Mapping hard real-time Tasks on Network-on-Chip manycore architectures

Chawki Benchehida

Université de Lille

Université Oran 1

École doctorale des Sciences Pour l'Ingénieur SPI-MADIS

Placement des tâches temps-réel dur sur des multicoeurs en réseau sur puce

Thèse de doctorat

Informatique

Chawki Benchehida

09 Novembre 2021

Audrey QUEUDET	Maître de conférences (HDR), Université de Nantes, France	Rapporteur
Mohamed BENMOHAMMED	Professeur, Université de Constantine 2, Algérie	Rapporteur
Sidi Mohammed BENSLIMANE	Professeur, ESI-SBA, Algérie	Examinateur
Samia BOUCHERKHA	Professeur, Université de Constantine 2, Algérie	Examinateur
Houssam Eddine ZAHAF	Maître de conférences, Université de Nantes, France	Examinateur
Giuseppe LIPARI	Professeur, Université de Lille, France	Directeur de thèse
Mohammed Kamel BENHAOUA	Professeur, Université de Mascara, Algérie	Directeur de thèse
Abdoulaye GAMATIE	Directeur de recherche, LIRMM/Université de Montpellier, France	Président

University of Lille
University of Oran1

École doctorale des Sciences Pour l'Ingénieur SPI-MADIS

Mapping Hard Real-Time Tasks on Network-on-Chip Manycore Architectures

---

Doctoral Thesis

Computer science

Chawki Benchehida

November 09, 2021

Audrey QUEUDET	Associate Professor (HDR), University of Nantes, France	Referee
Mohamed BENMOHAMMED	Full Professor, University of Constantine 2, Algeria	Referee
Sidi Mohammed BENSLIMANE	Full Professor, ESI-SBA, Algeria	Examiner
Samia BOUCHERKHA	Full Professor, University of Constantine 2, Algeria	Examiner
Houssam Eddine ZAHAF	Associate Professor, University of Nantes, France	Examiner
Giuseppe LIPARI	Full Professor, University of Lille, France	Director
Mohammed Kamel BENHAOUA	Full Professor, University of Mascara, Algeria	Director
Abdoulaye GAMATIE	CNRS Research Director, LIRMM/University of Montpellier, France	President

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

Date

Signature

Abstract

In this dissertation, we tackle the problem of execution complex multi-thread real-time applications on modern Network-on-Chip architectures.

Network-on-Chip (NoC) is a promising technology that fits the increasing performance demands of Cyber-Physical Systems (CPS). The introduction of NoCs is justified by the fact that classical multi-core single-bus architectures fail to address the performance requirements and the predictability needs of modern CPS applications, especially as the number of cores increases. Even if the use of cache memories mitigates the bottleneck effect of single bus architectures, caches introduce unpredictable delays in accessing data, which in turn make it difficult to estimate the execution time of tasks.

Most CPS applications are time-sensitive: tasks are assigned deadlines that must never exceed, otherwise a critical failure may occur. Such systems are denoted by *hard real-time*. Consequently, the communications that occur in the network, denoted by *on-chip communications*, must be predictable and as fast as possible to prevent deadline-missing. Since the task position on the NoC determines its communication cost, the allocation of the application tasks on the chip cores is a crucial problem.

In this thesis, we address specifically the problem of allocating a set of real-time applications, each composed of several parallel tasks, whose structure is described by a Directed Acyclic Graph (DAG), onto a Network-on-Chip processor.

First, we study the problem of bounding the communication cost depending on the different message scheduling policies at the router level. Then we address the problem of task scheduling and of verifying the schedulability of a certain allocation.

Then, we propose an approach to reduce the complexity of the task allocation problem and its analysis cost. Moreover, we propose a task mapping strategy through a meta-heuristic which performs an effective design-space exploration for DAG (Directed Acyclic Graph) tasks. Lastly, in addition to on-chip communications, we studied the mapping problem when off-chip communications are integrated into the model.

Acknowledgments

I truly believe that the individual success of a person always depends on those behind you. I take this opportunity to thank all those who supported me during my years of thesis. Without them, I wouldn't be who I am.

Being part of two teams was an unforgettable experience. First, I would like to thank my first PhD supervisor Prof. Kamel Benhaoua for his guidance, advice, support, trust, and understanding. I also would like to thank my second supervisor Prof. Giuseppe Lipari for his trust, support, advice, and for his great sympathy. Sincerely, I could not have asked for better supervisors than both of you. Furthermore, I want to thank Prof. Mohamed Benmohammed and Dr. Audrey Queudet for their acceptance of being my referees. I would like to thank also my examiners Prof. Sidi Mohammed Benslimane, Prof. Samia Boucherkha, and Prof. Abdoulaye Gamatie for devoting their precious time to review my work. Lastly, I would like to thank Dr. Houssam Eddine Zahaf for being a friend and an exemplary mentor; our scientific discussions were very valuable.

As a PhD student, the teams in which you pertain become the second family. A special thanks goes to all my colleagues in LAPECI Laboratory for the time spent together and their friendship. Also, I'm blessed to be a member of CRIStAL Laboratory. I would like to express my gratitude to Emeraude Team for their kindness and their warm welcome. I won't forget our shared meals and Julien's jokes.

A special thanks to all my friends, so many. Citing you one by one will take the complete acknowledgment section. Thank you to exist.

A special thought to my family, my parents, my sisters Cherifa and Wahiba, all my cousins, my aunts and uncles who have supported me unconditionally.

Contents

List of Figures

List of Tables

List of Symbols

τ	Task
Γ	Task set
T_i	Task period
C_i	Task worst-case execution time (WCET)
D_i	Task deadline
O_i	Task offset
J_i	Jitter release
$\mathcal{J}_i$	Job's task
U_i	Utilization rate
a_i	Task activation time
P_i	Task priority
R_i	Task response time
$H(\Gamma)$	Task set hyperperiod
M	Hardware resource
PE	Processing engine
$\mathcal{A}$	Tile
$\mathbb{M}$	Set of tiles
$\mathbb{I}$	Set of communications
λ	Unidirectional link
$\mathcal{R}$	Router
$\mathcal{M}$	Message
P	Packet
F	FLit
$TF(\mathcal{M}_i)$	Total number of FLit in a message
$\mathcal{V}$	Set of vertices
$\mathcal{E}$	Set of edges
VC	Virtual Channel
Δ	Total number of slots by a TDMA cycle
π	Path between source and destination sub-tasks
I_i	Task interference
v_i	Sub-task
$\mathcal{C}$	Communication between two tasks
lat	Communication latency
$e(v_i, v_j)$	Edge between two sub-tasks
w	Sub-task busy period
$\mathsf{pred}(v_i)$	Set of sub-task predecessors
$\mathsf{succ}(v_i)$	Set of sub-task successors
S	Task mapping schema

Ω	Front pareto set
B_i	DRAM Bank

Introduction

Objectives & Motivation

The increasing complexity of modern Cyber-Physical systems (CPS) requires the usage of powerful embedded computing systems to satisfy their timing constraints. Typically, the system monitors a physical environment using sensors, then process and react to the environment's state. This sense-compute-react pattern must be completed within a predefined time window imposed by the speed of the environment's evolution. Such constraints are known as real-time constraints. The correctness of the system design relies on its ability to provide guarantees on timing constraints. A violation of a timing constraint might be a serious source of damages.

Classical multi-core platforms design have limited settings in terms of the number of computing resources, which make them inadequate for current and near-future CPS applications. One serious problem of all interconnection architectures based on a single shared bus is that access to memory and devices is exclusive, therefore, the bus represents a bottleneck. Even if the use of local cache memories mitigates the performance problems by keeping data in the local cache, it also introduces unpredictable timing to access shared data. This makes it difficult to efficiently bound the execution time of real-time tasks.

Networks-on-Chip (also called on-Chip Networks) architectures have been proposed to solve the bus bottleneck problem by providing a more scalable architecture. NoCs can host hundreds of cores on a single chip, connected through a network. Data is moved from one core to another, or to the main memory, by means of network interfaces represented by simple routers. However, these enhanced features increase the complexity, since we have to deal with routing, switching protocols, congestion handling, and classical network problems even when accessing data in the main memory.

Supporting real-time constraints on NoC-based architectures requires particular attention. The system must be predictable; i.e. we must be able to estimate tight and safe bounds for inter-task communications as well as to compute task response time.

The goal of this thesis is to provide support for hard real-time applications on Networks-on-chip. We consider real-time constraints and define NoC parameters and configurations. We propose task and communication mapping such that all deadlines are respected.

In this thesis, we deal with hard-real systems: deadline misses are not tolerable, and the results produced after the deadline are no longer useful. We propose novel techniques and schedulability analyses for a set of real-time tasks modeled by **DAG**s (*Directed Acyclic Graphs*) on NoC resources. Further, we tackle memory-to-chip transfers by extending DAG model to the **AER** (*Acquisition, Execute, Write-back*) task model.

Contribution of this dissertation

Simulation and analysis of real-time communication The performance and the real-time behavior of a Network-on-Chip platform are extremely dependent on the micro-architecture of the routers. In fact, the latter are in charge of regulating the network traffic. Additionally, they also must be designed to deal with real-time constraints: in particular, when multiple incoming flows are directed toward the same output port, the router has to choose which one is sent first by using an arbitration mechanism.

We propose a comparative empirical study on the two main arbitration schemes used in NoC routers, the *priority-preemptive* vs. *round-robin*, onto a mesh-2D network structure. We implemented a simulator that models a wormhole communication mechanism and supports the two policies. Furthermore, we compared analytically the two policies in the worst-case scenario (highest contention rate) by using state-of-the-art formulas.

