the program's execution structure is encoded in the components correctly, this condition also marks that all executed components of the parallel program are handled.

Code example 6.3 The algorithm for the calculation of the end-to-end timing upper bound of a MPI program. It comprises an initialization phase and a calculation phase. The first phase sets the values of the applied variables and structures to an initial state and the second phase is responsible to reveal the timing upper bound of the program under investigation.

```
int calcProgWCET() {
        int wcet[numProcs] = 0;

        /* init phase */
        forall(comp_t c in components) {
                if(c.pre_components.is_empty()) {
                        c.is_active = true;
                        active_comps.append(c);
                } else {
                        c.is_active = false;
                }
                c.is_done = false;
        }

        /* calculation phase */
        while(!active_comps.is_empty()) {
                comp_t act = active_comps.getFirst();
                int max_wcet = 0;
                forall(comp_t par in act.participants) {
                        if(max_wcet < wcet[par])
                                max_wcet = wcet[par];
                }
                forall(comp_t par in act.participants) {
                        wcet[par] = max_wcet + act.wcet;
                }
                active_comps.remove(act);
                act.is_active = false;
                act.is_done = true;
                update_actives();
        }

        return max(wcet[*]);
}
```

Code example 6.4 The algorithm for updating the list of currently active components. It iterates over all components of the examined MPI program and checks the current state of each one. According to that state a component is either set active and appended to the list of active components or skipped for that handling.

```
void update_actives(void) {
        forall(comp_t c in components) {
                if(!c.is_done && !c.is_active) {
                        bool active = true;
                        forall(comp_t pre in c.pre_components) {
                                if(!pre.is_done) {
                                        active = false;
                                }
                        }
                        if(active) {
                                c.is_active = true;
                                active_comps.append(c);
                        }
                }
        }
}
```

At the end of the algorithm there is a WCET upper bound for each process of the parallel program. Thus, the maximum of all values must be calculated and returned to provide the end-to-end timing upper bound for the complete parallel program.

Code example 6.4 describes in pseudocode the update process of the list of active components. It is located in function `update_actives()` which is called in `calcProgWCET()` in Code example 6.3. For the update of the active components the list of all components is iterated. Thereby, only components that are not handled at the time of the update are considered. For already completed components we continue with the next iteration.

The handling of one component is twofold. At first, there is a check of all components that must be executed directly before the current component. This check considers if all previous components are already handled in the calculation. If this is the case the current component can be considered as active and will be handled accordingly. Otherwise the component can not be considered as active. The handling of a component to become active marks the second part of the update process. In this part the active flag of the current component is set and it is appended to the list of active components.

The presented algorithm shows the principle workflow of the calculation. However, there are some special situations that must be regarded. Some sections are often executed multiple time in the program. Hence, we are mainly talking about the bodies of loops and the execution of functions here. Due to this characteristic it may be feasible to analysis them separately in the same way as a complete program and use them as a single component afterwards. However, their timing behaviour may change according to the current execution and the input used for the component. Hence, the worst-case estimation typically overestimates the current execution. Thus, it is preferable to provide the correct input values for each execution of such an instance. An example could be to upper bound the length of a message that is sent within such a component according to the position where the function is called

or the actual iteration executed. These input parameters are similar to the parameters that must be provided for the analysis of the MPI operations. Overall, the consideration of this issue leads to a hierarchical view on the parallel execution control flow.

For point-to-point communications we must regard to another issue. Here it is possible that the send and the receive operation are located in different parts of the source code. However, this is an issue of the separation phase where the correct computation section must be reveal and then analysed correctly.

6.3 Case Study: Parallel LU Calculation

In this section we provide an example application of the previously presented procedure of analysing a parallel MPI program. For that we focus on a program that is provided by various benchmark suites that focus on parallel computing. It is the *Lower-Upper symmetric Gauß-Seidel (LU)* benchmark which is part of the NAS parallel benchmark suite[1] [Bai+91; Bai+95] and a kernel benchmark of the SPLASH-2 benchmark suite [Woo+95; BKK08]. This benchmark performs a factorisation of a dense matrix in a lower and an upper triangular matrix [Woo+95]. According to Asanovic's classification of algorithms [Asa+06] this problem is a representative of the class of algorithms that solve problems with dense matrices. Frieb [Fri19] already considered this benchmark in terms of real-time. However, he only provides an estimation of the WCET's order of magnitude, while we provide a more detailed analysis of this benchmark. Furthermore, the focus of Frieb's analysis is different to ours. He focusses on evaluating developed hardware extensions for the applied many-core platform, while we want to show the application of our formalisms and compare the communication patterns presented in Section 2.3.

For the timing analysis we ported the LU benchmark of the SPLASH-2 suite to MPI and located the construction of all needed communications to the initialization, since these constructions do not fit to a WCET analysis because of the need for memory allocations. In the execution of the calculation a look up to the desired communicator is performed for the communication operations.

6.3.1 Algorithm Structure

The algorithm for the LU calculation divides the quadratic input matrix M of size $m \times m$ in multiple blocks $b \in B$ that are used as basic workload packages for the working processes $v \in V$. Thereby, each block b is a quadratic submatrix of the input matrix and includes $e \times e$ elements. The block can be arranged in a matrix like order to build the complete input matrix M. Thus, we refer to a particular block with the symbol $b_{i,j}$. The indices indicate the position of the block and define the i-th block row and the j-th block column. However, if the number of elements per dimension in the matrix m is not an integer multiple of the number of elements per dimension in a block e, there are some blocks that are only partly filled. Hence, these blocks do not exhibit a quadratic shape. They are located at the end

[1] NAS parallel benchmarks - `https://www.nas.nasa.gov/publications/npb.html`[Online, last accessed on 6th Dezember 2019]

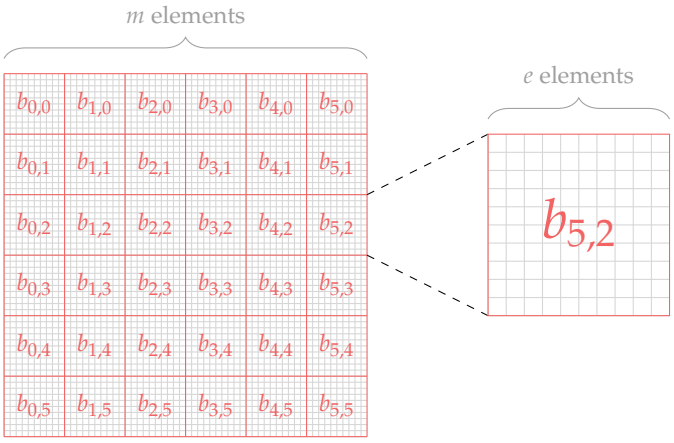


Figure 6.2: The block shape of the LU calculation's input matrix. The red squares and symbols mark the blocks and the according naming convention. The grey squares indicate single elements of the matrix.

of each row and column of blocks. Figure 6.2 illustrates the division of the input matrix to blocks.

Based on the presented structure of the matrix we subsequently describe the principle workflow of the calculation of the triangular matrices. Thereby, Figure 6.3 illustrates the presentation of the different calculation phases.

Overall the workflow is based on the blocks b of the matrix that act as basic workload packages for the algorithm. With the granularity of the input data in mind the algorithm can be hierarchical classified in two levels. On the higher level each step focuses on the calculation of one row and one column of blocks b. Thereby, that row and column of blocks are arranged in a way that both share the leftmost and utmost block. Subsequently, we refer to the set of blocks that are calculated in the i-th iteration with the symbol B^i and further distinguish the sets of blocks of the currently calculated row and column with the symbols B^i_{row} and B^i_{col}. Accordingly, the size of these sets are given with the symbols $|B^i|$, $|B^i_{row}|$ and $|B^i_{col}|$. In Figure 6.3 the set B^i is indicated with all grey areas of blocks in a certain iteration and B^i_{row} and B^i_{col} are marked with the grey areas in the third step of an iteration. Please note that we exclude the block that is located in the row and column at the same time (see grey block in step 2) in B^i_{row} and B^i_{col}.

On the other hand the lower level provides several steps needed for the complete calculation of the blocks in the regarded row and column. Furthermore, a preparation of the blocks that remain for calculation in subsequent iterations takes place. Subsequently, we refer to steps of the higher level as iterations and use the term step for a step of the lower level, for an easier distinction between the steps of the higher and the lower level.

The calculation of one iteration is performed with three steps which are separated via global barriers. Please note that there is no need for a barrier between two iterations (see Figure 6.3). The first step calculates and updates the values of the block that is shared between the row and column. Within this step only the process is working that owns this block and all other processes are waiting at the barrier to the next step.

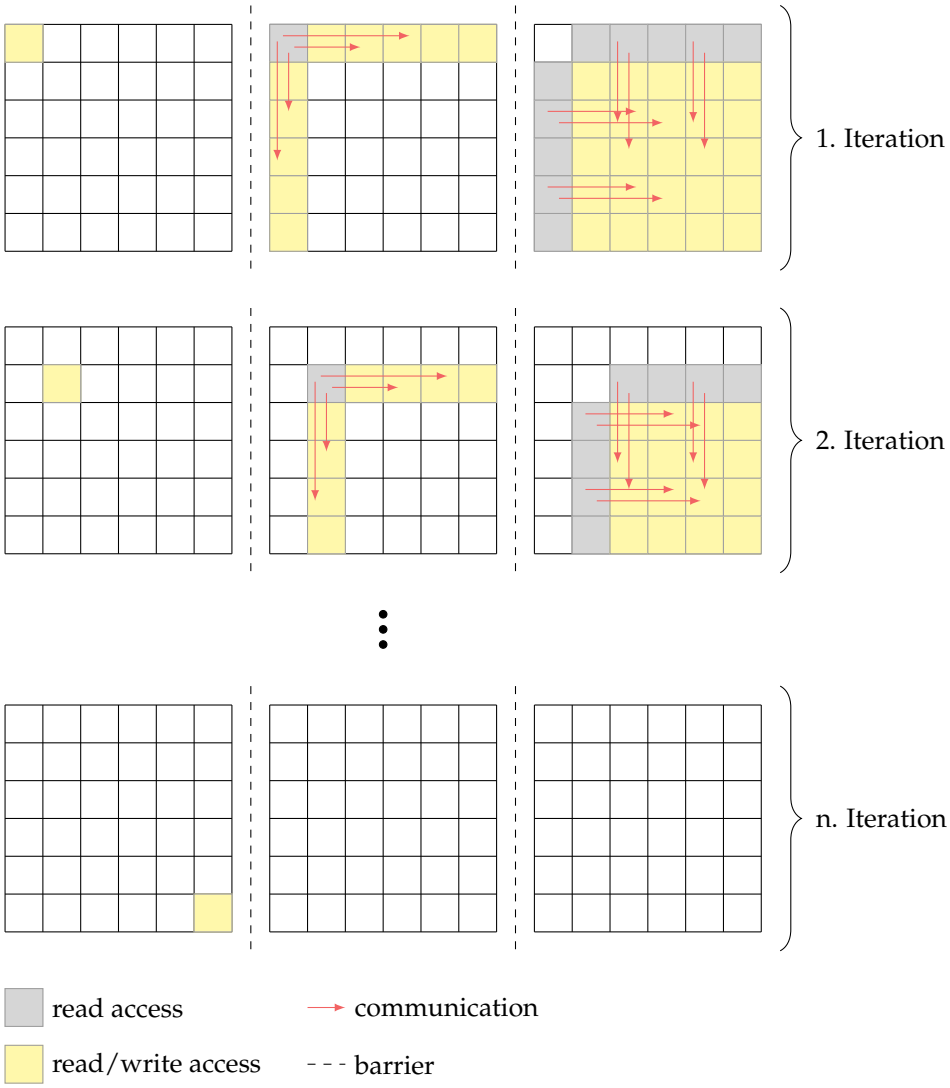


Figure 6.3: The overall workflow of the execution of the LU calculation. The depiction is based on blocks b which act as basic workload packages for the algorithm. Each row displays the calculation of one row and column of blocks b (one iteration) and each column marks one step within this calculation. The accesses to the data in each step are marked with gray and yellow colours. Barriers that separate different steps are indicated with dashed lines and red arrows indicate performed communication. Please note that the communication is only indicated and thus, only a part of the actual communication is displayed.

When the calculations of the first step have been finished all processes meet at the according barrier and the second step begins. The first action of this step is to send the block that was updated in the first step to all processes that are responsible for one or more blocks that are calculated within the current iteration. This is done via a single broadcast communication. Afterwards all blocks (except the shared block) of the regarded row and column are locally calculated and updated. The processes that finished their work wait at a barrier which separates the current step from the next step. At the end of this step the elements of the blocks B^i regarded in the current iteration i show the correctly calculated results for the targeted triangular matrices.

However, there is one more step left that must be performed in that iteration. This step focuses on the preparation of all remaining blocks to provide a suitable starting point for performing the next iteration. Like in the second step the third step is divided in a communication phase and a calculation phase. At first, the updated blocks are shared with the blocks that are not calculated yet. Thereby, each handled block is sent to the block owners of the lower blocks located in the same column or to the right-hand blocks of the same row depending if the considered block belongs to the row or the column of blocks calculated in this iteration. Afterwards all remaining blocks are updated according to their own values and the values of the received blocks. This finalizes the third step and equally completes the current iteration of the program.

The row and column of the next iteration is located one block below the row of the last iteration and one block right of the last iteration's column. Hence, the block that is calculated in the first step of an iteration is always the block that covers the principle diagonal of the matrix. As already mentioned successive iterations are not separated with a barrier. Hence, the processes directly start working at the first step of the next iteration after finishing their work at the previous iteration.

According to the algorithm structure there are as many iterations as there are blocks in one row or column of the matrix. Thereby, the last iteration calculates the rightmost and undermost block of the matrix. As there are no further blocks left to calculate afterwards, no work happens in the steps 2 and 3 of this last iteration.

Typically, there are more blocks available than processes that are working in parallel to solve the problem. Thus, each process handles the calculations of multiple blocks. According to this fact the blocks must be mapped to the processes in a way that the workload is nearly equally distributed among the processes. In the algorithm at hand a block-cyclic distribution is chosen to fulfil this requirement. A block-cyclic distribution causes the blocks to be allocated to the processes in a cyclic manner.

The concrete distribution of the block to the processes is best described at an example. For that we assume a parallel execution with four processes and a distribution of six blocks per row and column of the matrix. Then the mapping of processes to blocks is arranged as displayed in Figure 6.4.

The block-cyclic mapping of blocks to processes causes fixed numbers of processes that are involved in updating the blocks of one row or column. Thereby, we refer to the sets of processes responsible for a row or column in a similar way like with blocks. Generally, we indicate a set of processes of a row with the symbol V_{row} and a set of a column with V_{col}. Though the sets of different rows or column may differ, we do not concretise the

v_0	v_1	v_0	v_1	v_0	v_1
v_2	v_3	v_2	v_3	v_2	v_3
v_0	v_1	v_0	v_1	v_0	v_1
v_2	v_3	v_2	v_3	v_2	v_3
v_0	v_1	v_0	v_1	v_0	v_1
v_2	v_3	v_2	v_3	v_2	v_3

Figure 6.4: The mapping of blocks of the LU calculation's input matrix to processes. Each square indicates a block b of the input matrix and its position corresponds to the position of the block in the complete matrix. The different colours illustrate the four different processes. Thereby, each block is marked with the colour of the process that is responsible for its calculation. In addition the symbols v_i indicate the according processes as well.

various sets, as for the timing analysis of this benchmark we only need to refer to the size of these sets which is the same for all row sets or column sets, respectively. According to the applied mapping the number of processes responsible for one row or column are given in the following equations.

$$|V_{row}| = \left\lceil \sqrt{|V|} \right\rceil \tag{6.1}$$

$$|V_{col}| = \left\lfloor \sqrt{|V|} \right\rfloor \tag{6.2}$$

Please note that in the case if $|V_{row}| \cdot |V_{col}| < |V|$ there are $|V| - |V_{row}| \cdot |V_{col}|$ processes that do not participate in solving the mathematical LU problem and only wait at the barriers for the completion of the other processes.

6.3.2 Real-Time Analysis

Subsequently, we present the different aspects considered for the WCET analysis. With that knowledge at hand it is possible to calculate the end-to-end WCET upper bound for the LU benchmark. At first, we need to extract the execution workflow of the program. For illustration purposes we provide an overview of such a workflow at the example of 4 running processes in Figure 6.5.

As we described in the previous Section 6.3.1 the execution of the benchmark is dominated by a global loop that calculates one row and one column in each iteration. Within each iteration there are three steps of different calculations which are separated via global barriers. Hence, it is possible to calculate the WCET upper bounds for each execution part between two barriers separately instead of extracting the critical execution path of the whole program. Figure 6.5 represents this structure with the depicted overall loop and the three steps within the loop body. As also illustrated each process performs some work before entering the loop.

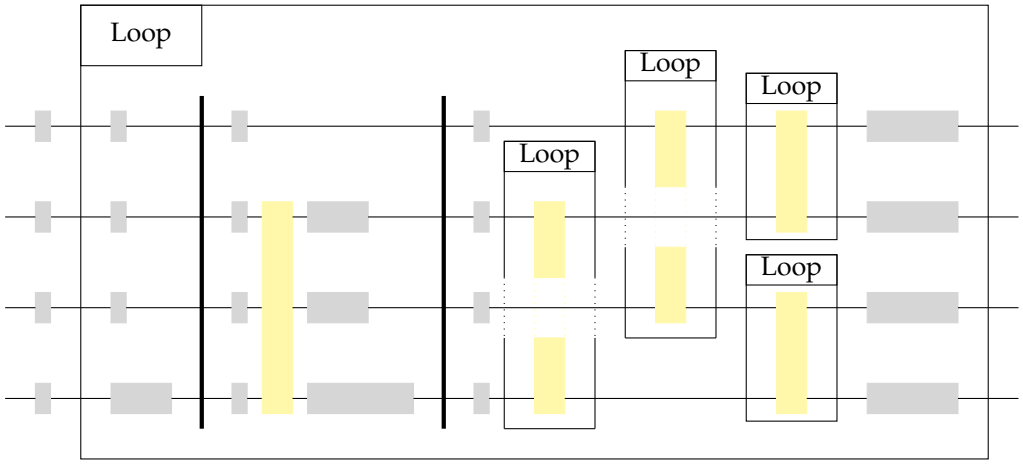


Figure 6.5: The execution workflow of the LU benchmark when it is executed with 4 processes. The time goes from left to right and each process is illustrated with a horizontal solid black line. Loops are indicated with rectangles that enclose their loop bodies and are marked with the label *loop*. Thick vertical solid black lines display global barriers. Grey rectangles indicate local computation of a process and yellow rectangles show performed broadcast communication operations.

In the first step each process performs some initialization work. Additionally, one process performs calculations on the block that is updated in this step. According to the function name of the calculation in the benchmark we refer to it as `lu0()` calculation. After finishing their work each process waits at the barrier to step 2 for the remaining processes that are still working. Thus, the timing of a process in the first step is composed according to the following equation.

$$WCET^{LU,s_1} = WCET^{s_1,init} + WCET^{lu0} + WCET^{barrier} \tag{6.3}$$

After passing the barrier the second step is executed. At first, there is some initial work and then a broadcast communication follows. The group size of this communication operation depends on the set of blocks B^i that are calculated within this iteration i and the distribution of the blocks to the running processes. Thereby, the maximal number of processes is determined by the number of responsible processes per current row ($|V_{row}|$) and column ($|V_{col}|$). However, if the number of blocks to calculate per row or column is lower than the number of processes for a row or column, the number of processes participating in the communication is smaller as well. Thus, the group size for this communication is given as follows.

$$\chi_{s_2}^{bcast} = 1 + \min(|B_{row}^i|, |V_{row}|) + \min(|B_{col}^i|, |V_{col}|) \tag{6.4}$$

The group size of Equation 6.4 is an upper bound for the participating processes that is revealed with the summation of the sender process and the number of processes responsible for the row and the column. Finally, in this step the processes update their blocks based on the previous values of a block and the data received at the broadcast communication. Subsequently, these updates are called `bdiv()` for the update in the current row and `bmodd()` for the update in the current column.

To obtain the timing upper bound of the program the maximal number of blocks that has to be updated by one process must be known. This numbers are revealed by dividing the number of updated blocks per row and column through the number of processes applied for a row or column. Subsequently, we identify the number of blocks that are updated at maximum by one process with the symbols b^v_{row} for the current row and b^v_{col} for the current column. Their values are determined with the following equations.

$$b^v_{row} = \left\lceil \frac{|B^i_{row}|}{|V_{row}|} \right\rceil \tag{6.5}$$

$$b^v_{col} = \left\lceil \frac{|B^i_{col}|}{|V_{col}|} \right\rceil \tag{6.6}$$

Since in the worst case a process participates in the update procedure of the row and the column, both timing upper bounds must be accumulated for the overall timing upper bound of the current step. In addition to updating the blocks the other aspects must be equally considered for the end-to-end WCET bound of step 2. Equation 6.7 shows the maximum timing bound of the second step of the algorithm.

$$\begin{aligned} WCET^{LU,s_2} = {} & WCET^{s_2,init} + WCET^{bcast}(\chi = \chi^{bcast}_{s_2}) \\ & + b^v_{row} \cdot WCET^{bdiv} + b^v_{col} \cdot WCET^{bmodd} \\ & + WCET^{barrier} \end{aligned} \tag{6.7}$$

The third step of an iteration is responsible for updating the still not fully calculated part of the matrix. Similar to the second step it can be subdivided in three parts. At first, there are some local calculations in each process to prepare data structures and communication groups for the communication part which follows next. The communication part is characterised by several broadcast communications among different groups of processes which can partly run in parallel. The last part of the third step is again local computation. However, this time the computation is executed to update all blocks of the matrix that are not fully calculated yet and are executed by the processes that are mapped to the corresponding blocks. Therefore, we need to reveal the number of performed broadcasts and the sizes of their groups as well as the maximal number of blocks that one process must update to enable the calculation of a timing upper bound for the third step.

Altogether, all blocks of B^i_{row} and B^i_{col} that are updated in step 2 must be sent to other the blocks of either their row or column. Thus, there are $|B^i_{row}| + |B^i_{col}|$ broadcasts in total. However, the decisive number for the analysis is the number of broadcasts that must be executed in sequential by one of the processes because this number determines the critical path. At first, it is to notice that broadcasts which deliver data along a row of blocks and broadcasts that are used for the communication along columns must be assumed as sequentially executed, since each group of a row communication also includes group members of all column communications and vice versa. Furthermore, we see that for each different source process of the broadcasts along a row there is a group of processes that is disjoint to all groups of communications along a row that exhibit different source processes. The same holds true for broadcasts along a column of blocks.

According to the insights we need to determine the maximum number of broadcasts that one process has to perform as a source process, once for horizontal communications

and once for vertical communications. Since the broadcast communications correspond to the blocks calculated in this iteration, we already determined the requested number. It is provided in Equations 6.5 and 6.6. In addition to these numbers we need the group sizes of the communications. These numbers are the numbers of processes that own a block from the same row or column and are given in Equations 6.8 and 6.9.

$$\chi^{bcast}_{vert,s_3} = \min(|B^i_{col}|, |V_{col}|) \tag{6.8}$$

$$\chi^{bcast}_{hori,s_3} = \min(|B^i_{row}|, |V_{row}|) \tag{6.9}$$

After the communication the last part to execute in step 3 is the computation of the updated values for the remaining part of the matrix. This is a local computation and according to the benchmark it is called `bmod()`. It must be executed for each of the remaining blocks. With respect to the block-cyclic mapping of processes to blocks we need to reveal the maximum number of blocks that are handled from one process. Therefore, we consider the number of blocks that one process must serve in maximum in one row and in one column. Afterwards, these results are applied to obtain the maximal total number of blocks for one process. These numbers are the same as the number of broadcasts that must be performed for one process. Hence, we can use Equations 6.5 and 6.6 to calculate the desired numbers for a row or column, respectively. Finally, the total number of processes is calculated with a multiplication of both values.

To summarize all aspects for the timing analysis of the third step we can state the following equation to compute the timing upper bound for this step.

$$\begin{aligned} WCET^{LU,s_3} = {} & WCET^{s_3,init} \\ & + b^v_{row} \cdot WCET^{bcast}(\chi = \chi^{bcast}_{vert,s_3}) + b^v_{col} \cdot WCET^{bcast}(\chi = \chi^{bcast}_{hori,s_3}) \\ & + b^v_{row} \cdot b^v_{col} \cdot WCET^{bmod} \end{aligned} \tag{6.10}$$

In Equation 6.10 the first line represents the first part of step 3, while the second line describes the second part and line 3 is for the last part of this step. As mentioned above in the second part there are several broadcasts executed in sequential by at minimum one of the processes. These are b^v_{row} for the vertical communications and b^v_{col} for the horizontal communications. The product of both values b^v_{row} and b^v_{col} reveal the maximum number of blocks that must be handled by one process in the current step and iteration and is used in line 3.

The overall timing upper bound of the LU calculation is determined by considering each step in all iterations. Additionally, there is some initial computation before the overall loop begins. This computation is local to all processes and performs some declaration and initialization of variables that are needed for the subsequent execution. Hence, the overall timing upper bound of the program is given in the following equation.

$$WCET^{LU} = WCET^{init} + \sum_{i=1}^{b} (WCET^{LU,s_1}_i + WCET^{LU,s_2}_i + WCET^{LU,s_3}_i) \tag{6.11}$$

We provide a timing analysis according to the WCET upper bound described and compose them to the end-to-end timing upper bound of the program as presented. However, there are the following small adaptations done in the actual analysis. Since there is no communication or synchronisation among the processes until they reach the barrier between step 1 and step

2, we analyse the timing bound of the code executed before entering the loop and the first step of the first iteration in one draft and use for the remaining iteration another timing bound for the first step. This different timing bound is analysed as the composition of the third step of the previous iteration and the first step of the current iteration. Furthermore, there is a second timing bound provided for the last step as well. This is needed for the last iteration of the algorithm which only regards to this step instead of composing it with the first step of the next iteration.

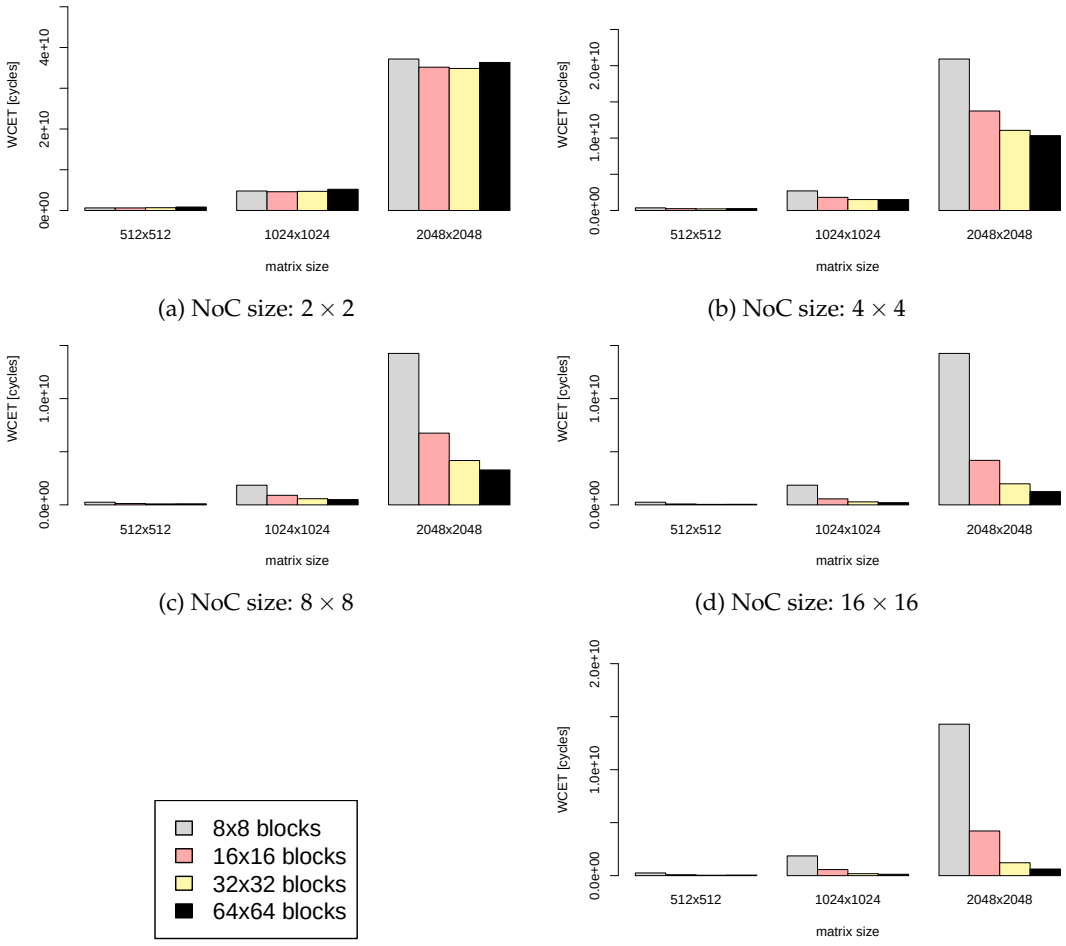
6.3.3 Evaluation

The results of the timing analysis of the LU benchmark are discussed in detail in this section. Thereby, we focus on the worst-case timing behaviour of the program when different input sets are applied and examine the influence of different communication patterns applied for the broadcast and the barrier communication operations. The total WCET upper bound, the time needed for the communication and the speed-ups of the parallel versions are depicted and discussed.

At first, we focus on different input sets for the execution of the benchmark. Three sizes for the input matrix are provided and for each matrix four different numbers of blocks $|B|$ are assumed. The different matrix sizes comprise 512×512, 1024×1024 and 2048×2048 elements. They approximately reflect the intermediate input sizes (Class A, B and C) of the LU benchmark of the NAS parallel benchmark suite. The investigated number of blocks are 8×8, 16×16, 32×32 and 64×64 blocks. We calculate the end-to-end timing upper bound of the MPI program for each input and block set and for the execution on different NoC sizes. Similar to previous discussions in this thesis (cf. Section 5.3) we assume the RC/MC architecture [Mis+17] as execution platform for the timing analysis. The investigated NoCs comprise 2×2, 4×4, 8×8, 16×16 and 32×32 nodes. Thereby, we assume that the examined parallel program uses all nodes of the platform for its execution and runs one process on each node. Hence, the number of nodes of a NoC is equal to the number of applied processes.

Figure 6.6 displays the end-to-end WCET upper bounds for different input sets and applied NoC sizes. The calculated WCET upper bounds show a wide variation depending on the investigated configuration. The results for a matrix with 512×512 elements range from $3.98 \cdot 10^7$ cycles to $2.53 \cdot 10^8$ cycles and the 1024×1024 matrix exhibits WCET bounds from $1.28 \cdot 10^8$ to $1.86 \cdot 10^9$ cycles. The largest range is observed for the 2048×2048 matrix. It goes from $6.24 \cdot 10^8$ to $1.42 \cdot 10^{10}$ cycles. As expected the analysis shows that the timing upper bound raises with increased input sets if the rest of the configurations equal.

In addition this obvious examination one can see that in most cases the timing bound for a given matrix and NoC size decreases with a rising number of blocks used for the execution. More blocks means that each single block is of smaller size and thus, the workload can be distributed in a more balanced way among the processes. Furthermore, the timing bounds of step 1 (see Figures 6.3 and 6.5) which are limited to the execution of one process as well as the bounds of step 2 which are limited to the execution with a subset of the applied processes are reduced with smaller blocks. Hence, the processes can be utilized more efficiently when more blocks are applied.