The goal of this study is to prove that, under *certain assumptions*, the RR protocol can be used as a Guarantee Service (GS) protocol. We compare RR with a priority-preemptive arbitration scheme, and we show that RR can better handle communications in the presence of high workloads and at the same time it is possible to compute an upper bound to the communication delay. We compare the two approaches in order to highlight the gap that separates the two arbitration approaches in the average case. We used both analysis methods and simulations to calculate the task response time in the worst-case and the average case respectively. We conclude that RR can be used in a hard real-time context where deadline-missing is forbidden.

Task allocation with earliest deadline scheduling on Network-on-chip Scheduling real-time tasks on single processor is a research problem that has been widely treated in the literature. However, scheduling real-time tasks on manycore platforms with migration is still an open research problem, and the existing solutions are difficult to deploy on a NoC.

A practical strategy is to transform the manycore scheduling problem into a much simpler and well-known problem, the single core scheduling problem. Therefore, in this study, we propose a technique to transform multicores scheduling

on several single core scheduling problems by means of partitioned scheduling. Contrary to global scheduling, partitioned scheduling doesn't allow task migration.

We used Earliest deadline first (EDF) scheduling policy, which is considered as an optimal scheduler on a single core. Therefore, it allows a greater schedulability rate compared to other schedulers. In this study, applications are modeled by a DAG (Directed-Acyclic-Graph) which is an expressive model used to highlight the communications between tasks. The nodes represent the sub-tasks while the edges express the communication between them. We use bin-packing heuristics to prove the efficiency of our contribution.

Memory-aware real-time tasks mapping on Network-on-Chip The problem of mapping a set of independent tasks on a multicore is well-known to be a NP-Hard problem. The complexity increases when considering tasks that communicate and the allocation of tasks' communication. In fact, we distinguish two types of communications: (i) task-to-task communications and (ii) and task-to-main-memory communications (also denoted by off-chip communications). The former regards the communications between sub-tasks whereas the latter concerns the communications between the task and the main-memory (typically, an external DRAM). Thus, finding the optimal system mapping requires sophisticated optimization techniques. In this work, we propose a task allocation algorithm onto NoC platform based on Simulated Annealing (SA) method.

We choose to use meta-heuristics for their trade-off in providing a reasonable coverage of the design space solution in a reasonable amount of time. To this end, we use an expressive DAG task model that includes the off-chip communications, called **AER** (*Acquisition-Execution-Restitution*) tasks.

Organization

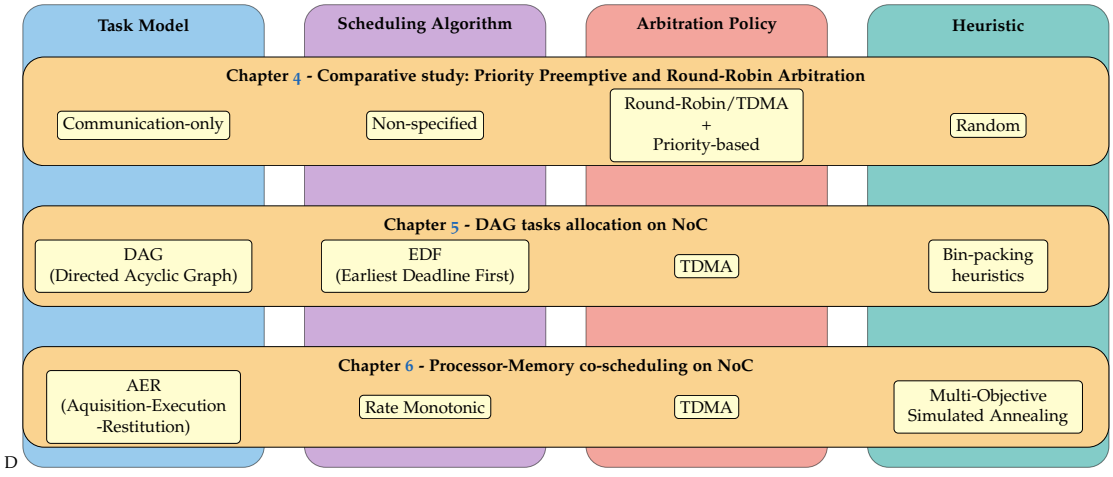


Figure 0.1.: Contributions of this dissertation by elements

In this thesis, we focus on the task mapping problem of hard real-time tasks onto a Network-on-Chip architecture. The problem has been studied in the literature from different points of view, always aiming at providing the best task mapping schema that fits the requirements. Thus, the main contributions of this thesis are:

- Determine the best arbitration mode for Network-on-Chip routers,
- Allocation of DAG tasks onto NoC with earliest deadline scheduling,
- Reducing the worst-case analysis complexity of real-time systems,
- Proposing a predictable Network-on-Chip architecture dealing with off-chip communications,
- Efficient memory-aware task allocation.

Figure 0.1 presents a summary of the thesis contributions by distinguishing the different parts of a study.

This thesis is structured as follows. The first chapter introduces real-time systems and scheduling theory. The chapter also introduces the schedulability analysis of hard real-time systems in multicores platforms. In chapter two, we give an overview of the Network-on-Chip architecture and its elements. We also present the router's architecture and its features. In chapter three, we introduce the task mapping problem and we present prior work available in the literature. Chapter four addresses an empirical comparative study of the routers' architectures of the Network-on-Chip. In chapter five, we trait about DAG task mapping technique with partitioned scheduling. We propose a series of transformations to reduce the schedulability analysis complexity. Finally, chapter six addresses the task mapping problem through meta-heuristics onto an execution platform with off-chip memory.

Part I.

Background, Context and Related work

1. Introduction to Real-Time Systems

Contents

1.1. Introduction

We define a Cyber-physical System (CPS) as a system that interacts with the external environment by means of physical reactions, but not only, as a reaction could be a spoken-order such as a warning or an advice, or simply a meaningful illustration. For instance, the physical reactions are applied by *actuators* and produced after a series of calculations based on a source of data coming from an external acquisition by means of intermediate sensors, cameras, etc. All of flight control systems, automotive applications, and telecommunication systems could be considered in part, or even entirely as CPS. We illustrate in Figure 1.1 a simple CPS schema, where the environment interacts with the system. However, these reactions are sensitive regarding the speed of their executions. In fact, some reactions must be executed before a given time called *deadline*, when exceeding the latter, the result has less importance or any at all. Such systems are denoted as *real-time systems*.

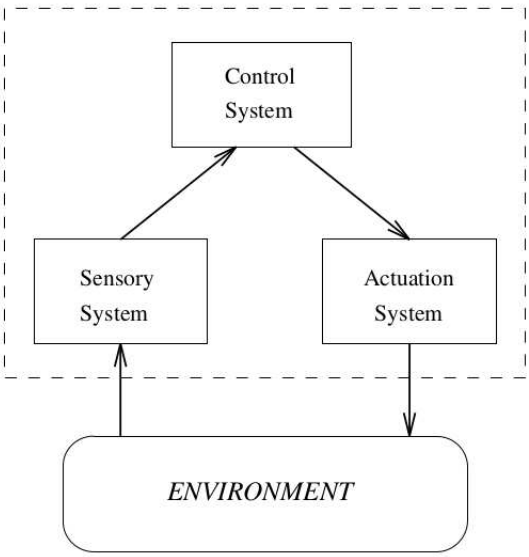


Figure 1.1.: A Cyber-physical System schema

In this section, we introduce real-time systems, a type of time-related systems that are subject to timing constraints. These systems may or may not tolerate deadline-misses depending on their class. We classify real-time systems as follows:

Hard real-time : A system is classified as *hard*, if producing the results after the deadline might involve catastrophic scenarios on the system itself.

Firm real-time : A system is classified as *firm*, if producing the results after the deadline is useless and doesn't deteriorate the system.

Soft real-time : A system is classified as *soft*, if producing the results after the deadline doesn't create consequences, except a performance degradation.

Definition 1.1.1. A real-time system is a system which its execution doesn't depends only in the results correctness but also on the time in which the results are produced Stankovic, 1988.

1.2. Task Model

Most real-time systems are modeled as sets of concurrent applications (or *tasks*). In this section we describe the a real-time application as a finite set Γ of *independent* real-time tasks τ: $\Gamma = \{\tau_0, \cdots, \tau_n\}$.

Typically, a independent task can be represented by either the *periodic* task model or the *sporadic* task model. Both models represent an infinite sequence of task instances (refereed by *jobs*). In the periodic task model, jobs are released strictly periodically, separated by a fixed time value (the period). In the sporadic task model, jobs may arrive at any time, separated by a minimum inter-arrival time.

Each task is characterized by the tuple (C_i, D_i, T_i) that represents: its worst-case execution time C_i, its relative deadline D_i, and its period or minimum inter-arrival time T_i. The utilization U_i of a task is calculated by the fraction $\frac{C_i}{T_i}$. R_i denotes the worst-case response time of a task, which is the largest interval of time from the release of a job until its completion.

Periodic task set may be classified as *synchronous* if all tasks that compose it are released simultaneously, or *asynchronous* if the tasks are not released at the same time. In the latter case, each task has an offset O_i that delays its first release.