(a) NoC size: 2×2

(b) NoC size: 4×4

(c) NoC size: 8×8

(d) NoC size: 16×16

(e) legend

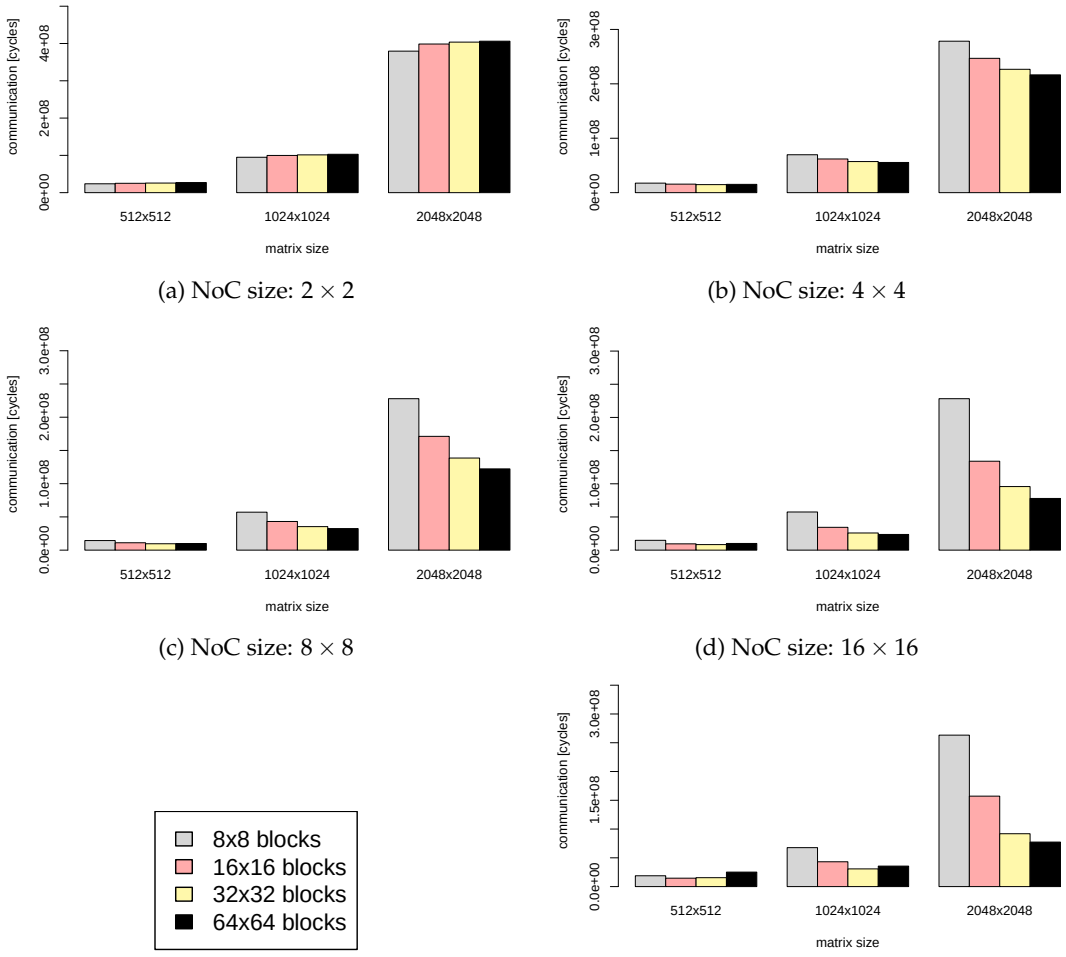
(f) NoC size: 32×32

Figure 6.6: The end-to-end WCET upper bound of the LU benchmark for different configurations of the input set and NoCs. Each figure depicts the execution on one concrete NoC, each with a different number of nodes. The horizontal axis displays the examined input matrices while the vertical axes show the WCET upper bounds in terms of processor cycles. Please note the different scales of the vertical axes of each figure. Each colour relates to a specific number of blocks that is applied for the execution. Figure 6.6e shows the legend for these colours.

However, especially for small NoCs the timing bound can also behave vice versa. When we take a closer look at Figure 6.6a, we can see that the timing bound for the execution with 64×64 blocks raises compared the other numbers of blocks. This is very likely caused by step 3 (see Figures 6.3 and 6.5). The application of smaller blocks causes the function `bmod()` to be executed more often on the same data elements of the matrix, since there are more iterations to proceed until the algorithm finishes. This fact in addition to the slightly higher communication costs in those cases (see Figure 6.7) are contrary to the better utilization of the parallelism. Hence, especially for a small number of processes it is possible that the negative effects of the application of more blocks prevail. However, in most cases a larger number of blocks results in a better utilization of the participating processes and therefore, to a lower timing upper bound. Especially, for the configuration with 8×8 blocks the high WCET bound indicates that the execution is not balanced among the processes. Altogether, there is the observation that with the increase of the input matrix it becomes more important to apply a large number of blocks to enable an efficient execution of the benchmark.

After investigating the total WCET upper bound we take a closer look on the communication of the program. Figure 6.7 depicts the timing upper bounds occupied for communication during the execution of the LU benchmark for all input sets and NoC sizes provided in this discussion. Overall the timing bounds for the communication are located between $1.46 \cdot 10^7$ to $2.52 \cdot 10^7$ cycles for the small input matrix and the intermediate matrix shows a range from $3.08 \cdot 10^7$ to $6.77 \cdot 10^7$ cycles. The large input set needs $7.74 \cdot 10^7$ to $2.63 \cdot 10^8$ cycles for the communication in the worst case. After this general overview we focus on some specific aspects of the communication.

If we only focus on the different input sizes and consider the values for a fixed number blocks and NoC size, it is revealed that the time for the communication increases for raising input sizes. However, this increase shows a smaller gradient as the one of the total WCET bound (cf. Figure 6.6). Hence, in most cases the percentage of the communication to the end-to-end execution of the program decreases. Furthermore, for a fixed input and NoC size typically the communication times decreases with an increased number of blocks. When only varying the number of blocks the total amount of data that is sent during the program execution stays the same. However, the amount of data sent per broadcast decreases, while the number of performed broadcasts increases. Furthermore, the delivery of some data elements is not performed by the broadcast of step 2 any more but is delivered by a broadcast of step 3. Typically, the group sizes of step 3 are smaller than the ones for step 2 (cf. Equations 6.4, 6.8 and 6.9). Thus, in total the time for communication decreases with a raising number of blocks. However, for a small total number of processes or for small input sizes this effect is probably outweighed by the constant costs of each broadcast, because the total number of executed broadcasts increases with the number of applied blocks. This effect can be observed e.g. in Figure 6.7a. Finally, a comparison of all diagrams of Figure 6.7 reveals that the timing upper bounds for the communication are generally relatively stable for all NoC sizes. This is probably caused by the fact that the total amount of delivered data is relatively stable as already mentioned above.



(a) NoC size: 2×2

(b) NoC size: 4×4

(c) NoC size: 8×8

(d) NoC size: 16×16

(e) legend

(f) NoC size: 32×32

Figure 6.7: The timing upper bounds that are calculated for the execution of the communication phases of the LU benchmark. The same configuration of input sets and for execution are depicted as in Figure 6.6. Each figure depicts the execution on one concrete NoC, each with a different number of nodes. The horizontal axis displays the examined input matrices while the vertical axes show the WCET upper bounds in terms of processor cycles. Please note the different scales of the vertical axes of each figure. Each colour relates to a specific number of blocks that is applied for the execution. Figure 6.6e shows the legend for these colours.

In Figure 6.8 we state the reached speed-ups for all investigated configurations of input sets and NoC sizes. For the calculation of these results the WCET upper bounds of the parallel executions are compared to the WCET upper bounds of a sequential algorithm for the LU calculation. This algorithm is commonly known as Doolittle algorithm for LU decomposition.

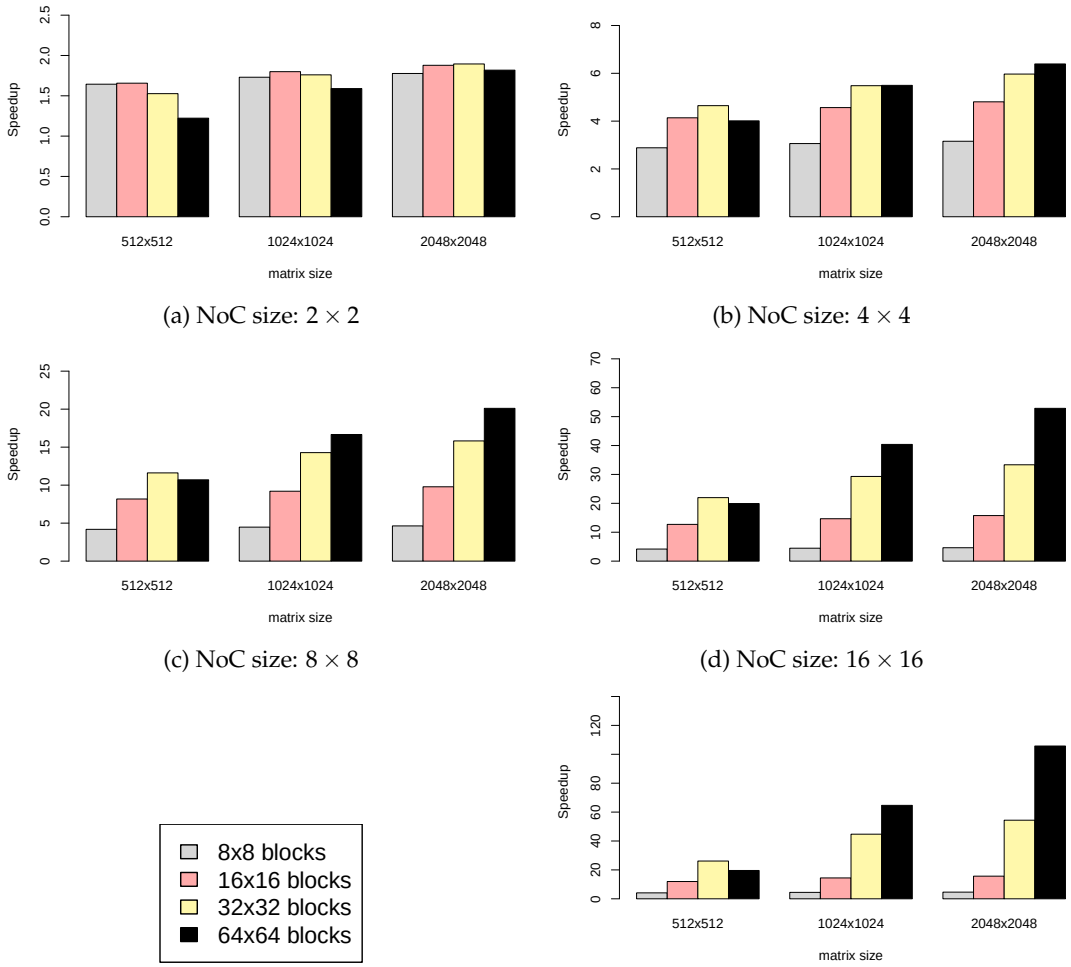
The calculated speed-ups are widely spread depending on the number of processes executing the benchmark, the input size and the number of applied blocks. In the 2×2 NoC the speed-up is located at a range of 1.22 to 1.89, and in the 4×4 NoC there is a speed-up of 2.88 to 6.39. The 8×8 NoC shows speed-ups between 4.18 and 20.11 and the larger NoCs show a even more widespread variance for the speed-up. Here the variances are mainly caused by execution configurations that do not utilize a large amount of the available performance. An example for such a case are the executions with 8×8 blocks in a 32×32 NoC. In these cases the speed-ups are below 5 with is far away from the values that are possible (the speed-up for the execution with 64×64 blocks is 105.72).

Please note that the structure of the algorithm and the according mapping of the workload to the running processes limit the average degree of parallelism to a certain amount. Hence, it is partly caused by the algorithm that it is not possible to fully exploit the possibly reachable performance of the high number of parallel running processes and nodes. This fact is most probably the main reason for the fact that the speed-ups are small compared to the applied number of processes, aka that the efficiency is low.

Typically, the speed-up of a specific input and NoC size increases with a larger number of blocks due to the increased utilization of the parallelism. However, there are also cases where the speed-up decreases. Since the speed-up results are directly related to the total timing upper bounds of the benchmark the reasons for the lower speed-ups are the same reasons as for the higher end-to-end timing bounds of Figure 6.6.

After the examination of different input sets we use a fixed set in the remainder of this evaluation. It is set to an input matrix of 2048×2048 elements and 64×64 blocks are applied. This configuration uses the largest provided input set and the block configuration that shows the best results of all investigated configurations. With this configuration we investigate the behaviour of the benchmark's timing when different communication patterns are applied. Firstly, we focus on the barrier communication and regard to the broadcast communication in later examinations.

Figure 6.9 displays the WCET upper bounds for the LU benchmark when different communication patterns for the barrier operations are used. Similar to previous examinations we provide results for various NoC sizes. When comparing the diagrams of this figure one can see that the timing upper bound decreases as expected with a raising size of the NoC. This effect is not caused by the application of different communication operations but is a subject to a higher degree of utilized parallelism. Hence, we only concentrate on the results for a specific NoC size in this investigation.



(a) NoC size: 2×2

(b) NoC size: 4×4

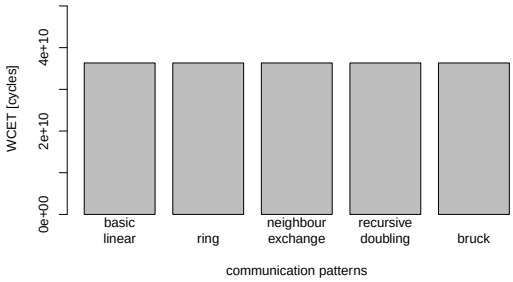
(c) NoC size: 8×8

(d) NoC size: 16×16

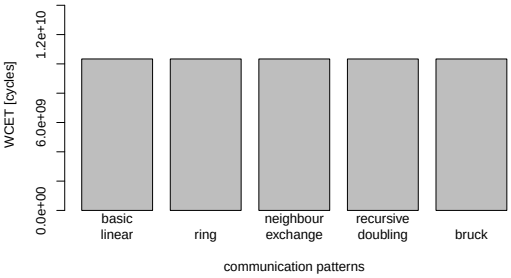
(e) legend

(f) NoC size: 32×32

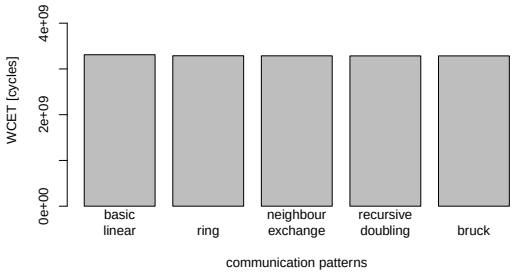
Figure 6.8: The speed-up results that are revealed for the parallel executions of the LU benchmark. A speed-up is provided for each configuration displayed in Figure 6.6. Each figure depicts the execution on one concrete NoC, each with a different number of nodes. The horizontal axis displays the examined input matrices while the vertical axes show the results for the speed-up. Please note the different scales of the vertical axes of each figure. Each colour relates to a specific number of blocks that is applied for the execution. Figure 6.6e shows the legend for these colours.



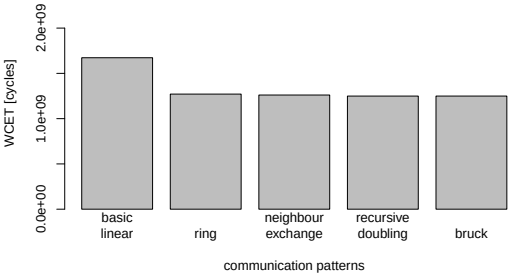
(a) NoC size: 2×2



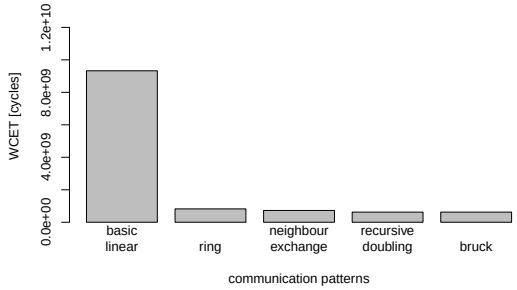
(b) NoC size: 4×4



(c) NoC size: 8×8



(d) NoC size: 16×16

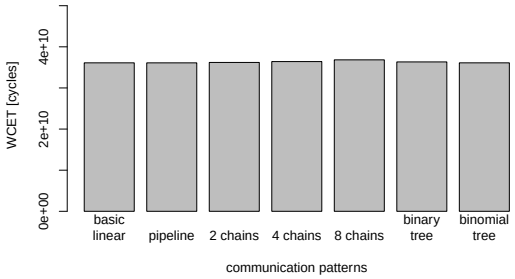


(e) NoC size: 32×32

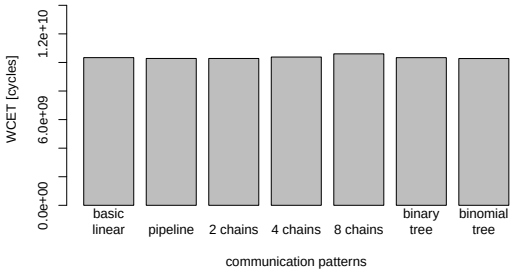
Figure 6.9: The end-to-end WCET upper bounds of the LU benchmark for different implementations of the barrier operation. The results are revealed for a input matrix with the size 2048×2048 elements and the application of 64×64 blocks on different NoC sizes. The horizontal axes show the different communication patterns that are applied for the barrier and the vertical axes display the WCET upper bound in terms of processor cycles. Please note the different scales of the vertical axes.

A comparison of the different communication patterns for the barrier communication shows that there is nearly no difference for the end-to-end timing bound of the benchmark for small and intermediate NoC sizes (up to 8×8 nodes). The reason for this behaviour is the small contribution of that communication to the global timing bound of the program. Each barrier works with an exchange of one synchronisation flit per participating process, which can be seen as a very short message. Thus, the communication is handled very fast compared to the rest of the program execution and there is nearly no difference when applying different communication patterns. This observation also holds true for the large NoCs except for the basic linear version of the barrier communication. With the basic linear barrier each participating process sends a flit to a dedicated central process and receives a flit from that process at the time when flits from all processes has arrived at the central process. Hence, when assuming $n \cdot n$ processes in short there are $2 \cdot n^2$ flits that must be regarded to be sent one after another. This is reasonably more communication to perform than in the other communication patterns and thus effects the total timing bound that much. However, since the number of handled flits is dependent on the NoC size, the described effect is only visible for large NoCs. The other communication pattern show differing timing bounds as well but the differences are marginal compared to the end-to-end timing bound of the program execution. Since the influence of the different barrier communication patterns except for the basic linear pattern is that small we omit the examination of the communication's percentage and the differences in speed-ups.

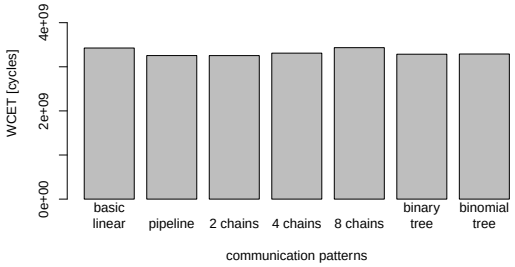
Subsequently, we investigate the timing behaviour of the benchmark when different communication patterns for the broadcast communication are applied. As a first step we focus in Figure 6.10 on the total WCET upper bound of the LU calculation for various NoC sizes. For small NoC sizes there is are only marginal differences when varying the communication patterns of the broadcast operations. This is caused by the large amount of workload that must be executed in each process because of the small number of processes that participate in the parallel execution. Similar to the barrier communication patterns the basic linear pattern exhibits the highest timing bounds for intermediate and large NoCs and shows significant differences to the other patterns. Furthermore, the patterns with 4 and 8 chains show an increased timing bound as well. This difference becomes stronger for larger NoCs and also the binomial tree shows an high timing upper bound when many nodes are applied. The best performances show the pipeline, the 2 chains and the binary tree patterns. This confirms the evaluation of Figures 5.21 and 5.22 because even for the execution with large NoC sizes the groups of processes participating in the broadcasts are of moderate size (cf. Equations 6.4, 6.8 and 6.9). However, the differences among the broadcast patterns are not that large as for the barrier patterns, because all broadcast patterns execute any kind of tree communication whereas the sophisticated communication patterns of the barrier communication are totally different to the corresponding basic linear pattern.



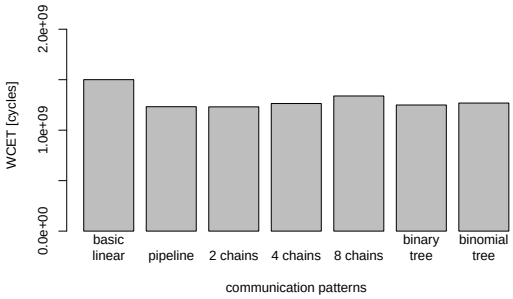
(a) NoC size: 2×2



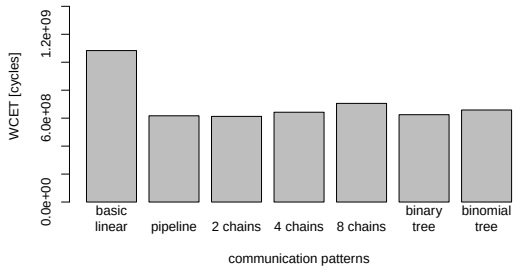
(b) NoC size: 4×4



(c) NoC size: 8×8

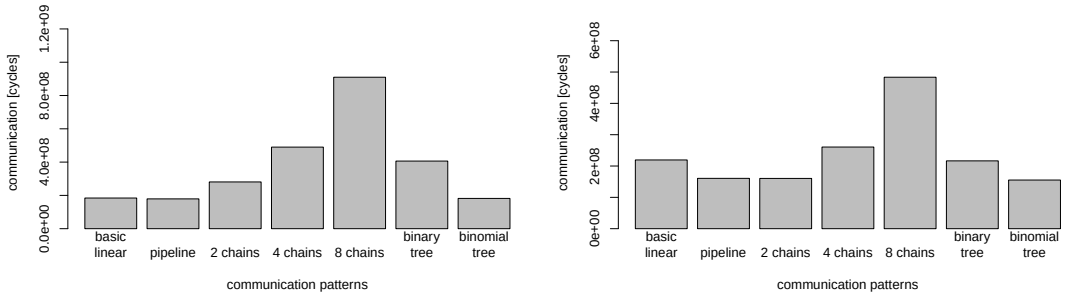


(d) NoC size: 16×16



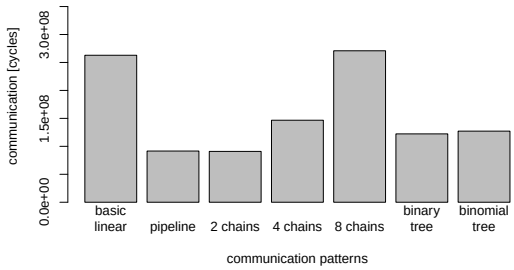
(e) NoC size: 32×32

Figure 6.10: The end-to-end WCET upper bounds of the LU benchmark for different implementations of the broadcast operation. The results are revealed for a input matrix with the size 2048×2048 elements and the application of 64×64 blocks on different NoC sizes. The horizontal axes show the different communication patterns that are applied for the barrier and the vertical axes display the WCET upper bound in terms of processor cycles. Please note the different scales of the vertical axes.

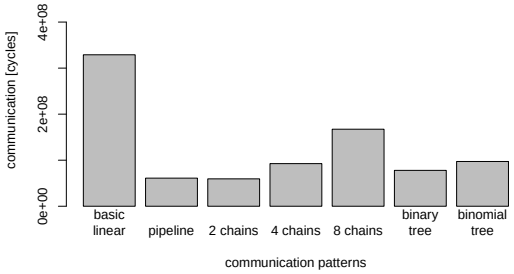


(a) NoC size: 2×2

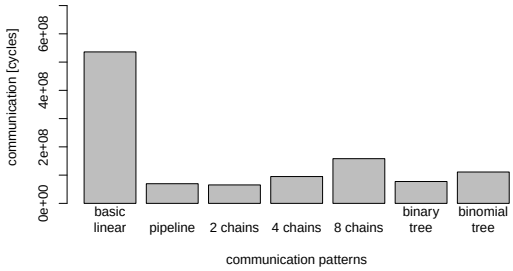
(b) NoC size: 4×4



(c) NoC size: 8×8



(d) NoC size: 16×16



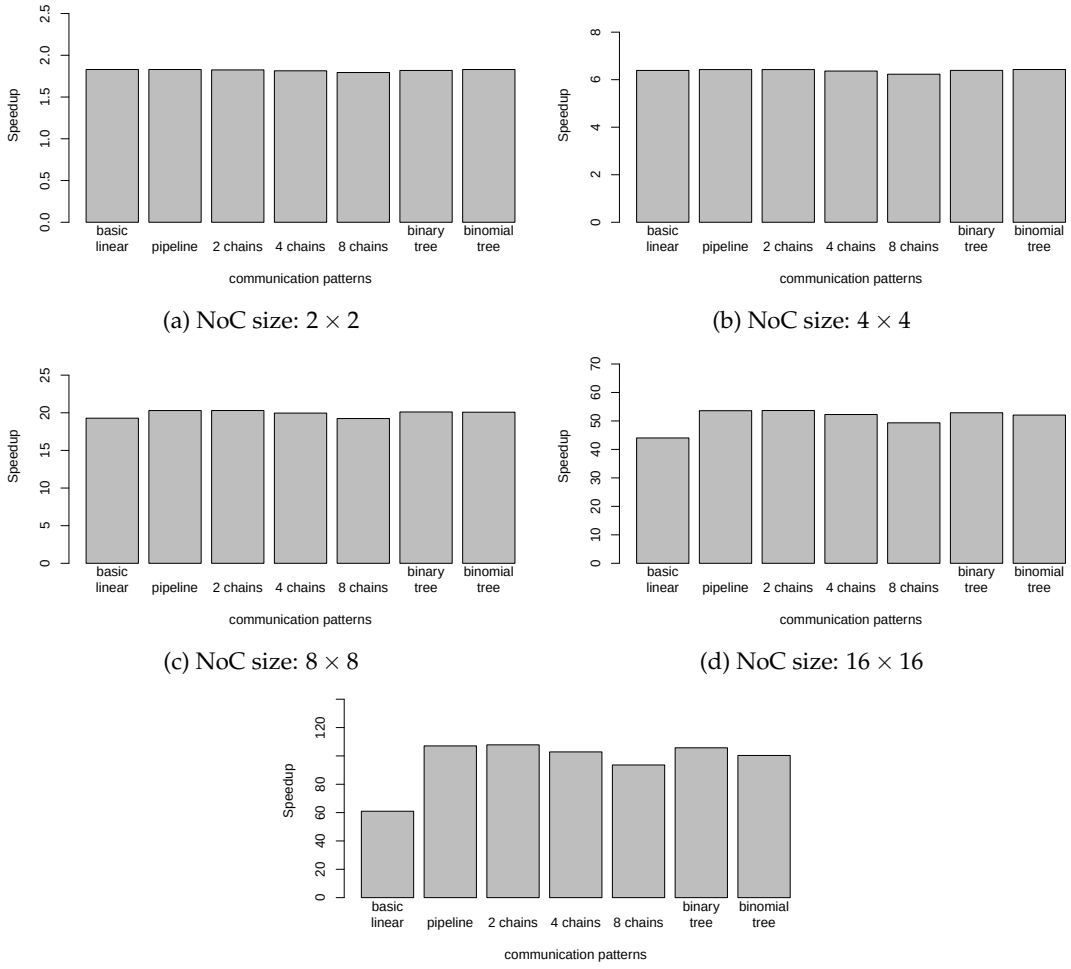
(e) NoC size: 32×32

Figure 6.11: The timing upper bounds for the execution of the communication phases of the LU benchmark for different implementations of the broadcast operation. The results are revealed for a input matrix with the size 2048×2048 elements and the application of 64×64 blocks on different NoC sizes. The horizontal axes show the different communication patterns that are applied for the barrier and the vertical axes display the WCET upper bound in terms of processor cycles. Please note the different scales of the vertical axes.

After examining the overall timing behaviour of the program we concentrate on the timing of the applied communication. These results are displayed in Figure 6.11 and show signifcant differences for the various communication patterns. Thereby, for small NoCs the pattern with 8 chains shows the largest timing bound of all patterns. This characteristic changes with an increasing number of nodes in the NoC. In large NoCs the execution with the basic linear pattern comprises the largest time for communication. On the other hand the binary pattern becomes more and more competitive to the other patterns when increasing the NoC size. For the largest platform it shows only marginal differences to the pattern 2 chains which is the pattern with the lowest timing bound. Again all these observations reflect the results revealed in Figures 5.21 and 5.22.

Figure 6.12 targets to investigate the influence of the communication patterns to the revealed speed-ups. As already mentioned in the examination of the different input sets the gained efficiency of the parallel executions is low. However, this is mainly caused by the structure of the algorithm. When comparing the speed-ups of the different communication patterns of a broadcast one can see that the revealed differences increase with the size of the observed NoC. For the smallest platform the speed-ups for the different communication patterns show only marginal disparities. In contrast, the largest NoC shows a high dependence on the applied communication pattern. It ranges from a speed-up of 60.97 to a speed-up of 107.81. Even if we exclude the basic linear pattern which exhibits a significant lower speed-up than all other patterns the variation still reaches from 93.61 for the pattern with 8 chains to 107.81 for the pattern with 2 chains. The pipeline and the binary tree pattern exhibit only marginal lower speed-ups to the pattern with 2 chains which is the one with the highest calculated value.

Please note that the investigations presented in this section are only a subset of the possible configurations that are applicable to examine with the analysis at hand. Due to the modular conception of the timing analysis it is possible to calculate the end-to-end timing upper bound of the benchmark as well as its building blocks (e.g. timing behaviour of communication parts) for a broad variation of different input sets, NoC sizes and other configurations with only small additional effort. Furthermore, due to the applied general-purpose schedules we do not need to regard to the concrete placement of processes on the NoC. This fact eases the timing analysis a lot as we can reuse the calculations of the communication operations for all investigated circumstances. We only need to analyse the building blocks of the program execution once and are then able to assemble them according to the preferred configuration. However, these advantages come to the price of increased overestimation.



(a) NoC size: 2×2

(b) NoC size: 4×4

(c) NoC size: 8×8

(d) NoC size: 16×16

(e) NoC size: 32×32

Figure 6.12: The speed-ups for the parallel execution of the LU benchmark for different broadcast implementations. The results are revealed for a input matrix with the size 2048×2048 elements and the application of 64×64 blocks on different NoC sizes. The horizontal axes show the different communication patterns that are applied for the barrier and the vertical axes display the according speed-ups. Please note the different scales of the vertical axes.

7 Conclusion and Future Work

This chapter concludes the thesis by summarizing the conducted research and the examined results. Additionally, we state interesting questions and issues that we see as subject to future work.