The *hyperperiod* $H(\Gamma)$ of a task set defines the time at which the complete execution schedule will repeat itself, and it is given by the least common multiple of the task periods (Equation (1.1)).

$$H(\Gamma) = lcm(T_0, \cdots, T_n) \tag{1.1}$$

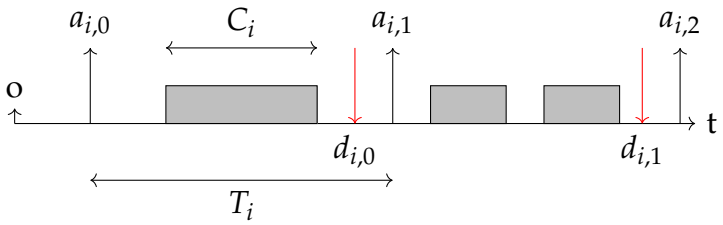


Figure 1.2.: The sporadic task model

We depict in Figure 1.2 a task model with its characteristics. The literature reports three levels of constraints on task deadline:

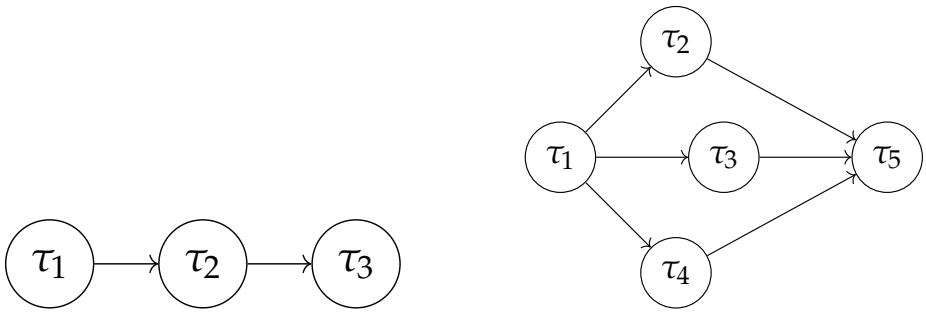
1. *Implicit deadline*: When the task deadline equals its period $(D_i = T_i)$.
2. *Constrained deadline*: When the task deadline is less than or equal its period $(D_i \leq T_i)$.
3. *Arbitrary deadline*: The task deadline may be less, equal or greater than its period.

Furthermore, a task presents the additional characteristics:

- **Arrival time** a_{ij}: it is the time at which the j-th job of task τ_i is activated and becomes ready for the execution; it is also denoted by the *release time*.
- **Start time** s_{ij}: it is the time at which the j-th job of task τ_i starts its execution.
- **Absolute deadline** d_{ij}: it is the deadline of the j-th job. The job must execute C_i units of execution time in interval $[a_{ij}, d_{ij}]$.
- **Density** δ_i: it's calculated as the ratio $\delta_i = \frac{C_i}{D_i}$. Nonetheless, if the deadline is implicit the density equals the task utilization U_i.
- **Jitter release** J_i: represent the maximum deviation time that a task can suffer before its start time.

1.2.1. Task Dependency

Many real-time systems can be modeled as a set of independent tasks when each of them generates an infinite number of job sequences. However, to meet functional requirements, the system must sometimes be modeled as a set of dependent tasks, which are therefore correlated and must follow an execution order with precedence constraints. Figure 1.3a shows the transaction model, where tasks are to be executed in a pipeline, with each task waiting for the completion of the previous one. Figure 1.3b represents the Directed-Acyclic-Graph (DAG) model, which allows more expressive dependencies.



(a) transaction-model (b) DAG fork-join model

Figure 1.3.: Task dependency levels

A DAG is composed of a set of vertices and edges. The vertices represent computing functions called *sub-tasks* which are typically chunks of the application's code. The edges model communications and precedence constraints between sub-tasks. An edge $e(v_i, v_j)$ represents a communication between two sub-tasks v_i and v_j; for every $k \geq 0$, the k-th instance of sub-task v_j cannot start execution before the $k - th$ instance of sub-task v_i has completed and the corresponding message from v_i has been received. We call this a *precedence constraint* between v_i and v_j.

Consequently, a vertex can be a *source vertex* if it has at least an outgoing edge; and it can be a *destination vertex* is it has one or more incoming edges. A *root vertex* has only outgoing edged and a *sink vertex* has only incoming edges, respectively. Finally, a DAG is *acyclic*: there is no closed cycle in the graph. A DAG is assigned a period T_i and it can be *periodic* or *sporadic*; it is also characterized by a relative

deadline D_i, that is the time by which all instances of the sub-tasks of the DAG must have completed from the corresponding DAG's activation.

1.3. Real-time systems scheduling analysis

Task scheduling defines how the sequence of jobs has to be orchestrated in order to access a resource which could be a calculation unit or a storage medium.

1.3.1. Scheduling algorithms classification

We can classify real-time scheduling algorithms according to the following characteristics:

Preemptive vs. Non-Preemptive

- Preemptive algorithms can suspend a running task on a processor, by saving its context, and replace it with another higher priority task.
- In non-preemptive algorithms, once a task has started its execution, it executes until completion with no suspensions, even when higher priority tasks arrive.

Static vs. Dynamic

- In static schedulers, all decisions are based on fixed parameters and assigned to tasks before their activation. They are also denoted by *compile-time* scheduling decisions.
- In dynamic schedulers, decisions are based on dynamic parameters that may change during the system execution. They are also denoted by *run-time* decisions.

Off-line vs. Online

- A scheduling algorithm is used off-line if it is executed on the task set before tasks activation. The schedule generated from this operation is stored and later executed by a dispatcher.
- A scheduling algorithm is used on-line if the scheduling decisions are taken at run-time and the schedule may change anytime a new task enters the system.

Optimal vs. Heuristic

- An algorithm is said to be optimal if it minimizes the cost related to a function defined over the task set.
- An algorithm is said to be heuristic if it is guided by a heuristic function that defines its scheduling decisions.

1.3.2. Scheduling characteristics

A real-time schedule is **correct** if all jobs of all tasks start execution no later than their arrival time, and complete their execution no earlier than their absolute deadline. A task set that is correctly scheduled by algorithm A is said to be *schedulable* by algorithm A.

Additionally to the scheduling classes, a scheduling algorithm has the following characteristics:

Optimality : a scheduler is denoted as optimal with respect to a system and a task model if it can schedule all of the task sets that comply with the task model and are feasible on the system. Davis and Alan Burns, 2011

Feasibility : A task set is said to be *feasible* if it exists a scheduling algorithm that generates a correct schedule.

Predictability : An algorithm is said to be *predicable* if during the system execution the jobs response time cannot be increased while the task parameters remain constant.

Comparability : comparing two task set by a given scheduling algorithm, there are three possible outcomes

- *Dominance*: Algorithm A is said to *dominate* algorithm B, if all the task sets that are schedulable by B are also schedulable by A and there are task sets schedulable by A, but not according to B.
- *Equivalence*: Algorithm A and B are *equivalent* if all the task set that are schedulable by B are also schedulable by A, and vice-versa.
- *Incomparable*: Algorithm A and B are *incomparable* if there are task sets that are schedulable by B and not by A, and similarly, there are task sets schedulable by A and not by B.

Sustainability : An algorithm is said to be *sustainable* associated to a task model, if and only if any task set remains schedulable when: (i) decreasing executions times, (ii) increasing periods or minimum inter-arrival times, and (iii) increasing relative or absolute deadlines.

1.3.3. Scheduling analysis

Assuming a scheduling algorithm and a task set, a *scheduling analysis algorithm* tests the schedulability of the task set with the scheduling algorithm. Typically,

the schedulability test analyses the worst-case scenario of the system. A given schedulability test can be classified as:

Sufficient test : A schedulability test is termed sufficient, with respect to a scheduling algorithm and a system if all of the task sets that are deemed schedulable according to the test are in fact schedulable.

Necessary test : conversely, a schedulability test is termed necessary if all of the task sets that are deemed unschedulable according to the test are in fact unschedulable.

Exact test : if the schedulability test is both sufficient and necessary.

1.4. Uniprocessor Scheduling

In this section, we introduce uniprocessor scheduling and the different scheduling algorithms associated with it. We will facilitate the functioning of the algorithms by bringing some simple examples.

1.4.1. Rate Monotonic Scheduling

The rate monotonic scheduling belongs to the *fixed-priority scheduling class* of algorithms. It simply assigns priorities to tasks according to their rates (inverse of the periods). Specifically, tasks with shorter periods will have higher priorities. We denote task priority by P_i.

Γ	C_i	T_i	P_i
τ_1	2	6	1
τ_2	3	8	2

Table 1.1.: A task set with RM priority assignment

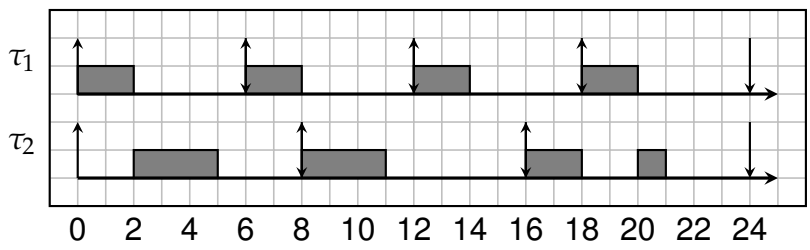


Figure 1.4.: The RM tasks scheduling

Example 1.4.1. Let define Γ a task set with two periodic tasks with implicit deadline (task characteristics are in Table 1.2)

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{2}{6} + \frac{3}{8} \approx 0.70 \tag{1.2}$$

When task set utilization U_i remains under $n(2^{1/n} - 1)$ (n represents the task set cardinality) Liu and Layland, 1973 all the task are schedulable. Thus, the latter value constitutes the feasibility upper bound of RM.

1.4.2. Deadline Monotonic Scheduling

The DM scheduling algorithm is similar to RM since they pertain to the same class of fixed-priority schedulers. RM assigns priority according to the relative deadline: tasks with shorter deadlines have higher priorities. The algorithm has been proposed by Leung and Whitehead J. Y.-T. Leung and Whitehead, 1982 to tasks with *constrained deadline*.

The feasibility of a task set with constrained deadlines can be guaranteed using the utilization based sufficient scheduling test:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1) \tag{1.3}$$

To find a necessary and sufficient schedulability test for DM, N. Audsley et al., 1993 proposed an efficient test for periodic tasks, called *response time analysis*. The proposed method computes the longest response time R_i of a periodic task τ_i at the critical instant, which is the sum of its computation time and the interference I_i of higher priority tasks:

$$R_i = C_i + I_i$$

where

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

Hence,

$$R_i^{(n+1)} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil \cdot C_j \tag{1.4}$$

The calculation of R_i in Equation 1.4 is performed iteratively by defining: $R_i^{(0)} = C_i$. The stop condition of the iteration is either when: $R_i^{n+1} = R_i^n$ or $R_i > D_i$. The system is declared as not schedulable when the latter condition occurs.

Γ	C_i	T_i	D_i	P_i	R_i
τ_1	5	10	9	2	9
τ_2	4	15	7	1	4
τ_3	6	30	14	3	15

Table 1.2.: A task set with DM priority assignment

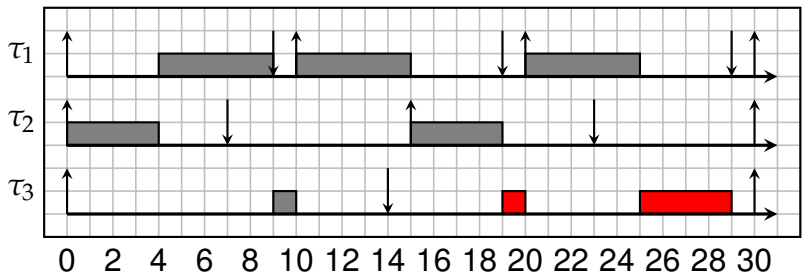


Figure 1.5.: The DM task scheduling (unschedulable case)

1.4.3. Earliest Deadline First

Earliest Deadline First (EDF) is an algorithm that selects jobs according to their absolute deadline. Specifically, jobs with earlier deadlines have a higher priority. Thus, the closer a task execution is to its deadline, the higher is its priority. The absolute deadline of a periodic task is calculated as follows:

$$d_{i,j} = O_i + (j-1)T_i + D_i \tag{1.5}$$

EDF is a dynamic priority assignment. Typically, tasks are executed in preemptive mode since the current active task can be replaced by a task with an earlier deadline.

Concerning the implicit deadline task sets ($T_i = D_i$), the schedulability analysis under EDF can be verified through the processor utilization factor. Thus, EDF is known to be optimal since the tasks remain schedulable when the processor utilization up to 100%, by the following sufficient condition:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{1.6}$$

Demand Bound Function

Under EDF, the analysis of periodic tasks with *constrained deadline* ($D_i < T_i$) can be performed using the *processor demand* metric. We refer this method to the work of Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell, 1990a.

The processor demand of a task τ_i in an interval $[t_1, t_2]$ is expressed by the amount of processing time $g_i(t_1, t_2)$ requested by the sequence of jobs activated in $[t_1, t_2]$ and must be completed in the same interval, such that:

$$g_i(t_1, t_2) = \sum_{r_{i,k} \geq t_1, d_{i,k} \leq t_2} C_i \tag{1.7}$$

However, when the task set is synchronous, which means all tasks are released simultaneously and activated at $t = 0$ (i.e: $\Phi_i = 0$ for all the tasks), we must verify the demand of all jobs in $[0, t]$. Consequently, we use the *Demand Bound Function* Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell, 1990a:

$$\mathsf{dbf}(t) = \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \cdot C_i \tag{1.8}$$

Thus, a synchronous set of periodic tasks with constrained deadline is schedulable by EDF if and only if Equation (1.9) is verified.

$$\forall\, t > 0 \quad \mathsf{dbf}(t) \leq t \tag{1.9}$$

1.5. Multiprocessor scheduling

This section introduces the multiprocessors systems and some terminology and notation used in multiprocessors scheduling research. Moreover, we relate a taxonomy of multiprocessors scheduling and develop each part in a dedicated section. Unlike uniprocessor scheduling, multiprocessor scheduling must face the two following problems:

- *The allocation problem*: on which processor a task should execute.
- *The priority problem*: when, and in what order should a job execute regarding the sequence of jobs of other tasks.

We can distinguish through Figure 1.6 the difference between the partitioned and the global scheduling when the former has a queue for each processor, whereas the latter gathers all the tasks into one queue and dispatches the tasks to processors.

On the other hand scheduling algorithms can be classified according to the task migration tolerance and priority changing in run-time (See Carpenter et al., n.d. for more details) as follows:

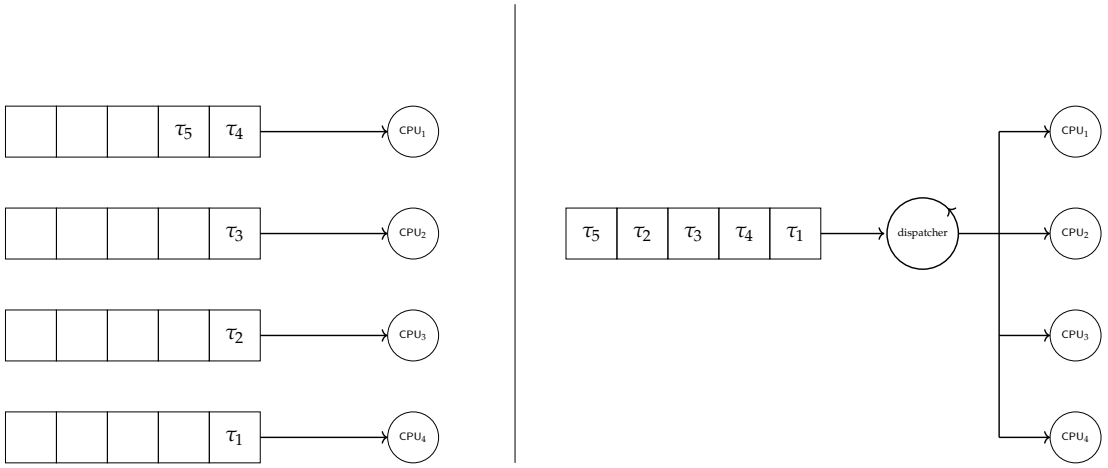


Figure 1.6.: Partitioned vs. global scheduling

Allocation-based classification

- *No migration*: each task is definitively allocated to a processor and no migration is permitted.
- *Task-level migration*: the jobs that pertain to a task may execute on different processors. However, each job can only execute on a single processor.
- *Job-level migration*: It's permitted to a job to migrate to a different processor during its execution.

Priority-based classification

- *Fixed task priority*: each task receives a fixed priority that will be applied to all of its jobs.
- *Fixed job priority*: the jobs of the same tasks may have different priorities, but each job has a signle static priority. EDF is considered from this class.
- *dynamic priority*: a single job may have different priorities during its execution, as Least Laxity First (LLF) scheduling.

1.5.1. Partitioned Scheduling

The partitioned scheduling doesn't allow task migration. At this fact, it has the advantages of:

- There is no migration cost to consider or any interference related to the migration process, which renders the schedulability analysis easy.
- If a task exceeds its worst-case execution time budget, it impacts only the tasks on the same processor.

- The partitioned approaches use a separate run-queue per processor, rather than a single global queue. Therefore, the overheads of queue manipulating are less important onto large systems.

Task partitioning strategies

Determining an optimal task partitioning is known to be an NP-Complete problem in the strong sense, where it is usually described as a bin-packing problem. Since each partitioning heuristic has its own strategy, several of them consist of two phases according to S. Baruah and Fisher, 2005:

- The pre-processing phase where the tasks are pre-sorted according to a task parameter, e.g., according to the shortest deadline in DM or the task utilization.
- The assignment phase, which consists of placing tasks into processors.

There are usual assignment strategies commonly used for task partitioning:

- First-Fit (FF): the task is always allocated to the first processor that fits its demand, while processors are increasingly sorted based on their ID.
- Best-Fit (BF): the task is allocated to the first processor while the set of processors is sorted in decreasing order with respect to their utilization.
- Worst-Fit (WF): similar to BF, but the processors are sorted in increasing order with respect to their utilization.
- Arbitrary-Fit (AF): each task is assigned to a processor in a random order.

1.5.2. Global Scheduling

Unlike the partitioned scheduling, this class of algorithms allows task migration from one processor to another. Global scheduling presents the following advantages compared to partitioned scheduling:

- It experiences only fewer context switches/preemptions when it is deployed, as the scheduler preempts tasks when there are no idle processors Andersson and Jonsson, 2000.
- The processor availability (space capacity) when a task executes less than its worst-case execution time, which can be used by all other tasks, not just those on the same processor.
- It is typically more appropriate to open systems, as there is no need to run load balancing or task re-scheduling when the set of tasks changes.

Global-EDF and Global-DM are examples of the adaptation of uniprocessor schedulers to global scheduling. The priority assignment logic doesn't change but it allows the m highest tasks/jobs to run in parallel and at the same time. Usually, when a task arrives and there is no idle processor, it preempts a lower priority task and executes on the freed processor.