7.1 Conclusion

In this thesis we provided a method for estimating an end-to-end WCET upper bound for parallel MPI programs. Thereby, we mainly focused on the communication applied in such programs and assumed that local computations were timing analysable with state-of-the-art tools. In this thesis the timing predictability of the communication was based on various general-purpose TDM protocols. We investigated several sophisticated methods to distribute, collect or exchange data among multiple processes in terms of worst-case timing. Furthermore, we showed the composition of local computation WCET upper bounds and communication worst-case timing upper bounds to the end-to-end timing of a parallel MPI program. Finally, a case study was provided to demonstrate the application of the presented methods.

As basis for the research of this thesis we provided relevant background knowledge and developed a detailed system model that comprises several different topics. In the system model we defined suitable hardware architectures and described our view on routing within those architectures. Details about TDMA and its timing behaviour were defined afterwards as it was applied for enabling QoS. Thereby, we also distinguished between application specific and general-purpose TDM schedules. Since the MPI standard is an essential part of this thesis, the relevant aspects and our view on communication internals were presented. Finally, informations about timing analysis concluded the background knowledge and completed our system model.

Based on the developed system model we investigated different TDMA protocols in detail. Thereby, the main focus was on general-purpose schedules as they are able to provide timing guarantees for standard communications that are not optimized for a specific mapping of an application to the applied hardware. Therefore, the revealed timing behaviour was independent of the placement of the application's tasks on the hardware. At first, we contributed with a formal view on already existing TDM schedules and proved their freedom of conflicts. Afterwards, we proposed two new schedules that aim to overcome some shortcomings of existing schedules that occur in some situations. A procedure for the timing analysis of TDMA was presented and applied to all investigated schedules to reveal their characteristics in terms of bandwidth and latency. Furthermore, there was a detailed comparison of the schedules' timing behaviour for the case when one node sends data to multiple other nodes or vice versa. Finally, we chose representatives of application specific schedules to give an assessment about the performance of general-purpose schedules compared to the application specific schedules.

After developing and investigating the techniques that were applied for guaranteed service in this thesis, we took a closer look on the communication that is provided by MPI. At first, an overall analysis workflow was presented that bases on the well known technique of separating computation and communication parts for analysing them separately and afterwards combining the timings accordingly. Thereby, we stated the exact locations where to separate computation and communication, applied with hints about the analysis of the separated parts and finally contributed with a focus on the assembling of the revealed timing upper bounds. Thereby, our main focus was on the assembling phase. We developed an assembling scheme for simple point-to-point communication as well as for concurrent sendreceive communication. Based on the obtained formalism we further considered sophisticated communication among groups of multiple processes. Internally, there were different communication patterns depending on the applied communication. If it exhibited a central process that distributes or collects data, the communication was performed along a tree like pattern. On the other hand if no such process exists, patterns were applied that execute multiple steps of uniform data exchange with sendreceive communications.

We examined the timing upper bounds of these communication operations in detail and presented the composition of the overall WCET estimates of each communication pattern by extracting the building blocks of the timing. A comparison of the different communication patterns was provided for the overall timing upper bounds as well as for the pure WCTT upper bounds to put the performance of each pattern in relation to the other patterns. Furthermore, we compared the timing bounds of the different schedules for each presented communication pattern. Thereby, we only focused on the upper bounds for the traversal times because the dominant local computation hid the relevant traversal times. This dominance was probably mainly a subject of the low optimization level applied in the compilation process. Based on the revealed results we picked out the One-to-One and Triplet schedules as most promising schedules and conducted further investigations for them. We examined different configurations for each kind of group communication and revealed the communication patterns that exhibit the lowest timing upper bound for each configuration and TDM schedule. Finally, both schedules were compared by contrasting their best pattern for each configuration.

Based on the timing analysis of communication operations we presented our workflow for the analysis of a complete MPI program. Similar to our analysis of the communication operations we mainly focused on the assembling of revealed timing bounds. A method for calculating the end-to-end timing upper bound was provided and a MPI benchmark was analysed in a case study to show the applicability of our approach. For the case study we firstly investigated the algorithm structure of the program and examined important characteristics for the timing analysis.

During the investigations in this thesis it turned out that the application of a specific TDM schedule significantly influences the WCTT of flits as well as the bandwidth of the NoC. Thereby, important factors for the concrete timing are the size of the regarded NoC, the number of nodes participating in the communication and the message length. Depending on those parameters there are different communication situations and for each situation there are some general-purpose schedules that perform well in that situation. However, none of the known state-of-the-art schedules exhibits an acceptable performance in all situations. Thus, in this thesis new general-purpose schedules are presented to overcome that issue. The Triplet schedule never shows the best performance compared to all other schedules but it is competitive to the best schedule of each situation which is an unique selling point of this TDM schedule.

When regarding the communication of MPI operations it turned out that the application of a suitable communication pattern is highly decisive for a concrete timing upper bound. In most situations the communications that are based on a central process show the best results when the binary tree is applied. Furthermore, the communication based on uniform data exchange show the lowest WCET bounds with the bruck pattern. However, all investigated MPI communication operations reveal end-to-end timing upper bounds that are highly dominated by local computation. This characteristic is mainly caused by unoptimized code compilation and the absence of optimization for sending data (e.g. utilization of a DMA). When regarding to the pure traversal times the Triplet and the One-to-One TDM schedules show the lowest timing bounds for those communications with the application of sophisticated communication patterns. Thereby, mostly the One-to-One is slightly better than the Triplet schedule. However, if the application of sophisticated communication patterns is not possible the Triplet schedule shows advantages in most situations.

It was also shown that the provided analyses are applicable for the analysis of complete MPI programs. After examining the execution structure of the parallel MPI program the building blocks for computation and communication were analysed. Finally, the revealed timing upper bounds can be assembled according to the revealed execution structure and targeted configuration. Thereby, after performing the first steps of the analysis they can be reused in the assembling phase for each interesting configuration. Hence, it was possible to investigate the end-to-end WCET upper bounds for a wide range of execution configurations.

7.2 Future Work

The first part of this thesis focusses on the development and investigation of general-purpose TDM schedules. For this purpose we needed to consider concrete instances of these schedules that were built for a concrete NoC topology. In this thesis we assumed

a uni-directional torus topology. Since there are also NoCs available that apply other network topologies, the definition of general-purpose TDM schedules for those NoCs that are equivalent to the presented schedules should be investigated, too. Furthermore, a comparison of their timing behaviour to the timing of the schedules presented in this work would provide further insights which topology facilitates general-purpose TDM schedules most.

When focusing directly on the Triplet schedule an interesting option is to extend the usage of the schedule from pure data delivery to also apply synchronisation without additional network hardware. For that purpose the One-to-All part of the Triplet schedule could be utilized for the synchronisation of data transfers. With such a mechanism no additional synchronisation NoC is needed, which is the case for the current solution of Walter [Wal19]. However, this idea is only suitable for schedules that include a One-to-All part like the Triplet schedule. For the One-to-One schedule for example this technique would be not suitable. The investigation of such a mechanism is currently considered as work in progress by Unte [Unt20].

The investigation of the worst-case timing behaviour of message-passing based communication can be extended with several aspects. In addition to point-to-point communication and communication operation among a group of processes the MPI standard defines another type of communication. It is called neighbourhood collective communication and bases on logical process topologies. These topologies are virtually defined in the software application and are not related to the hardware topology of a NoC. In a neighbourhood collective operation each participating process performs a communication with all its neighbours defined in the applied logical process topology. A WCET analysis of this kind of communication would complete the analysis of the MPI standard's communication.

MPI communicators provide the context for each communication. Thereby, the standard defines two different types of theses objects which are called the intra- and the inter-communicators. The intra-communicator is responsible for communication among processes that are included in that communicator. In contrast, inter-communicators target the communication between the groups of processes each connected via a separate intra-communicator. This thesis focuses on communication that can be performed with intra-communicators and omit the communication with inter-communicators. However, since this kind of communication is part of the de-facto standard for message-passing its detailed examination seems to be a relevant objective that complements the investigations in the thesis.

In this thesis we presented procedures for analysing MPI operations as well as MPI programs. Though we developed scripts for the automatic assembling of single timing upper bounds to end-to-end upper bounds for an operation or program, there is still a lack in supporting the separation of local computation and parallel communication parts. Furthermore, it would be preferable to be able to perform an automated calculation of the WCET upper bounds of the separated parts. An automatically working tool that deals with these issues or at least parts of them would cause a great reduction of the analysis effort for the methods presented in this thesis. Therefore, we see the development of such a tool as a relevant task for future work.

Finally, the methods provided in this thesis very strongly focus on an efficient WCET analysis. However, this efficiency comes to the cost of a certain amount of overestimation,

mainly caused by the general-purpose TDM schedules and the modular usage of the timing upper bounds of the communication operations. Since the WCET upper bounds are desired to be close to the actual WCET it would be an interesting question for future research to reveal the actual distribution of the execution times of a MPI operation or MPI program depending on given input sets and processor pre-states. Even though it is hard to reveal the actual WCET of an operation or program, a distribution of measured execution times would reveal relevant insights to the timing behaviour and enables an estimation about the amount of introduced overestimation of the calculated WCET upper bound. In future research this knowledge may help to reduce the overestimation which is tolerated in this thesis.

Bibliography

[Ale+97] Albert Alexandrov et al. "LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation". In: *Journal of Parallel and Distributed Computing* 44.1 (1997), pp. 71–79. ISSN: 0743-7315. DOI: https://doi.org/10.1006/jpdc.1997.1346. URL: http://www.sciencedirect.com/science/article/pii/S0743731597913460.

[Alm+05] George Almási et al. "Optimization of MPI Collective Communication on BlueGene/L Systems". In: *Proceedings of the 19th Annual International Conference on Supercomputing*. ICS '05. Cambridge, Massachusetts: ACM, 2005, pp. 253–262. ISBN: 1-59593-167-8. DOI: 10.1145/1088149.1088183. URL: http://doi.acm.org/10.1145/1088149.1088183.

[Asa+06] Krste Asanovic et al. *The landscape of parallel computing research: A view from berkeley*. Tech. rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[Bai+91] D. H. Bailey et al. "The NAS parallel benchmarks summary and preliminary results". In: *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*. Nov. 1991, pp. 158–165. DOI: 10.1145/125826.125925.

[Bai+95] David Bailey et al. *The NAS parallel benchmarks 2.0*. Tech. rep. NAS-95-020, NASA Ames Research Center, 1995.

[Bak+09] M. Bakhouya et al. "Analytical Modeling and Evaluation of On-Chip Interconnects Using Network Calculus". In: *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip*. NOCS. Washington, DC, USA: IEEE Computer Society, 2009, pp. 74–79. ISBN: 978-1-4244-4142-6. DOI: 10.1109/NOCS.2009.5071447. URL: http://dx.doi.org/10.1109/NOCS.2009.5071447.

[Bal+11] Clément Ballabriga et al. "OTAWA: An Open Toolbox for Adaptive WCET Analysis". In: *Software Technologies for Embedded and Ubiquitous Systems*. Vol. 6399. LNCS. Springer Berlin Heidelberg, 2011, pp. 35–46.

[BD01] L. Benini and G. De Micheli. "Powering networks on chips". In: *International Symposium on System Synthesis*. Sept. 2001, pp. 33–38. DOI: 10.1145/500001.500009.

[BDM02] George Brunemann, Thomas A Dollmeyer, and Joseph C Mathew. *System and method for transmission of application software to an embedded vehicle computer*. US Patent 6,487,717. 2002.

[Ben+03] Gregory D. Benson et al. "A comparison of MPICH allgather algorithms on switched networks". In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 10th European PVM/MPI User's Group Meeting, Venice, Italy*. Springer. 2003, pp. 335–343. DOI: 10.1007/978-3-540-39924-7_47. URL: https://doi.org/10.1007/978-3-540-39924-7_47.

[Bie+12] Pierre Bieber et al. "New Challenges for Future Avionic Architectures." In: *AerospaceLab* 4 (May 2012), p. 1–10. URL: https://hal.archives-ouvertes.fr/hal-01184101.

[Bin+03] Niraj Bindal et al. "Scalable sub-10ps skew global clock distribution for a 90nm multi-GHz microprocessor". In: *IEEE ISSCC*. 2003, pp. 346–498.

[BKK08] C. Bienia, S. Kumar, and Kai Li. "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on Chip-Multiprocessors". In: *IEEE International Symposium on Workload Characterization*. Sept. 2008, pp. 47–56. DOI: 10.1109/IISWC.2008.4636090.

[Bol+07] E. Bolotin et al. "Routing Table Minimization for Irregular Mesh NoCs". In: *Design, Automation and Test in Europe Conference and Exhibition*. Apr. 2007, pp. 1–6. DOI: 10.1109/DATE.2007.364414.

[Bol02] Béla Bollobás. *Modern graph theory*. Vol. 184. Springer Science & Business Media, 2002. ISBN: 0-387-98488-7.

[Boy+13] Marc Boyer et al. "Combining static priority and weighted round-robin like packet scheduling in AFDX for incremental certification and mixed-criticality support". In: *Proceedings of the 5th European Conference for Aeronautics and Space Sciences (EUCASS)*. Munich, Germany, 2013.

[BPG91] Michael Barnett, David G. Payne, and Robert A. van de Geijn. *Optimal Broadcasting in Mesh-Connected Architectures*. Tech. rep. Austin, TX, USA, 1991. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.6999.

[BR05] C. Burguiere and C. Rochange. "A contribution to branch prediction modeling in WCET analysis". In: *Design, Automation and Test in Europe*. Mar. 2005, 612–617 Vol. 1. DOI: 10.1109/DATE.2005.7.

[Bru+97] Jehoshua Bruck et al. "Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems". In: *IEEE Transactions on Parallel and Distributed Systems* 8.11 (Nov. 1997), pp. 1143–1156. ISSN: 1045-9219. DOI: 10.1109/71.642949. URL: https://doi.org/10.1109/71.642949.

[BS12] Florian Brandner and Martin Schoeberl. "Static Routing in Symmetric Real-time Network-on-chips". In: *Proceedings of the 20th International Conference on Real-Time and Network Systems*. RTNS '12. Pont á Mousson, France: ACM, 2012, pp. 61–70. ISBN: 978-1-4503-1409-1. DOI: 10.1145/2392987.2392995. URL: http://doi.acm.org/10.1145/2392987.2392995.

[Byu+98] Byungjae Kim et al. "A real-time communication method for wormhole switching networks". In: *Proceedings of the International Conference on Parallel Processing*. Aug. 1998, pp. 527–534. DOI: 10.1109/ICPP.1998.708526.

[Cha+04] E. W. Chan et al. "On optimizing collective communication". In: *IEEE International Conference on Cluster Computing*. Sept. 2004, pp. 145–155. DOI: 10.1109/CLUSTR.2004.1392612.

[Cil+15] B. Cilku et al. "A TDMA-Based arbitration scheme for mixed-criticality multicore platforms". In: *International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. June 2015, pp. 1–6. DOI: 10.1109/EBCCSP.2015.7300671.

[Cor18] Microsoft Corporation. *Microsoft MPI*. https://docs.microsoft.com/en-us/message-passing-interface/microsoft-mpi. Accessed: 2019-07-18. 2018.

[CP00] Antoine Colin and Isabelle Puaut. "Worst Case Execution Time Analysis for a Processor with Branch Prediction". In: *Real-Time Systems* 18.2 (May 2000), pp. 249–274. ISSN: 1573-1383. DOI: 10.1023/A:1008149332687. URL: https://doi.org/10.1023/A:1008149332687.

[CP01] A. Colin and I. Puaut. "A modular and retargetable framework for tree-based WCET analysis". In: *Proceedings 13th Euromicro Conference on Real-Time Systems*. June 2001, pp. 37–44. DOI: 10.1109/EMRTS.2001.933995.

[Cru91a] R. L. Cruz. "A calculus for network delay. I. Network elements in isolation". In: *IEEE Transactions on Information Theory* 37.1 (Jan. 1991), pp. 114–131. DOI: 10.1109/18.61109.