1.5.3. Semi-partitioned Scheduling

This class aims at addressing the fragmentation of spare capacity in partitioned systems is to split a small number of tasks between processors Davis and Alan Burns, 2011. Andersson and Tovar, 2006 have proposed an approach to scheduling periodic task sets with implicit deadlines, based on partitioned scheduling, but splitting some tasks into two components that execute at different times on different processors.

Besides that, Andersson, Bletsas, and Sanjoy Baruah, 2008 developed the idea of job splitting to cater for sporadic task sets with implicit deadlines. In this case, each processor p executes at most two split tasks, one executed by processor $p-1$ and one executed by processor $p+1$.

1.6. Introduction to parallel programming

In this section, we introduce parallel processing and we will see how real-time programs are usually expressed in a low level language. Later, we explore popular libraries to write parallel programs.

1.6.1. Programming real-time systems

As we know, a real-time system is itself a set of periodic/sporadic tasks based on the Liu and Layland model Liu and Layland, 1973. We can write the C-equivalent of a real-time task by a function that has an infinite loop that generates the job sequence. At the end of the processing, the task sleeps until its next activation after the periodic time.

```
void taskA()
{
  while (1)
  {
    // activation

    processing() // task code

    sleep_until_next_period()
  }
}
```

1.6.2. POSIX Thread

The *pthread* library is known to be the most used low level API to create POSIX threads. It allows the creation of many threads that could run simultaneously on a multiprocessor. It also has primitive routines related to parallel programming

such as: Mutexes, Semaphores, Monitors. As the partitioned scheduling offers many advantages (See Section 1.5.1), the *pthread* library has an interface to allocate a thread to a given CPU_i called *pthread affinity*. We give the following example of thread allocation in C:

```
// thread creation
pthread_t thread;
pthread_create(&thread,NULL,th_func,NULL);
pthread_join(thread,NULL);

// set the pthread_affinity variables
cpu_set_t cpuset;

CPU_ZERO(&cpuset);       //clears the cpuset
CPU_SET(2, &cpuset); //set CPU 2 on cpuset

// allocate the thread on CPU 2
sched_setaffinity(thread, sizeof(cpuset), &cpuset);
```

1.6.3. Fork-join model

In some problems, we model the program as a series of functions that collaborate to solve the problem effectively. A DAG task can represent such modeling (See Figure 1.3b). OpenMP Chandra et al., 2001 provides an interface to implement parallel programs based on the fork-join model, which is a special-case of DAG structure composed of a single root node and a single sink node. The following algorithm divides the addition of two arrays of N elements onto 10 threads:

```
// define the number if threads
omp_set_num_threads(10);

#pragma omp parallel
#pragma omp for
for(i = 0; i < N; i++)
{
  c[i] = a[i] + b[i];
}
```

1.6.4. Message-passing interface

The message-passing interface (MPI) is a protocol for parallel programming, that allows point-to-point communication, without necessarily using shared memory. It's considered as another parallel programming style by comparing with the Fork-joint model (Section 1.6.3). The C-library that implements MPI protocol «mpi.h» offers many routines to allow such inter-processor communications. Hereafter, a brief example of an MPI implementation:

```
// define MPI environment
MPI_Init(NULL, NULL);

// data to send
int buf[256];

// send to a given processor (sender side)
MPI_Send(buf, sizeof(buf), MPI_INT, proc_id, 0, MPI_COMM_WORLD);

// receive from a processor (receiver side)
MPI_Recv(buf, sizeof(buf), MPI_INT, proc_id, 0, MPI_COMM_WORLD,
    MPI_STATUS_IGNORE);

// terminate the session
MPI_Finalize();
```

1.7. Conclusion

In this chapter, we introduced the real-time systems and scheduling theory. We defined the task models and presented feasibility tests for scheduling policies and we have briefly introduced the multiprocessors scheduling. Furthermore, we gave an overview of real-time programming interfaces and how such models are implemented in a real-time operating system.

In the next chapter, we will explore the multiprocessor execution platforms in-depth and we will present realistic models of such platforms.

2. On-chip Networks & Manycore architectures

Contents

2.1. Introduction

MPSoCs (MultiProcessors System-on-Chip) are popular as they bring many advantages in their ability of integrating heterogeneous components onto a single chip and their efficiency in performance/energy ratio. However, they show rapidly their limits when the chip embeds hundreds of processing engines (PEs) involving a high communication latency and bus saturation.

Network-on-Chip (NoCs) Benini and De Micheli, 2002 have been presented as a solution to the increasing demand for communication requirements thanks to their alternative bus topology. In NoCs, the PEs are connected each other and with the external memory via a *network* while the classical MPSoCs use a single communication bus. Figure 2.1 shows the difference between two *homogeneous* manycore: in the NoC, a PE communicates through the network by: (i) its adjacent router and (ii) its *network interface* (NI), while in usual bus-based multiprocessor, the PEs send their messages through the bus.

In this chapter, we introduce the NoC architecture and its elements. We will also present the NoC classes and those that are suited to real-time systems. We finally present some NoC that are available in the market and used by the industry.

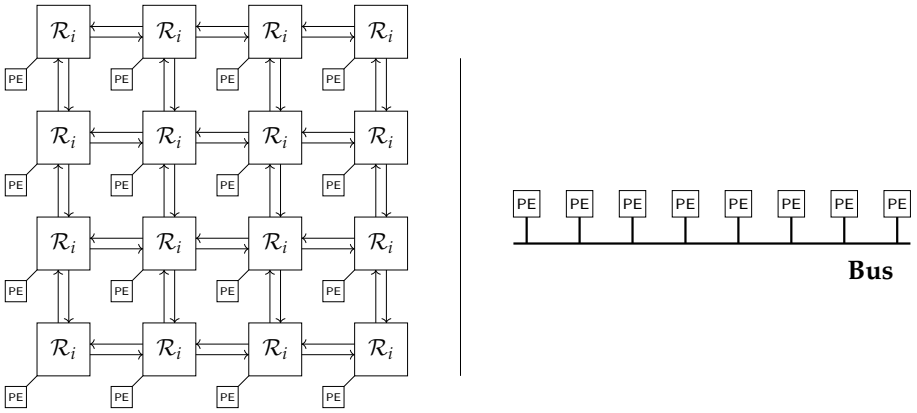


Figure 2.1.: 2D-Mesh Network-on-Chip vs. Bus-based Multiprocessor

2.2. Network-on-Chip classes

2.2.1. Preliminaries

The basic structure of a NoC platform, as shown in Figure 2.1, consists of a set of n processing engines $\{\mathsf{PE}_0, \cdots, \mathsf{PE}_n\}$ connected using a set of routers $\{R_0, \cdots, R_n\}$ by their associated network interface NI. Basically, the NoCs are appreciated for their communication performance. When a communication is established between two processing engines, a message $\mathcal{M}_i$ is sent through the network; the message is divided into data sub-units (Packets, Flits, $\cdots$) based on the communication

granularity. We present in the following the basic terms in relation with NoC performance:

- *NoC performance guarantees*: refers to the minimum of performance provided by the NoC, such as: maximum latency or the minimum throughput.
- *Throughput*: denotes the rate at which the NoC produces *Flits* (See Section 2.3.3) during a communication.
- *End-to-end latency*: refers to the total duration of a communication between two cores. Obviously, that includes blocking times and the interference that may come from concurrent flux.
- *Packet latency*: refers to the delay experienced by a packet, from its release from the source until its arrival time at the destination core.

Regarding the set of processing engines, the NoCs can be classified into three categories:

- **Heterogeneous**: The processors are different in terms of architecture and performance; hence the rate of execution of a task depends on both the processor and the task. However, not all tasks may be able to execute on all processors as it depends on task specifications.
- **Homogeneous**: The rate of execution of a task depends only on the speed of the processor. Thus, a processor with speed 2 executes a task at exactly twice the rate of a processor of speed 1.
- **Identical**: The processors are identical: hence the rate of execution of all tasks is the same on all processors.

2.2.2. Circuit-Switched NoCs

The circuit-switching (CS) approach uses the *resource reservation* approach, also referred by the name *connection-oriented* approach. Basically, a message is not be injected into the network until the entire path between the source and destination is reserved. Thus, a communication is achieved by the following three steps: (i) checking or reserving a free path; (ii) initiate the communication, (iii) cancelation or deallocation phase to release the path Pham et al., 2010. This approach presents the advantage of being bufferless. As the links are pre-reserved, buffers are not needed at each hop to hold packets that are waiting for allocation, thus saving energy. However, CS-NoCs experience poor bandwidth utilization which are not suitable for real-time critical systems.

2.2.3. Packet-Switching NoCs

Packet-switched (PS) NoCs, also denoted by connection-less NoCs, are promoted for the efficient bandwidth and network resource usage compared to CS-NoCs Hesham et al., 2017. In this class, during the communication, data are divided into packets which find their way according to the NoC configuration: the flow

control, the routing algorithm, an arbitration rule and the current traffic road. All these parameters will be detailed in Section 2.3.

2.3. Network-on-Chip elements

The Network-on-Chip architecture is very flexible and can be shaped according to the criticality level of the applications. Hereafter, we detail several NoC configurations.

2.3.1. Topology

The Network-on-Chip topology determines the physical layout and connections between nodes (also called tiles). It affects profoundly the network cost-performance, hence its importance. Moreover, the topology determines the number of hops (or routers) a message must traverse from the source to the destination core, thus it influences network latency significantly. Consequently, the number of hops affects directly the NoC energy consumption. Additionally, the topology dictates the total number of alternative paths between cores, which affects traffic workload and bandwidth utilization.