[Cru91b] R. L. Cruz. "A calculus for network delay. II. Network analysis". In: *IEEE Transactions on Information Theory* 37.1 (Jan. 1991), pp. 132–141. DOI: 10.1109/18.61110.

[Cul+93] David Culler et al. "LogP: Towards a realistic model of parallel computation". In: *ACM Sigplan Notices*. Vol. 28. 7. ACM. 1993, pp. 1–12.

[DC98] Elizias De Korte and David Cayer. *Multiplexed random access memory with time division multiplexing through a single read/write port*. US Patent 5,822,776. 1998.

[de +14] B. D. de Dinechin et al. "Time-critical computing on a single-chip massively parallel processor". In: *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Mar. 2014, pp. 1–6. DOI: 10.7873/DATE.2014.110.

[Die+11] J. Diemer et al. "Real-time communication analysis for networks with two-stage arbitration". In: *Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*. Oct. 2011, pp. 243–252. DOI: 10.1145/2038642.2038680.

[Din+14] Benoît Dupont de Dinechin et al. "Guaranteed Services of the NoC of a Manycore Processor". In: *Proceedings of the 2014 International Workshop on Network on Chip Architectures*. NoCArc '14. Cambridge, United Kingdom: ACM, 2014, pp. 11–16. ISBN: 978-1-4503-3064-0. DOI: 10.1145/2685342.2685344. URL: http://doi.acm.org/10.1145/2685342.2685344.

[DT04] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Elsevier, 2004. ISBN: 0122007514.

[EIS75] S. Even, A. Itai, and A. Shamir. "On the complexity of time table and multi-commodity flow problems". In: *16th Annual Symposium on Foundations of Computer Science (SFCS 1975)*. Oct. 1975, pp. 184–193. DOI: 10.1109/SFCS.1975.21.

[Eng+03] Jakob Engblom et al. "Worst-case execution-time analysis for embedded real-time systems". In: *International Journal on Software Tools for Technology Transfer* 4.4 (Aug. 2003), pp. 437–455. ISSN: 1433-2787. DOI: 10.1007/s100090100054. URL: https://doi.org/10.1007/s100090100054.

[EV10] Sinem Coleri Ergen and Pravin Varaiya. "TDMA Scheduling Algorithms for Wireless Sensor Networks". In: *Wireless Networks* 16.4 (May 2010), pp. 985–997. ISSN: 1022-0038. DOI: 10.1007/s11276-009-0183-0. URL: http://dx.doi.org/10.1007/s11276-009-0183-0.

[Fei91] Dror G. Feitelson. *Communicators: Object-based multiparty interactions for parallel programming*. Hebrew University of Jerusalem. Leibniz Center for Research in Computer Science. Department of Computer Science, 1991.

[Fel+07] Meik Felser et al. "Dynamic software update of resource-constrained distributed embedded systems". In: *Embedded System Design: Topics, Techniques and Trends*. Springer, 2007, pp. 387–400.

[Fri+16a] Martin Frieb et al. "A Parallelization Approach for Hard Real-Time Systems and Its Application on Two Industrial Programs". In: *International Journal of Parallel Programming* 44.6 (2016), pp. 1296–1336. ISSN: 1573-7640. DOI: 10.1007/s10766-016-0432-7.

[Fri+16b] Martin Frieb et al. "Employing MPI Collectives for Timing Analysis on Embedded Multi-Cores". In: *16th International Workshop on Worst-Case Execution Time Analysis (WCET)*. 2016. DOI: 10.4230/OASIcs.WCET.2016.10.

[Fri+18] Martin Frieb et al. "Lightweight Hardware Synchronization for Avoiding Buffer Overflows in Network-on-Chips". In: *Architecture of Computing Systems (ARCS)*. Ed. by Mladen Berekovic et al. Cham: Springer International Publishing, 2018, pp. 112–126. ISBN: 978-3-319-77610-1.

[Fri19] Martin Frieb. "Hardware Extensions for a Timing-Predictable Many-Core Processor". PhD thesis. University of Augsburg, 2019.

[FW99] Christian Ferdinand and Reinhard Wilhelm. "Efficient and Precise Cache Behavior Prediction for Real-Time Systems". In: *Real-Time Systems* 17.2 (Nov. 1999), pp. 131–181. ISSN: 1573-1383. DOI: 10.1023/A:1008186323068. URL: https://doi.org/10.1023/A:1008186323068.

[Gab+04] Edgar Gabriel et al. "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation". In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting Budapest, Hungary*. Ed. by Dieter Kranzlmüller, Péter Kacsuk, and Jack Dongarra. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 97–104. ISBN: 978-3-540-30218-6. DOI: 10.1007/978-3-540-30218-6_19.

[GAC+13] Kees Goossens, Arnaldo Azevedo, Karthik Chandrasekar, et al. "Virtual execution platforms for mixed-time-criticality systems: The CompSOC architecture and design flow". In: *ACM SIGBED Review* 10.3 (2013), pp. 23–34.

[Gan08] Om Prakash Gangwal. *Network-On-Chip Environment and Method For Reduction of Latency*. US Patent App. 11/910,750. Aug. 2008.

[GCL06] S. Gopalakrishnan, M. Caccamo, and Lui Sha. "Switch Scheduling and Network Design for Real-Time Systems". In: *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*. Apr. 2006, pp. 289–300. DOI: 10.1109/RTAS.2006.42.

[GDR05] K. Goossens, J. Dielissen, and A. Radulescu. "AEthereal network on chip: concepts, architectures, and implementations". In: *IEEE Design Test of Computers* 22.5 (Sept. 2005), pp. 414–421. DOI: 10.1109/MDT.2005.99.

[GH10] Kees Goossens and Andreas Hansson. "The Aethereal Network on Chip After Ten Years: Goals, Evolution, Lessons, and Future". In: *47th Design Automation Conference*. Anaheim, California, 2010. ISBN: 978-1-4503-0002-5. DOI: 10.1145/1837274.1837353.

[GMC09] S. Gobriel, D. Mosse, and R. Cleric. "TDMA-ASAP: Sensor Network TDMA Scheduling with Adaptive Slot-Stealing and Parallelism". In: *29th IEEE International Conference on Distributed Computing Systems*. June 2009, pp. 458–465. DOI: 10.1109/ICDCS.2009.80.

[GO00] R. A. Guerin and A. Orda. "Networks with advance reservations: the routing perspective". In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. 19th Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 1. Mar. 2000, 118–127 vol.1. DOI: 10.1109/INFCOM.2000.832180.

[Goo+17] Kees Goossens et al. "NoC-based multiprocessor architecture for mixed-time-criticality applications". In: *Handbook of hardware/software codesign* (2017), pp. 1–40.

[Gro+11] Boris Grot et al. "Kilo-NOC: A Heterogeneous Network-on-chip Architecture for Scalability and Service Guarantees". In: *Proceedings of the 38th Annual International Symposium on Computer Architecture*. ISCA '11. San Jose, California, USA: ACM, 2011, pp. 401–412. ISBN: 978-1-4503-0472-6. DOI: 10.1145/2000064.2000112. URL: http://doi.acm.org/10.1145/2000064.2000112.

[Haf+11] Mehnaz Hafeez et al. "Survey of MPI Implementations". In: *Digital Information and Communication Technology and Its Applications*. Ed. by Hocine Cherifi, Jasni Mohamad Zain, and Eyas El-Qawasmeh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 206–220. ISBN: 978-3-642-22027-2.

[Ham+18] Arne Hamann et al. "Response Time Analysis for Fixed Priority Servers". In: *Proceedings of the 26th International Conference on Real-Time Networks and Systems*. RTNS '18. Chasseneuil-du-Poitou, France: ACM, 2018, pp. 254–264. ISBN: 978-1-4503-6463-8. DOI: 10.1145/3273905.3273927. URL: http://doi.acm.org/10.1145/3273905.3273927.

[Har+18] Tim Harde et al. “Configurations and Optimizations of TDMA Schedules for Periodic Packet Communication on Networks on Chip”. In: *Proceedings of the 26th International Conference on Real-Time Networks and Systems*. ACM. 2018, pp. 202–212.

[HCG07] Andreas Hansson, Martijn Coenen, and Kees Goossens. “Channel Trees: Reducing Latency by Sharing Time Slots in Time-multiplexed Networks on Chip”. In: *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS ’07. Salzburg, Austria: ACM, 2007, pp. 149–154. ISBN: 978-1-59593-824-4. DOI: 10.1145/1289816.1289855. URL: http://doi.acm.org/10.1145/1289816.1289855.

[HE05] A. Hamann and R. Ernst. “TDMA time slot and turn optimization with evolutionary search techniques”. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Mar. 2005, 312–317 Vol. 1. DOI: 10.1109/DATE.2005.299.

[HKR14] S. Han, K. Kang, and J. Ryu. “Determination of Delay Bound over Multihop Real-Time Switches with Virtual Output Queuing”. In: *IEEE 28th International Conference on Advanced Information Networking and Applications*. May 2014, pp. 892–898. DOI: 10.1109/AINA.2014.144.

[HLL08] Torsten Hoefler, Florian Lorenzen, and Andrew Lumsdaine. “Sparse Nonblocking Collectives in Quantum Mechanical Calculations”. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Ed. by Alexey Lastovetsky, Tahar Kechadi, and Jack Dongarra. Springer Berlin Heidelberg, 2008, pp. 55–63. ISBN: 978-3-540-87475-1.

[HLR07] T. Hoefler, A. Lumsdaine, and W. Rehm. “Implementation and performance analysis of non-blocking collective operations for MPI”. In: *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC ’07*. Nov. 2007, pp. 1–10. DOI: 10.1145/1362622.1362692.

[Hoc94] Roger W. Hockney. “The communication challenge for MPP: Intel Paragon and Meiko CS-2”. In: *Parallel Computing* 20.3 (1994), pp. 389–398. ISSN: 0167-8191. DOI: https://doi.org/10.1016/S0167-8191(06)80021-9. URL: http://www.sciencedirect.com/science/article/pii/S0167819106800219.

[Hoe+04] Torsten Hoefler et al. “A Survey of Barrier Algorithms for Coarse Grained Supercomputers”. In: *Chemnitzer Informatik Berichte* 04.03 (Dec. 2004). ISSN: 0947-5152.

[Hoe+05] T. Hoefler et al. “A practical approach to the rating of barrier algorithms using the LogP model and Open MPI”. In: *International Conference on Parallel Processing Workshops (ICPPW’05)*. June 2005, pp. 562–569. DOI: 10.1109/ICPPW.2005.14.

[Hoe+11] Torsten Hoefler et al. “The scalable process topology interface of MPI 2.2”. In: *Concurrency and Computation: Practice and Experience* 23.4 (2011), pp. 293–310. DOI: 10.1002/cpe.1643. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1643.

[HP11] Damien Hardy and Isabelle Puaut. "WCET analysis of instruction cache hierarchies". In: *Journal of Systems Architecture* 57.7 (2011). Special Issue on Worst-Case Execution-Time Analysis, pp. 677–694. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2010.08.007. URL: http://www.sciencedirect.com/science/article/pii/S1383762110001074.

[HP19] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Sixth Edition. Elsevier, 2019. ISBN: 978-0-12-811905-1.

[HS12] T. Hoefler and T. Schneider. "Optimization principles for collective neighborhood communications". In: *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. Nov. 2012, pp. 1–10. DOI: 10.1109/SC.2012.86.

[HSG09] Andreas Hansson, Mahesh Subburaman, and Kees Goossens. "Aelite: A Flit-synchronous Network on Chip with Composable and Predictable Services". In: *Design, Automation and Test in Europe*. DATE '09. Nice, France: European Design and Automation Association, 2009, pp. 250–255. ISBN: 978-3-9810801-5-5. URL: http://dl.acm.org/citation.cfm?id=1874620.1874679.

[HT09] T. Hoefler and J. L. Traff. "Sparse collective operations for MPI". In: *IEEE International Symposium on Parallel Distributed Processing*. May 2009, pp. 1–8. DOI: 10.1109/IPDPS.2009.5160935.

[Hui95] Hui Zhang. "Service disciplines for guaranteed performance service in packet-switching networks". In: *Proceedings of the IEEE* 83.10 (Oct. 1995), pp. 1374–1396. DOI: 10.1109/5.469298.

[Ind14] Leandro Soares Indrusiak. "End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration". In: *Journal of Systems Architecture* 60.7 (2014), pp. 553–561. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2014.05.002. URL: http://www.sciencedirect.com/science/article/pii/S1383762114000800.

[Int10] Intel Labs. *SCC external architecture specification (EAS)*. Tech. rep. Intel Corparation, 2010.

[Jal+08] Antoine Jalabert et al. "xpipesCompiler: A Tool for Instantiating Application-Specific Networks on Chip". In: *Design, Automation, and Test in Europe: The Most Influential Papers of 10 Years Date*. Ed. by Rudy Lauwereins and Jan Madsen. Dordrecht: Springer Netherlands, 2008, pp. 157–171. ISBN: 978-1-4020-6488-3. DOI: 10.1007/978-1-4020-6488-3_12. URL: https://doi.org/10.1007/978-1-4020-6488-3_12.

[JC97] Jenq-Shyan Yang and Chung-Ta King. "Efficient tree-based multicast in wormhole-routed 2D meshes". In: *Proceedings of the 1997 International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN'97)*. Dec. 1997, pp. 494–500. DOI: 10.1109/ISPAN.1997.645142.

[JC98] Jenq-Shyan Yang and Chung-Ta King. "Designing tree-based barrier synchronization on 2D mesh networks". In: *IEEE Transactions on Parallel and Distributed Systems* 9.6 (June 1998), pp. 526–534. DOI: 10.1109/71.689440.

[JII94] Jin-Young Choi, Insup Lee, and Inhye Kang. "Timing analysis of superscalar processor programs using ACSR". In: *Proceedings of 11th IEEE Workshop on Real-Time Operating Systems and Software*. May 1994, pp. 63–67. DOI: 10.1109/RTOSS.1994.292559.

[Jin+05] Jing Chen et al. "Performance evaluation of Allgather algorithms on terascale Linux cluster with fast Ethernet". In: *Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)*. Nov. 2005, pp. 437–442. DOI: 10.1109/HPCASIA.2005.75.

[JOD06] Jiri Jaros, Milos Ohlidal, and Vaclav Dvorak. "Complexity of collective communications on NoCs". In: *International Symposium on Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006.* IEEE. 2006, pp. 127–133.

[JP86] M. Joseph and P. Pandya. "Finding Response Times in a Real-Time System". In: *The Computer Journal* 29.5 (Jan. 1986), pp. 390–395. ISSN: 0010-4620. DOI: 10.1093/comjnl/29.5.390. URL: https://doi.org/10.1093/comjnl/29.5.390.

[Kas+16] E. Kasapaki et al. "Argo: A Real-Time Network-on-Chip Architecture With an Efficient GALS Implementation". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.2 (Feb. 2016), pp. 479–492. DOI: 10.1109/TVLSI.2015.2405614.

[Kas15] Evangelia Kasapaki. "An Asynchronous Time-Division-Multiplexed Network-on-Chip for Real-Time Systems". PhD thesis. 2015.

[KB03] H. Kopetz and G. Bauer. "The time-triggered architecture". In: *Proceedings of the IEEE* 91.1 (Jan. 2003), pp. 112–126. DOI: 10.1109/JPROC.2002.805821.

[KBV00] Thilo Kielmann, Henri E. Bal, and Kees Verstoep. "Fast Measurement of LogP Parameters for Message Passing Platforms". In: *Parallel and Distributed Processing*. Ed. by José Rolim. Springer Berlin Heidelberg, 2000, pp. 1176–1183. ISBN: 978-3-540-45591-2.

[KG94] H. Kopetz and G. Grunsteidl. "TTP/spl minus/a protocol for fault-tolerant real-time systems". In: *Computer* 27.1 (Jan. 1994), pp. 14–23. DOI: 10.1109/2.248873.