Since the first decision of NoC designers is the topology choice, it is useful to know the performance of the different topologies available. Here, we describe the several metrics that come in handy when comparing different topologies at design step.

Degree refers to the number of links at each tile. Obviously, it follows the number of neighbors that a tile is physically connected to. The degree is a useful metric of the network's cost when determining the implementation complexity. Thus, a higher degree requires more ports at routers, which involves a higher cost.

Hop count defines the number of routers that traverse a message from the source to its destination core. It is considered as a useful parameter to calculate the message latency since every on-chip communication has at least one router traversal.

Maximum channel load This metric is used to estimate the maximum bandwidth the network can support, or the maximum number of bits per second (bps) that can be injected by every tile into the network before its saturation.

Path diversity determines the number of possible paths that a topology could provide between source to destination core. Obviously, path diversity within the topology gives flexibility and more load-balanced traffic capability to routing algorithms.

Direct topologies

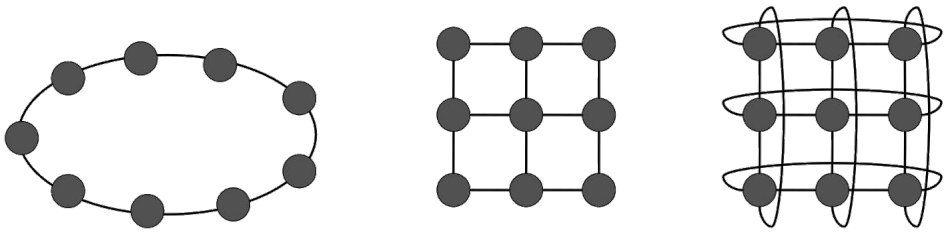


Figure 2.2.: Direct topologies: Ring, 2D-Mesh and torus

Ring, mesh and torus are classified as direct topologies (Figure 2.2). The latter two can be described as k-ary n-cube, where k is the number of tiles in each dimension, and n is the number of dimensions. For instance, a 4×4 torus is a 4-ary 2-cube with 16 tiles, while $4 \times 4 \times 4$ is a 4-ary 3-cube with 16 tiles. Finally, ring topologies fall into the torus family of k-ary 1-cube.

Indirect topologies

The butterfly topology is an example that pertains to indirect class. Butterfly network is described as k-ary n-flies. However, we distinguish two types of nodes: terminal and intermediate switches. Thus, such network would consists of k^n of terminal nodes (in our case, tiles) and n stages of $k^{n-1} \times k \times k$ of intermediate switch nodes.

The tree topology is logically a binary tree network in which resources increase for stages closer to the root node. In a tree, messages traverse the tiles until an ancestor is reached and then routed down to the destination; this permits the tree to take advantage of locality between communicating tiles. Figure 2.3 illustrates the two indirect topologies.

2.3.2. Routing Algorithms

The routing algorithm dictates the path that a message must follow during its traversal in the topology. Typically, routing algorithms are divided into three classes: deterministic, oblivious and adaptive. We draw a comparison of them in Figure 2.4.

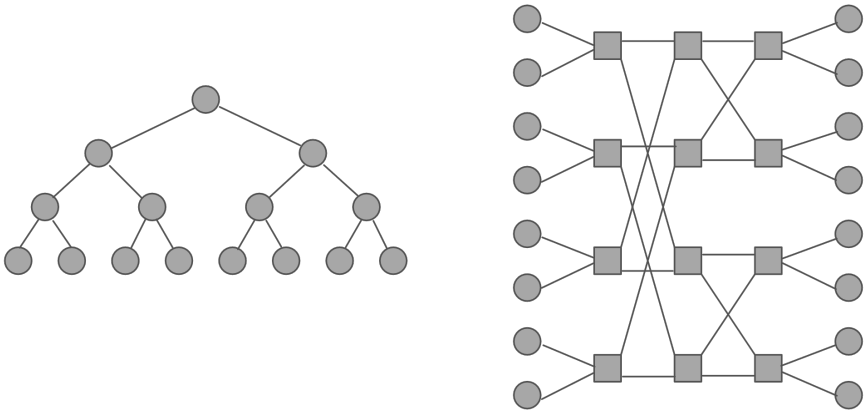


Figure 2.3.: Indirect topologies: Tree and butterfly

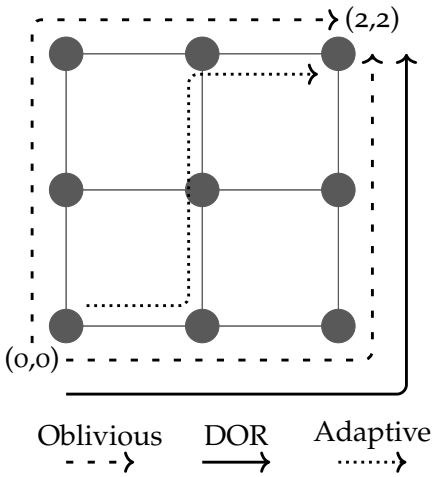


Figure 2.4.: We illustrate a communication between two tiles: from (0,0) to (2,2) in 2D-Mesh. *DOR* routes always the packets following the X and Y axis, while *oblivious* alternates the paths between X-Y and Y-X axis. Finally, *adaptive* follows a path guided by a heuristic.

Deterministic routing

A routing algorithm is said to be deterministic if the message sent from tile A to tile B takes the same path, independently of the current network state. The most common used deterministic algorithm in Network-on-Chip is *dimension-ordered routing* (DOR) due to its simplicity. In DOR, all messages from tile A to B traverse the same path dimension-by-dimension (X-Y routing) as follows: the message is first routed along the X-axis while each router has its 2-dimension coordinates. Thus, it is guided by routers until it arrives at the destination tile column, then it follows the Y-dimension.

This class of routing algorithm is appreciated since it offers a *clairvoyance* for the message path traversal and its latency.

Oblivious routing

Conversely to deterministic routing, this class behaves arbitrarily, where messages traverse different paths from tile A to B, but the path is designated regardless of the network traffic. For instance, a router could randomly choose an alternative path among the possibilities.

Adaptive routing

This class of routing algorithms is considered as more sophisticated, in which the path a message takes from tile A to B depends on the network traffic situation, and therefore, is adaptive. Additionally, an adaptive routing may choose a path according to a given heuristic, e.g.: the minimal routing algorithm selects only paths that experience a small number of hops between source and destination tiles.

2.3.3. Flow Control techniques

Flow control dictates the allocation of network buffers and links. Precisely, it determines when buffers and links are assigned to messages, the granularity at which they are allocated, and by which policy the network resources are shared among messages. A good flow control protocol permits a low message latency at low traffic workload by not imposing high overhead in resource allocation while experiencing low power consumption.

Communication granularity

In general, when a message delivered from a communication is injected into the network, it may be segmented into sub-units according to the flow control segmentation policy. We denotes three level of granularity as depicted in Figure 2.5.

Message-based flow control The *Circuit Switching* protocol pre-allocates links along the routing path to the entire message. A probe (a small setup message) is sent into the network to reserve the links while an acknowledgment message is returned to the source tile to confirm the success of the operation. After that, the sending operation can start. Further details are reported in Jerger and Peh, 2009.

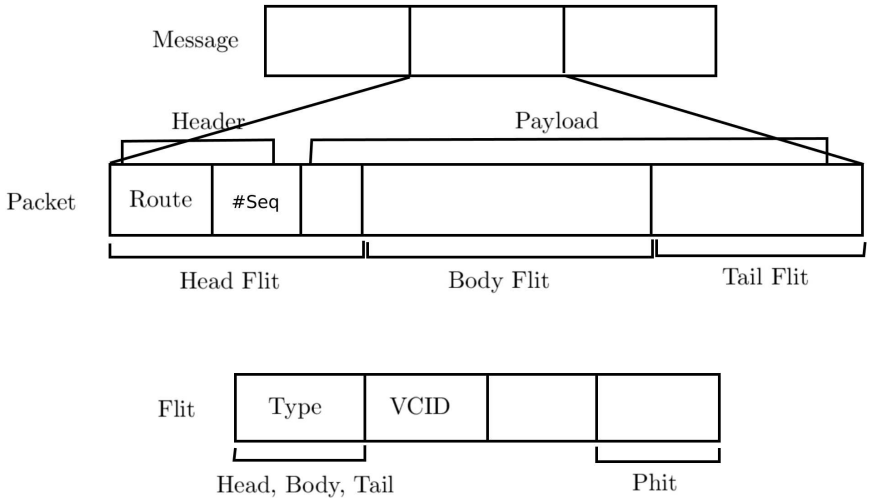


Figure 2.5.: Composition of different communication granularity: Message, Packet, Flit

Packet-based flow control The *Store and Forward* technique divides a message into multiple packets, where each packet is handled independently by the network. In this protocol, the routers must wait for the reception of the entire packet before forwarding it to the next router. As a result, it experiences long delays which makes this protocol unsuitable for NoC that are usually delay-critical. Conversely, the *Cut-Through* flow control allows transmission of a packet to the next tile while the packet is not completely received at the current router. Thus, latency experienced by a packet is drastically reduced.

Flit-based flow control The *Wormhole Switching* Ferrandiz, Frances, and Fraboul, 2009 is the most used protocol in Flit-based communications. For wormhole flow control, each packet is divided into *FLits*, which they move on to the next router before the entire packet is received at the current router, if there is sufficient buffering to receive them. This allows relatively small flit-buffers to be used in each router, even for large packet sizes. For instance, the wormhole switching is always associated with *Virtual Channels* which serves as buffers in input routers (See Section 2.4.1).