[KGP15] H. Kashif, S. Gholamian, and H. Patel. "SLA: A Stage-Level Latency Analysisfor Real-Time Communicationin a Pipelined Resource Model". In: *IEEE Transactions on Computers* 64.4 (Apr. 2015), pp. 1177–1190. DOI: 10.1109/TC.2014.2315617.

[Kop11] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011. ISBN: 978-1-4419-8236-0.

[KP16] H. Kashif and H. Patel. "Buffer Space Allocation for Real-Time Priority-Aware Networks". In: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Apr. 2016, pp. 1–12. DOI: 10.1109/RTAS.2016.7461324.

[KSR98] Arkady Kanevsky, Anthony Skjellum, and Anna Rounbehler. "MPI/RT - an emerging standard for high-performance real-time systems". In: *31th Hawaii International Conference on System Sciences*. IEEE. 1998, pp. 157–166.

[Kum+02] S. Kumar et al. "A network on chip architecture and design methodology". In: *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*. Apr. 2002, pp. 117–124. DOI: 10.1109/ISVLSI.2002.1016885.

[Kuw+09] Y. Kuwata et al. "Real-Time Motion Planning With Applications to Autonomous Urban Driving". In: *IEEE Transactions on Control Systems Technology* 17.5 (Sept. 2009), pp. 1105–1118. ISSN: 2374-0159. DOI: 10.1109/TCST.2008.2012116.

[Le 96] Jean-Yves Le Boudec. *Network calculus made easy*. Tech. rep. EPFL-DI 96/218, 1996.

[Le 98] J. - Y. Le Boudec. "Application of network calculus to guaranteed service networks". In: *IEEE Transactions on Information Theory* 44.3 (May 1998), pp. 1087–1096. DOI: 10.1109/18.669170.

[Lee03] Sunggu Lee. "Real-time wormhole channels". In: *Journal of Parallel and Distributed Computing* 63.3 (2003), pp. 299–311. ISSN: 0743-7315. DOI: https://doi.org/10.1016/S0743-7315(02)00055-2. URL: http://www.sciencedirect.com/science/article/pii/S0743731502000552.

[Lei85] C. E. Leiserson. "Fat-trees: Universal networks for hardware-efficient supercomputing". In: *IEEE Transactions on Computers* C-34.10 (Oct. 1985), pp. 892–901. ISSN: 0018-9340. DOI: 10.1109/TC.1985.6312192.

[Lis12] Björn Lisper. "Towards Parallel Programming Models for Predictability". In: *12th International Workshop on Worst-Case Execution Time Analysis*. Ed. by Tullio Vardanega. Vol. 23. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 48–58. ISBN: 978-3-939897-41-5. DOI: 10.4230/OASIcs.WCET.2012.48. URL: http://drops.dagstuhl.de/opus/volltexte/2012/3556.

[Liu+03] Jiuxing Liu et al. "Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics". In: *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. SC '03. Phoenix, AZ, USA: ACM, 2003. ISBN: 1-58113-695-1. DOI: 10.1145/1048935.1050208.

[LJS05] Zhonghai Lu, Axel Jantsch, and Ingo Sander. "Feasibility Analysis of Messages for On-chip Networks Using Wormhole Routing". In: *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*. ASP-DAC '05. Shanghai, China: ACM, 2005, pp. 960–964. ISBN: 0-7803-8737-6. DOI: 10.1145/1120725.1120767. URL: http://doi.acm.org/10.1145/1120725.1120767.

[LM95] Yau-Tsun Steven Li and Sharad Malik. "Performance Analysis of Embedded Software Using Implicit Path Enumeration". In: *Proceedings of the ACM SIGPLAN 1995 Workshop on Languages, Compilers, and Tools for Real-time Systems*. LCTES '95. La Jolla, California, USA: ACM, 1995, pp. 88–98. DOI: 10.1145/216636.216666. URL: http://doi.acm.org/10.1145/216636.216666.

[LMN94] Xiaola Lin, P. K. McKinley, and L. M. Ni. "Deadlock-free multicast wormhole routing in 2-D mesh multicomputers". In: *IEEE Transactions on Parallel and Distributed Systems* 5.8 (Aug. 1994), pp. 793–804. ISSN: 1045-9219. DOI: 10.1109/71.298203.

[LRM06] Xianfeng Li, Abhik Roychoudhury, and Tulika Mitra. "Modeling out-of-order processors for WCET analysis". In: *Real-Time Systems* 34.3 (Nov. 2006), pp. 195–227. ISSN: 1573-1383. DOI: 10.1007/s11241-006-9205-5. URL: https://doi.org/10.1007/s11241-006-9205-5.

[LRS14] P. A. Lasota, G. F. Rossano, and J. A. Shah. "Toward safe close-proximity human-robot interaction with standard industrial robots". In: *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. Aug. 2014, pp. 339–344. DOI: 10.1109/CoASE.2014.6899348.

[LT01] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Berlin, Heidelberg: Springer-Verlag, 2001. ISBN: 3-540-42184-X.

[Lud+11] Daniele Ludovici et al. "Mesochronous NoC Technology for Power-efficient GALS MPSoCs". In: *Proceedings of the Fifth International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*. INA-OCMC '11. Heraklion, Greece: ACM, 2011, pp. 27–30. ISBN: 978-1-4503-0272-2. DOI: 10.1145/1930037.1930045. URL: http://doi.acm.org/10.1145/1930037.1930045.

[Mar+09] R. Marculescu et al. "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.1 (Jan. 2009), pp. 3–21. ISSN: 0278-0070. DOI: 10.1109/TCAD.2008.2010691.

[McK+94] P. K. McKinley et al. "Unicast-based multicast communication in wormhole-routed networks". In: *IEEE Transactions on Parallel and Distributed Systems* 5.12 (Dec. 1994), pp. 1252–1265. DOI: 10.1109/71.334899.

[McK99] N. McKeown. "The iSLIP scheduling algorithm for input-queued switches". In: *IEEE/ACM Transactions on Networking* 7.2 (Apr. 1999), pp. 188–201. DOI: 10.1109/90.769767.

[Met13] Stefan Metzlaff. "Analysable Instruction Memories for Hard Real-Time Systems". PhD thesis. University of Augsburg, 2013.

[Mil+04] M. Millberg et al. "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip". In: *Design, Automation and Test in Europe Conference and Exhibition*. Vol. 2. Feb. 2004, 890–895 Vol.2. DOI: 10.1109/DATE.2004.1269001.

[Mis+17] Jörg Mische et al. "Reduced Complexity Many-Core: Timing Predictability Due to Message-Passing". In: *Architecture of Computing Systems - ARCS 2017: 30th International Conference, Vienna, Austria, April 3–6, 2017, Proceedings*. Cham: Springer International Publishing, 2017, pp. 139–151. ISBN: 978-3-319-54999-6. DOI: 10.1007/978-3-319-54999-6_11.

[MPI15] MPI-forum. *MPI: A Message-Passing Interface Standard Version 3.1*. available at `http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`. 2015.

[MSA08] Ben D. MacArthur, Rubén J. Sánchez-García, and James W. Anderson. "Symmetry in complex networks". In: *Discrete Applied Mathematics* 156.18 (2008), pp. 3525–3531.

[MTR94] P. K. McKinley, Y.-J. Tsai, and D. Robinson. *A survey of collective communication in wormhole-routed massively parallel computers*. Tech. rep. MSU-CPS-94-35, Dept. of Computer Science, Michigan State University, East Lansing, Michigan, 1994.

[MTR95] P. K. McKinley, Yih-jia Tsai, and D. F. Robinson. "Collective communication in wormhole-routed massively parallel computers". In: *Computer* 28.12 (Dec. 1995), pp. 39–50. ISSN: 0018-9162. DOI: `10.1109/2.476198`.

[MU14] Jörg Mische and Theo Ungerer. "Guaranteed Service Independent of the Task Placement in NoCs with Torus Topology". In: *22nd International Conference on Real-Time Networks and Systems*. RTNS '14. Versaille, France: ACM, 2014, pp. 151–160. ISBN: 978-1-4503-2727-5. DOI: `10.1145/2659787.2659804`.

[OG04] Sarp Oral and Alan D. George. "Multicast performance modeling and evaluation for high-speed unidirectional torus networks". In: *Microprocessors and Microsystems* 28.9 (2004), pp. 477–489.

[Pan95] Dhabaleswar K. Panda. "Fast barrier synchronization in wormhole k-ary n-cube networks with multidestination worms". In: *Future Generation Computer Systems* 11.6 (1995). High-Performance Computer Architecture, pp. 585–602. ISSN: 0167-739X. DOI: `https://doi.org/10.1016/0167-739X(95)00026-O`. URL: `http://www.sciencedirect.com/science/article/pii/0167739X95000260`.

[PC01] Guangyu Pei and Charles Chien. "Low power TDMA in large wireless sensor networks". In: *2001 MILCOM Proceedings Communications for Network-Centric Operations: Creating the Information Force*. Vol. 1. Oct. 2001, 347–351 vol.1. DOI: `10.1109/MILCOM.2001.985817`.

[PGS06] I. M. Panades, A. Greiner, and A. Sheibanyrad. "A Low Cost Network-on-Chip with Guaranteed Service Well Suited to the GALS Approach". In: *2006 1st International Conference on Nano-Networks and Workshops*. Sept. 2006, pp. 1–5. DOI: `10.1109/NANONET.2006.346219`.

[Pje+07] Jelena Pješivac-Grbović et al. "Performance analysis of MPI collective operations". In: *Cluster Computing* 10.2 (2007), pp. 127–143.

[PNP13] Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti. "Mapping a multi-rate synchronous language to a many-core processor". In: *19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2013, pp. 293–302.

[Pop+04] P. Pop et al. "Design optimization of multi-cluster embedded systems for real-time applications". In: *Design, Automation and Test in Europe Conference and Exhibition*. Vol. 2. Feb. 2004, 1028–1033 Vol.2. DOI: `10.1109/DATE.2004.1269028`.

[Psa+15] A. Psarras et al. "PhaseNoC: TDM scheduling at the virtual-channel level for efficient network traffic isolation". In: *2015 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Mar. 2015, pp. 1090–1095. DOI: 10.7873/DATE.2015.0418.

[PSS15] Wolfgang Puffitsch, Rasmus Bo Sørensen, and Martin Schoeberl. "Time-division Multiplexing vs Network Calculus: A Comparison". In: *Proceedings of the 23rd International Conference on Real-Time and Networks Systems*. RTNS '15. Lille, France: ACM, 2015, pp. 289–296. ISBN: 978-1-4503-3591-1. DOI: 10.1145/2834848.2834868. URL: http://doi.acm.org/10.1145/2834848.2834868.

[QLD09] Y. Qian, Z. Lu, and W. Dou. "Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip". In: *2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*. May 2009, pp. 44–53. DOI: 10.1109/NOCS.2009.5071444.

[QLD10] Y. Qian, Z. Lu, and W. Dou. "Analysis of Worst-Case Delay Bounds for On-Chip Packet-Switching Networks". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.5 (May 2010), pp. 802–815. DOI: 10.1109/TCAD.2010.2043572.

[Rap08] Rapita Systems Ltd. *RapiTime White Paper*. Whitepaper. 2008. URL: www.edn.com/Pdf/ViewPdf?contentItemId=4137753.

[RE15] Eberle A. Rambo and Rolf Ernst. "Worst-case Communication Time Analysis of Networks-on-chip with Shared Virtual Channels". In: *Proceedings of the 2015 Design, Automation and Test in Europe Conference and Exhibition*. DATE '15. Grenoble, France: EDA Consortium, 2015, pp. 537–542. ISBN: 978-3-9815370-4-8. URL: http://dl.acm.org/citation.cfm?id=2755753.2755874.

[Ren04] Yuqing Ren. *Embedded software update system*. US Patent 6,760,908. 2004.

[RMC95] David F Robinson, Philip K McKinley, and Betty HC Cheng. "Optimal multicast communication in wormhole-routed torus networks". In: *IEEE Transactions on Parallel and Distributed Systems* 6.10 (1995), pp. 1029–1042.

[RMC97] David F Robinson, Philip K McKinley, and Betty HC Cheng. "Path-based multicast communication in wormhole-routed unidirectional torus networks". In: *Journal of Parallel and Distributed Computing* 45.2 (1997), pp. 104–121.

[Roc+10] Christine Rochange et al. "WCET Analysis of a Parallel 3D Multigrid Solver Executed on the MERASA Multi-Core". In: *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*. Ed. by Björn Lisper. Vol. 15. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010, pp. 90–100. ISBN: 978-3-939897-21-7. DOI: 10.4230/OASIcs.WCET.2010.90. URL: http://drops.dagstuhl.de/opus/volltexte/2010/2829.

[ROG06] Venkatesh Rajendran, Katia Obraczka, and J. J. Garcia-Luna-Aceves. "Energy-efficient, Collision-free Medium Access Control for Wireless Sensor Networks". In: *Wireless Networks* 12.1 (Feb. 2006), pp. 63–78. ISSN: 1022-0038. DOI: 10.1007/s11276-006-6151-z. URL: http://dx.doi.org/10.1007/s11276-006-6151-z.

[Ros+07] J. Rosen et al. "Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip". In: *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. Dec. 2007, pp. 49–60. DOI: 10.1109/RTSS.2007.24.

[RS09] Christine Rochange and Pascal Sainrat. "A Context-Parameterized Model for Static Analysis of Execution Times". In: *Transactions on High-Performance Embedded Architectures and Compilers II*. Ed. by Per Stenström. Springer Berlin Heidelberg, 2009, pp. 222–241. ISBN: 978-3-642-00904-4. DOI: 10.1007/978-3-642-00904-4_12. URL: https://doi.org/10.1007/978-3-642-00904-4_12.

[San+01] Sangman Moh et al. "Four-ary tree-based barrier synchronization for 2D meshes without nonmember involvement". In: *IEEE Transactions on Computers* 50.8 (Aug. 2001), pp. 811–823. DOI: 10.1109/12.947001.

[SB08] Z. Shi and A. Burns. "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching". In: *Second ACM/IEEE International Symposium on Networks-on-Chip (NOCS 2008)*. Apr. 2008, pp. 161–170. DOI: 10.1109/NOCS.2008.4492735.

[SB09] Zheng Shi and Alan Burns. "Improvement of Schedulability Analysis with a Priority Share Policy in On-Chip Networks". In: *17th International Conference on Real-Time and Network Systems*. Ed. by Laurent George and Maryline Chetto andMikael Sjodin. Paris, France, Oct. 2009, pp. 75–84. URL: https://hal.inria.fr/inria-00441970.

[SB10] Zheng Shi and Alan Burns. "Schedulability analysis and task mapping for real-time on-chip communication". In: *Real-Time Systems* 46.3 (Dec. 2010), pp. 360–385. ISSN: 1573-1383. DOI: 10.1007/s11241-010-9108-3. URL: https://doi.org/10.1007/s11241-010-9108-3.

[Sch+12] Martin Schoeberl et al. "A Statically Scheduled Time-Division-Multiplexed Network-on-Chip for Real-Time Systems". In: *Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*. NOCS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 152–160. ISBN: 978-0-7695-4677-3. DOI: 10.1109/NOCS.2012.25.

[Sch07] M. Schoeberl. "A Time-Triggered Network-on-Chip". In: *2007 International Conference on Field Programmable Logic and Applications*. Aug. 2007, pp. 377–382. DOI: 10.1109/FPL.2007.4380675.

[Sch12] Johannes Scheller. "Real-time operating systems for many-core platforms". In: *Mém. de mast. Toulouse, France: ISAE/ONERA* (2012).

[SCT10] A. Schranzhofer, J. Chen, and L. Thiele. "Timing Analysis for TDMA Arbitration in Resource Sharing Systems". In: *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. Apr. 2010, pp. 215–224. DOI: 10.1109/RTAS.2010.24.

[SEE01] Friedhelm Stappert, Andreas Ermedahl, and Jakob Engblom. "Efficient Longest Executable Path Search for Programs with Complex Flows and Pipeline Effects". In: *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. CASES '01. Atlanta, Georgia, USA: ACM, 2001, pp. 132–140. ISBN: 1-58113-399-5. DOI: 10.1145/502217.502240. URL: http://doi.acm.org/10.1145/502217.502240.

[SG12] Paul Sack and William Gropp. "Faster Topology-aware Collective Algorithms Through Non-minimal Communication". In: *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '12. New Orleans, Louisiana, USA: ACM, 2012, pp. 45–54. ISBN: 978-1-4503-1160-1. DOI: 10.1145/2145816.2145823. URL: http://doi.acm.org/10.1145/2145816.2145823.

[Skj+04] Anthony Skjellum et al. "The Real-Time Message Passing Interface Standard (MPI/RT-1.1)". In: *Concurrency and Computation: Practice and Experience* 16.S1 (2004).

[Skj+94] Anthony Skjellum et al. "The design and evolution of Zipcode". In: *Parallel Computing* 20.4 (1994). Message Passing Interfaces, pp. 565–596. ISSN: 0167-8191. DOI: https://doi.org/10.1016/0167-8191(94)90029-9. URL: http://www.sciencedirect.com/science/article/pii/0167819194900299.

[SKS13] Jens Sparsø, Evangelia Kasapaki, and Martin Schoeberl. "An Area-efficient Network Interface for a TDM-based Network-on-chip". In: *Design, Automation and Test in Europe*. DATE '13. Grenoble, France: EDA Consortium, 2013, pp. 1044–1047. ISBN: 978-1-4503-2153-2. URL: http://dl.acm.org/citation.cfm?id=2485288.2485538.

[SL90] Anthony Skjellum and Alvin P. Leung. "A portable multicomputer communication library atop the reactive kernel". In: *Proceedings of the Fifth Distributed Memory Computing Conference*. IEEE, 1990.

[SMG14] R. A. Stefan, A. Molnos, and K. Goossens. "dAElite: A TDM NoC Supporting QoS, Multicast, and Fast Connection Set-Up". In: *IEEE Transactions on Computers* 63.3 (Mar. 2014), pp. 583–594. ISSN: 0018-9340. DOI: 10.1109/TC.2012.117.

[SMT94] D. Saha, S. Mukherjee, and S. K. Tripathi. "Multi-rate traffic shaping and end-to-end performance guarantees in ATM networks". In: *Proceedings of ICNP - 1994 International Conference on Network Protocols*. Oct. 1994, pp. 188–195. DOI: 10.1109/ICNP.1994.344361.

[Sør+14] R. B. Sørensen et al. "A Metaheuristic Scheduler for Time Division Multiplexed Networks-on-Chip". In: *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. June 2014, pp. 309–316. DOI: 10.1109/ISORC.2014.43.

[Sør+15] Rasmus Bo Sørensen et al. "Message Passing on a Time-predictable Multicore Processor". In: *18th International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE. 2015, pp. 51–59.

[SP68] T. Sekimoto and J. Puente. "A Satellite Time-Division Multiple-Access Experiment". In: *IEEE Transactions on Communication Technology* 16.4 (Aug. 1968), pp. 581–588. ISSN: 0018-9332. DOI: 10.1109/TCOM.1968.1089895.

[SPT09] Nikolay Stoimenov, Simon Perathoner, and Lothar Thiele. "Reliable Mode Changes in Real-time Systems with Fixed Priority or EDF Scheduling". In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE '09. Nice, France: European Design and Automation Association, 2009, pp. 99–104. ISBN: 978-3-9810801-5-5. URL: http://dl.acm.org/citation.cfm?id=1874620.1874645.

[SR90] John A. Stankovic and Krithi Ramamritham. "What is predictability for real-time systems?" In: *Real-Time Systems* 2.4 (Nov. 1990), pp. 247–254. ISSN: 1573-1383. DOI: 10.1007/BF01995673. URL: https://doi.org/10.1007/BF01995673.

[Ste+16] Alexander Stegmeier et al. "WCTT Bounds for MPI Primitives in the PaterNoster NoC". In: *14th International Workshop on RealTime Networks (RTN)*. Vol. 13. 4. New York, NY, USA: ACM, Nov. 2016, pp. 25–30. DOI: 10.1145/3015037.3015041.

[Ste+18] Alexander Stegmeier et al. "Analysing Real-Time Behaviour of Collective Communication Patterns in MPI". In: *Proceedings of the 26th International Conference on Real-Time Networks and Systems*. RTNS '18. Chasseneuil-du-Poitou, France: ACM, 2018, pp. 137–147. ISBN: 978-1-4503-6463-8. DOI: 10.1145/3273905.3273906. URL: http://doi.acm.org/10.1145/3273905.3273906.

[Stu+06] S. Stuijk et al. "Resource-Efficient Routing and Scheduling of Time-Constrained Network-on-Chip Communication". In: *9th EUROMICRO Conference on Digital System Design (DSD'06)*. Aug. 2006, pp. 45–52. DOI: 10.1109/DSD.2006.81.

[Sub+10] S. Suboh et al. "Analytical modeling and evaluation of network-on-chip architectures". In: *2010 International Conference on High Performance Computing Simulation*. June 2010, pp. 615–622. DOI: 10.1109/HPCS.2010.5547064.

[Sun+95] Sung-Soo Lim et al. "An accurate worst case timing analysis for RISC processors". In: *IEEE Transactions on Software Engineering* 21.7 (July 1995), pp. 593–604. DOI: 10.1109/32.392980.

[Sun+98] Sung-Soo Lim et al. "A worst case timing analysis technique for multiple-issue machines". In: *Proceedings 19th IEEE Real-Time Systems Symposium*. Dec. 1998, pp. 334–345. DOI: 10.1109/REAL.1998.739765.

[TC94] R. Thakur and A. Choudhary. "All-to-all communication on meshes with wormhole routing". In: *Proceedings of 8th International Parallel Processing Symposium*. Apr. 1994, pp. 561–565. DOI: 10.1109/IPPS.1994.288248.

[TE17a] S. Tobuschat and R. Ernst. "Efficient Latency Guarantees for Mixed-Criticality Networks-on-Chip". In: *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Apr. 2017, pp. 113–122. DOI: 10.1109/RTAS.2017.31.

[TE17b] S. Tobuschat and R. Ernst. "Real-time communication analysis for Networks-on-Chip with backpressure". In: *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2017*. Mar. 2017, pp. 590–595. DOI: 10.23919/DATE.2017.7927055.

[Tea16] Open MPI Development Team. *Open MPI main development repository*. https://github.com/open-mpi/ompi. Accessed: 2016-05-19. 2016.

[Tea19] MPICH Development Team. *MPICH: High-Performance Portable MPI*. https://www.mpich.org/. Accessed: 2019-07-18. 2019.

[TG03] Rajeev Thakur and William D. Gropp. "Improving the performance of collective operations in MPICH". In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2003, pp. 257–267.

[The02] Henrik Theiling. *Control flow graphs for real-time systems analysis: reconstruction from binary executables and usage in ILP-based path analysis*. 2002. DOI: http://dx.doi.org/10.22028/D291-25778.

[TPL96] Yu-Chee Tseng, D. K. Panda, and Ten-Hwang Lai. "A trip-based multicasting model in wormhole-routed networks with virtual channels". In: *IEEE Transactions on Parallel and Distributed Systems* 7.2 (Feb. 1996), pp. 138–150. ISSN: 1045-9219. DOI: 10.1109/71.485503.

[TRG05] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. "Optimization of Collective Communication Operations in MPICH". In: *The International Journal of High Performance Computing Applications* 19.1 (2005), pp. 49–66. DOI: 10.1177/1094342005051521. URL: https://doi.org/10.1177/1094342005051521.

[Tu+12] Bibo Tu et al. "Performance analysis and optimization of MPI collective operations on multi-core clusters". In: *The Journal of Supercomputing* 60.1 (Apr. 2012), pp. 141–162. ISSN: 1573-0484. DOI: 10.1007/s11227-009-0296-3.

[Ung+16] Theo Ungerer et al. "Parallelizing Industrial Hard Real-Time Applications for the parMERASA Multicore". In: *ACM Transactions on Embedded Computer Systems* 15.3 (May 2016), 53:1–53:27. ISSN: 1539-9087. DOI: 10.1145/2910589. URL: http://doi.acm.org/10.1145/2910589.

[Unt20] Tilmann Unte. "Echtzeitfähiger DMA Controller für ein TDM-basiertes Network-on-Chip". work in progress. MA thesis. Universität Augsburg, 2020.

[Wal19] Dominik Walter. "Implementierung und Evaluierung des One-to-One Schedules im echtzeitfähigen RC/MC-Prozessor". MA thesis. Universität Augsburg, Feb. 2019.

[Wil+08] Reinhard Wilhelm et al. "The Worst-case Execution-time Problem – Overview of Methods and Survey of Tools". In: *ACM Transactions on Embedded Computer Systems* 7.3 (May 2008), 36:1–36:53. ISSN: 1539-9087. DOI: 10.1145/1347375.1347389. URL: http://doi.acm.org/10.1145/1347375.1347389.

[Woo+95] Steven Cameron Woo et al. "The SPLASH-2 Programs: Characterization and Methodological Considerations". In: *Proceedings of the 22nd Annual International Symposium on Computer Architecture*. ISCA '95. S. Margherita Ligure, Italy: ACM, 1995, pp. 24–36. ISBN: 0-89791-698-0. DOI: 10.1145/223982.223990. URL: http://doi.acm.org/10.1145/223982.223990.

[WT06] Ernesto Wandeler and Lothar Thiele. "Optimal TDMA Time Slot and Cycle Length Allocation for Hard Real-time Systems". In: *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*. ASP-DAC '06. Yokohama, Japan: IEEE Press, 2006, pp. 479–484. ISBN: 0-7803-9451-8. DOI: 10.1145/1118299.1118417. URL: https://doi.org/10.1145/1118299.1118417.

[Wu+17] Bichen Wu et al. "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. July 2017.

[YPM16] Karthik Yagna, Onkar Patil, and Frank Mueller. "Efficient and Predictable Group Communication for Manycore NoCs". In: *High Performance Computing*. Ed. by Julian M. Kunkel, Pavan Balaji, and Jack Dongarra. Cham: Springer International Publishing, 2016, pp. 383–403. ISBN: 978-3-319-41321-1.

[ZBN93] N. Zhang, A. Burns, and M. Nicholson. "Pipelined processors and worst case execution times". In: *Real-Time Systems* 5.4 (Oct. 1993), pp. 319–343. ISSN: 1573-1383. DOI: 10.1007/BF01088834. URL: https://doi.org/10.1007/BF01088834.

List of Figures

1.1 Schematic illustration of a parallel example program 4
1.2 Workflow for the two higher abstraction levels of the timing analysis of parallel MPI programs 4

2.1 Several widely used topologies for NoCs 9
2.2 A folded torus of a uni-directional torus with 4×4 nodes 10
2.3 Conflicting communication 11
2.4 Usage of registers in a physical link 12
2.5 Illustration of functions Pos and PE 15
2.6 Concept of TDMA 19
2.7 Global view of an example for a TDM schedule 22
2.8 The nodes' local views of the examples from Figure 2.7 23
2.9 Example communication for general-purpose schedules 29
2.10 Communication patterns $cp \in CP^{P2P}$ that deliver data according to point-to-point communications 36
2.11 The functionality of collective communications that are based on a central process 40
2.12 The functionality of collective communications that are based on uniform data exchange 41
2.13 Communication patterns which deliver data in a tree like way 42
2.14 Communication patterns for performing uniform data exchange 44
2.15 Different aspects of the execution times of a task 45

4.1 Logical network structure of the assumed topology 57
4.2 Overlapped execution of a schedule's rounds 60
4.3 Procedures for delivering flits along one direction in a horizontal or vertical ring of a NoC 61
4.4 Global view on the One-to-All schedule 65
4.5 Global view on the All-to-One schedule 69
4.6 Global view on the One-to-One schedule 73
4.7 The execution of one TDM cycle for one node in the All-to-All schedule . . . 80
4.8 Global view on an example of the All-to-All schedule 81
4.9 Overview of overlapped execution of the Alternate schedule 91
4.10 Overview of overlapped execution of the Triplet schedule 94
4.11 The local view of a node n_i participating in a TDM schedule 100
4.12 WCTT of 1:N communication 107

4.13 Comparison of application specific and general-purpose schedules in terms of WCTT . . . 111

5.1 Workflow for timing analysis of parallel MPI operations . . . 114
5.2 Connection of computation and communication instances at basic communication operations . . . 116
5.3 Boundaries between WCET and WCTT . . . 119
5.4 Possible communication patterns for performing multiple send-receive communications at the same time . . . 121
5.5 The timing characteristics of a send-receive operation . . . 122
5.6 Delivery of four flits along a row of three processes . . . 127
5.7 The building blocks of the overall WCET of point-to-point communication . . . 138
5.8 Comparison of the different point-to-point communications . . . 139
5.9 WCETs of the building blocks that form the patterns' end-to-end WCET of MPI_Broadcast . . . 140
5.10 Overall WCETs of the communication patterns for MPI_Broadcast . . . 141
5.11 WCTTs of the communication patterns for MPI_Broadcast . . . 141
5.12 WCETs of the building blocks that form the patterns' end-to-end WCET of MPI_Allgather . . . 143
5.13 Overall WCETs of the communication patterns for MPI_Allgather . . . 144
5.14 WCTTs of the communication patterns for MPI_Allgather . . . 144
5.15 WCTTs as a function of the group size of the communication patterns for MPI_Broadcast . . . 146
5.16 WCTTs as a function of the number of flits of the communication patterns for MPI_Broadcast . . . 147
5.17 WCTTs as a function of the group size of the communication patterns for MPI_Allgather . . . 148
5.18 WCTTs as a function of the number of flits of the communication patterns for MPI_Allgather . . . 149
5.19 The minimal WCTTs exhibited for any communication pattern of the broadcast communication when applying the One-to-One schedule . . . 151
5.20 The minimal WCTTs exhibited for any communication pattern of the broadcast communication when applying the Triplet schedule . . . 153
5.21 The minimal WCTTs exhibited for any communication pattern of the allgather communication when applying the One-to-One schedule . . . 156
5.22 The minimal WCTTs exhibited for any communication pattern of the allgather communication when applying the Triplet schedule . . . 157
5.23 The difference in terms of WCTTs between the Triplet and the One-to-One TDM schedule for the broadcast communication . . . 158
5.24 The difference in terms of WCTTs between the Triplet and the One-to-One TDM schedule for the allgather communication . . . 159

6.1 Workflow for timing analysis of parallel MPI programs . . . 162
6.2 The block shape of the LU calculation's input matrix . . . 169
6.3 The overall workflow of the execution of the LU calculation . . . 170

6.4 The mapping of blocks of the LU calculation's input matrix to processes . . . 172
6.5 The execution workflow of the LU benchmark when it is executed with 4 processes . . . 173
6.6 The end-to-end WCET upper bound of the LU benchmark for different configurations of the input set and NoCs . . . 177
6.7 The timing upper bounds for the execution of the communication phases of the LU benchmark . . . 179
6.8 The speed-up results for the parallel executions of the LU benchmark . . . 181
6.9 The end-to-end WCET upper bounds of the LU benchmark for different implementations of the barrier operation . . . 182
6.10 The end-to-end WCET upper bounds of the LU benchmark for different implementations of the broadcast operation . . . 184
6.11 The timing upper bounds of the communication phases of the LU benchmark for different implementations of the broadcast operation . . . 185
6.12 The speed-ups for the parallel execution of the LU benchmark for different broadcast implementations . . . 187