Flow control protocol	Links	Buffers
Circuit-switching	Messages	buffer-less
Store and Forward	Packet	Packet
Cut-Through	Packet	Packet
Wormhole Switching + Virtual Channels	Flit	Flit

Table 2.1.: Summary of flow control techniques

2.4. Network-on-Chip routers

Routers are referred as master components of NoC topology, since they are responsible for linking processing elements between them. Routers are defined by a micro-architecture and designed to meet latency and throughput requirements, under chip dimension area and power constraints. While routers design complexity increases with bandwidth demands, a trade-off between performance and design simplicity must be found in order to provide predictable time latency.

Thus, the router micro-architecture determines the overall network latency and per-hop delay. It also impacts the network energy and the circuit components activity. The design and implementation of such architecture is based on the choice of its components and their features. Moreover, the area footprint of the routers is also determined by its microarchitecture and the underlying components.

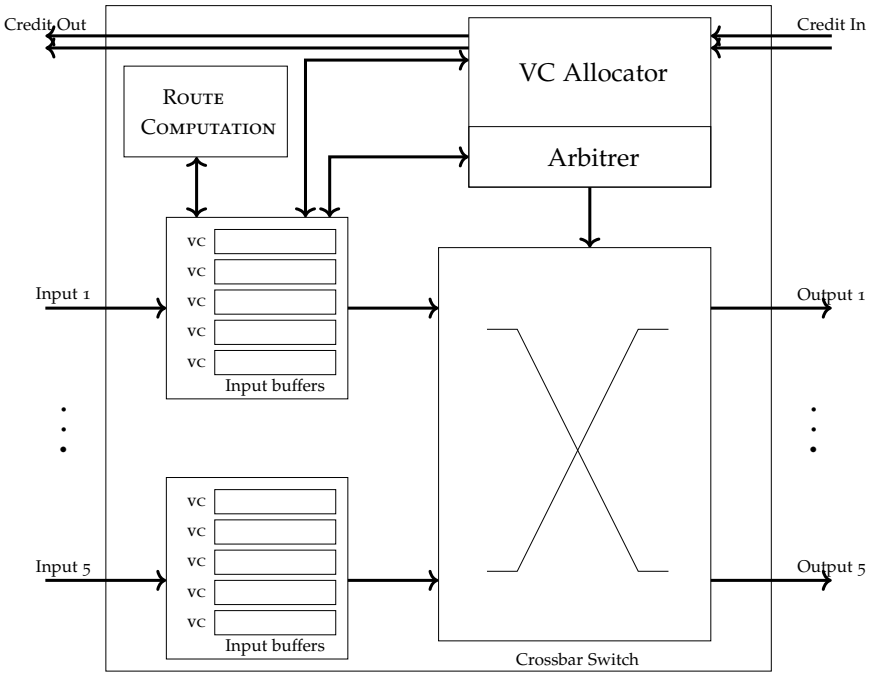


Figure 2.6.: Router architecture

Figure 2.6 depicts a state-of-the-art router microarchitecture assuming 2D-Mesh topology. The router is composed of: five input ports and output ports corresponding to the four neighboring directions and one toward the local processing element. The major components which constitute the router, additionally to the previous elements, are route computation logic, virtual channels allocator, switch allocator and arbiter.

2.4.1. Virtual Channels

Basically, a Virtual channel (VC) Dally et al., 1992 is a separated queue in the router input which hold data segment according to the flow control deployed

in the NoC; Obviously, it serves to buffer either packets of flits. VCs are first proposed as a solution for deadlock avoidance and Head-of-line blocking issue (more information are provided in Dally et al., 1992). Multiple VCs share the physical link between routers, while a VC is reserved for a single communication stream which holds it until the entire packet has moved on. Thus, when a packet holding a virtual channel becomes blocked, other packets can still traverse the physical link through other VCs. Thus, VCs increase the utilization of physical links and extend overall network throughput.

2.4.2. Allocators and arbiters

Allocator is a component responsible to match N requests to M resources, when generally the resources are limited ($N > M$). In routers, the resources are the VCs. In the case of wormhole switching protocol with multiple VCs per *input port*, the allocator has a mission to hold a VC for the message during its traversal and releases it when all message's flits moved to the next router. Conversely, the arbiter matches N requests to 1 resource, which is an *output port* for common routers. Thus, arbiters resolve conflicts when two or more communications flows would like to take the same output port.

Both allocators and arbiters are important since they can provide high network throughput and low latency when delivering high matching probability combined with an efficient arbitration strategy. Also, they must be fast and pipelinable so they can work under high clock frequency.

Priority-based arbitration

This strategy uses the traffic flows priority. Each message receives a priority value based on the scheduling policy. Hence, the allocator reserves a VC that equals the message priority, while the highest priority VC has straightforward access to the output port allowed by the arbiter. The priority-based policies are categorized by packet-level (non-preemptive) or flit-level (preemptive) while the wormhole switching belongs to the latter class.

Time-Division-Multiplexing

In TDM, resources are shared in time according to the global TDM table, which defines the periodic sequence of output port access by assigning to each VC a fixed time slot. Once data are injected into the network in their assigned time slot, TDM arbitration guarantees their traversal in a contention-free scheme without any intermediate stalls. Thus, through TDM, the communication latency is known *a priori* when the TDM table is statically defined.

2.5. Industrial Network-on-Chip

As on-chip networks being a well-diffused research area, we present in the following some COTS (commercial off-the-shelf) chips that have been designed to embed an on-chip network.

Intel TeraFLOPS

The TeraFLOPS processor Vangal et al., 2007 is a research prototype chip that is targeted at exploring future processor designs with significant amount of cores. It is a 65 nm, 275 mm^2 chip with 80 tiles that can run at frequencies up to 5GHz. Each tile is composed of 1 processing engine (PE) connected to its adjacent NoC router. The PE embeds 2 single-precision floating-point multiply-accumulator (FPMAC) units. The PE is interfaced with its router through router interface block (RIB). Two FPMACs in each PE providing 20 GigaFLOPS of aggregate performance, coupled with a maximum bisection bandwidth of 320 GBytes/s in the NoC enable the chip to realize a stable performance of 10^{12} floating-point operations per second (1.0 TeraFLOPS) while dissipating less than 100W.

Tilera TILE64 and TILE64Pro

The TILE64 architectures Bell et al., 2008 are products from Tilera that are suitable for high-performance embedded applications such as networking and real-time video processing. These chips support a shared memory space across the 64 tiles that compose the chip. Each tile consisting of two levels of cache and a 3-issue VLIW (Very Long Instruction Word) processor core connected to the four mesh networks. The TILE64 chip is designed at 90nm, 750MHz, has 5MB of on-chip cache and on-chip networks that provide a maximum bisection bandwidth of 2Tb/s with each tile dissipating less than 300mW.

Kalray MPPA2-256

Kalray MPPA2-256 (Bostan version) Dinechin et al., 2013 has been designed to fit timing constraints for critical applications with efficient energy footprint. It is composed of clusters of tiles, where tiles from the same cluster share a local memory. Thus, it integrates 256 PEs and 32 management cores grouped onto 16 compute clusters and 2 I/O clusters. Indeed, all the processing engines are based on 32-bits/64-bits VLIW core architecture. The NoC exploits the 2D torus topology to create a wide range of alternative paths between tiles.

2.6. Simulation tools

The expensiveness of Network-on-Chip pushed the community to instigate of simulators that behave like industrial NoCs, which permit the researchers to explore the architecture efficiently. Some of them are popular such as:

Noxim

Noxim Catania et al., 2015 is presented as an open, configurable, extensible, cycle-accurate NoC simulator developed in SystemC, which allows to analyze the performance and power figures of both conventional wired NoC and emerging WiNoC architectures.

HNOCS

HNOCS Ben-Itzhak et al., 2012a is an omnet++ based tool providing additionally to wormhole switching simulation, statistical measurements features such as: measurements at flit and packet levels, end-to-end latencies, network throughput, VC acquisition latencies, transfer latencies, etc. It also supports modeling of heterogeneous NoCs with variable link capacities and number of VCs per unidirectional port.

DARSIM

DARSIM Lis et al., 2010a is presented as a parallel, highly configurable, cycle-level network-on-chip simulator that implements the wormhole switching flow control. The parallel simulation engine offers cycle-accurate as well as periodic synchronization, permitting trade-offs between perfect accuracy and high speed with very good accuracy. It is also highly configurable including different topologies, bandwidth settings, crossbar dimensions, and pipeline depths.

2.7. Conclusion

We presented in this section the Network-on-Chip and its elements. The NoCs are very flexible and provide better performance. We related the NoC features and the industrial ones. We also present some simulators that are used in the literature.

In the following chapter, we will discuss about the task allocation onto NoC. Indeed, real-time systems are very rigorous and they require a special attention when they are executed on such architectures.

3. Task Mapping: Related work

Contents

3.1. Introduction

This chapter introduces and defines the task allocation problem. It is known that application mapping is one of the most important open problems in Network-on-Chip (NoC) research. We present the state-of-the-art of task mapping onto NoC by taking at each step different levels of task criticality: we first provide a state-of-the-art for prior contributions for the NoC mapping of non-critical applications and we end by presenting studies aiming the safety-critical applications.

3.2. Problem definition

The execution platform has a finite number of resources like processing engines as well as memories (registers, DRAM size, $\cdots$). However, in order to fit applications criteria such as compute performance and energy consumption, resources must be exploited following strategies that dictate how those resources should be allocated efficiently.

Typically, in modern software design, applications need to be partitioned (parallelized) into multiple sub-tasks in order to be executed concurrently on cores. This procedure is mainly performed by the application designers or through a specific tool as in Ceng et al., 2008. The parallelization of the applications requires adding task synchronizations, inter-task communications, and memory management. All those elements need an expressive abstract model, such DAG (Directed Acyclic Graph) in order to facilitate the system analysis and resource allocation.

We model the applications by a set of n DAG tasks $(\tau_1, \cdots, \tau_n)$, each one consists of a number of sub-tasks (vertices) communicating to each other. As result, an application includes a set of m communications described by a *source* sub-task that follows a path until a *destination* sub-task. For executing the system on a given hardware platform, the *mapping phase* determines on which core a sub-task has to be executed. Similarly, the mapping process can also determine how communication resources are allocated such that each task communicates properly.

The mapping schema can be defined as a set $\mathbb{M}$ of couples (sub-task, core) that associates a sub-task with a core:

$$\mathbb{M} = \left\{ (\{\tau_{0,0}, \cdots, \tau_{i,0}\}, PE_0), \cdots, (\{\tau_{0,n}, \cdots, \tau_{j,n}\}, PE_n) \right\} \tag{3.1}$$

Likewise, any other hardware resources M (Communication medium, memory, $\cdots$) can be exclusively or partially allocated to a given task during its execution:

$$\mathbb{I} = \left\{ (\{M_{0,0}, \cdots, M_{i,0}\}, \tau_0), \cdots, (\{M_{0,n}, \cdots, M_{j,n}\}, \tau_n) \right\} \tag{3.2}$$

In general, the mapping problem is categorized as an NP-hard problem due to the exponential number of possible combinations. The main difficulty is to efficiently explore the set of possible combinations (called the *design space*) in a reasonable amount of time. Thus, a well-tuned search algorithm helps to explore the thousands of possible mapping schemas and find the "best one" (according to some definition of "better") that fits the system performance requirement.

3.3. NoC Resource allocation for GP-tasks

In this section, we report the prior studies available in the literature of the task mapping problem on the Network-on-Chip architecture aiming at non-critical applications, such as multimedia applications. Figure 3.1 presents a taxonomy of mapping techniques.

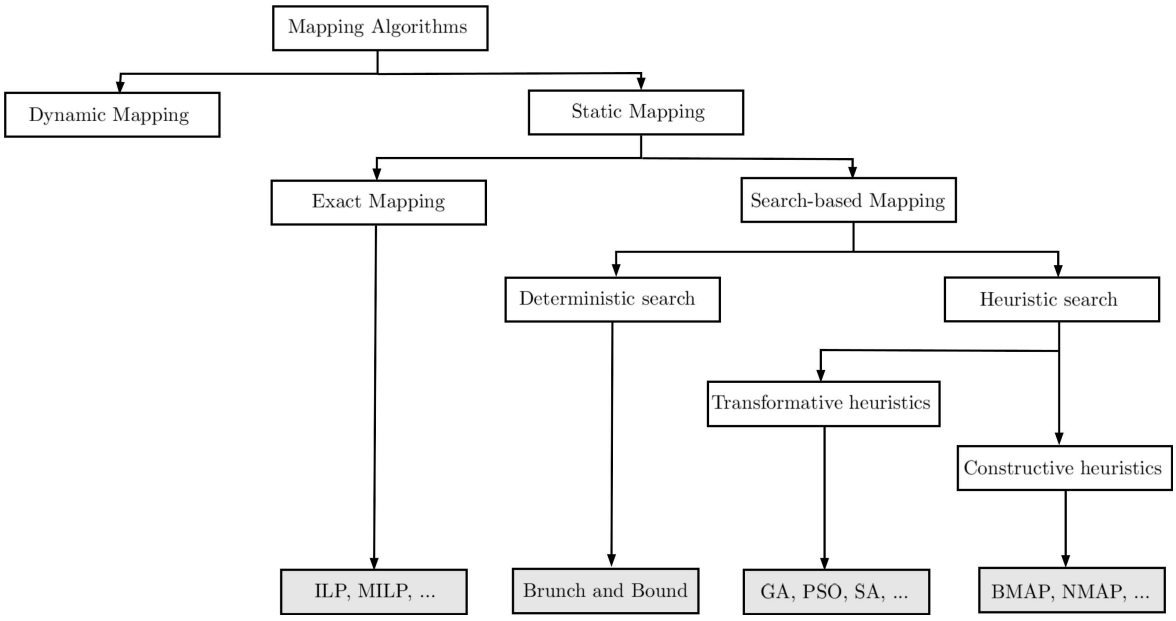


Figure 3.1.: Mapping algorithms taxonomy. ILP: Integer Linear Programming, MILP: Mixed Integer Linear Programming, GA: Genetic Algorithm, PSO: Particle Swarm Optimization, SA: Simulated Annealing.

3.3.1. Dynamic Mapping

The dynamic task mapping is an on-online strategy in which the ready tasks are allocated to the processors following the current state of the system at run-time. Obviously, based on the observed state of processors, the placement of tasks on NoC can change during the system execution.

The authors of Chen, Li, and Kandemir, 2007 have proposed a compiler based technique, which can perform all of tasks allocation, scheduling, data mapping

and packet routing, however with an important compilation time which may involve the system performance degradation.

In Carvalho, Calazans, and Moraes, 2007, the authors have proposed a heuristic consisting of an initial task mapping phase prior to the dynamic mapping phase. In the latter phase, they proposed the use of bin-packing techniques such as First-Fit (FF). In Chou and Marculescu, 2007, the authors presented a dynamic mapping technique based on multiple voltage level. In fact, the NoC is divided into multiple regions where each region operates by a fixed voltage differently from the others.

All the previous works target *heterogeneous* NoCs. In Chou and Marculescu, 2008, a dynamic allocation strategy has been proposed for *homogeneous* NoC. They used in their contribution the user behavior information in order to propose an adaptive allocation which is driven by the user needs. In Chou, Ogras, and Marculescu, 2008, they proposed a communication-aware mapping in which the heterogeneous NoC has multiple voltage level as in Chou and Marculescu, 2007. A mapping technique called DSM (Dynamic Spiral Mapping) has been proposed in Mehran, Khademzadeh, and Saeidi, 2008 which the task allocation follows the spiral path in the architecture. Thus, tasks are placed by beginning from the center to the edges of the NoC. It tries to allocate tasks close to each other in order to reduce the communication latency. The authors of Al Faruque, Krist, and Henkel, 2008 proposed an agent-based dynamic mapping in which the task allocation algorithm follows a distributed approach. Agents are defined as a small tasks that collect information about a NoC region while they collaborate between them to find a suitable mapping schema. On the other hand, in A. K. Singh, Jigang, et al., 2009 and A. K. Singh, Srikanthan, et al., 2010 the authors proposed a two-phases mapping heuristic: the first phase performs an initial task mapping on the first available position found that fits the task demand, whereas the second phase attempts to find for requesting tasks a better performance gain mapping in run-time.

Besides that, energy-aware heuristics are proposed in Mandelli, Ost, et al., 2011, Mandelli, Amory, et al., 2011 named lower energy consumption based on dependencies-neighborhood (LEC-DN), in which both communication distance between tasks and the communication volume (number of Llits) are considered.

3.3.2. Static Mapping

We define as static mapping also said *off-line* mapping, all the techniques in which the allocation of NoC resources for the tasks are decided before the system execution and is not changed thereafter. As illustrated in Figure 3.1, the static mapping groups various techniques that can be split into two major classes: Exact mapping and Search-based mapping. Hereafter, we list some contributions related to the two classes.

Exact Mapping

Most of exact techniques are based on mathematical frameworks when such techniques produce an optimal solution. At this aim, Rhee, Jeong, and Ha, 2004 proposed a MILP (Mixed Integer Linear Programming) formulation of task mapping while considering the task to core allocation and network interface modes. In Murali, Benini, and De Micheli, 2005, they proposed a task mapping into heterogeneous NoC with a memory footprint interest. Their technique uses a greedy mapping as first step followed my a MILP formulation to improve the tasks energy consumption. Likewise, Srinivasan, Chatha, and Konjevod, 2006 proposed a MILP-based method that attempt to optimize the routers links onto the NoC topology by generating a mapping schema with custom topology.

Ostler and Chatha, 2007 proposed a two stage ILP (Integer Linear Programming) for task and data mapping on SMP (Symmetric Multi-Processing) NoC with block multi-threading feature. Likewise, Ozturk, Kandemir, and Son, 2007 Explore the possibility of minimizing the NoC energy consumption by disabling certain communication links in SMP NoC while satisfying the performance constraint. In this continuity, Ghosh, Sen, and Hall, 2009, Huang et al., 2011b formulate a MILP that take care of mapping problem as well as operating voltage to minimize the NoC energy footprint. Lastly, Tosun, 2011 proposed an ILP formulation for energy-aware mapping in a clustered NoC.

Search-based Mapping

Deterministic search This category gathers the techniques that enumerate the solutions while exploring the search space. The Brunch-and-Bound (BB) searching method pertains to this category. Typically, this method is appreciated to its ability to explore the solutions in tree branches and bounding unallowable solutions. Also, it can be applied to smaller problems in which the search time can grow exponentially with the size of the problem.

Thus, the following contributions Hu and Marculescu, 2003a Hu and Marculescu, 2003b Hu and Marculescu, 2005 proposed an energy and performance-aware mapping under bandwidth reservation constraint. They applied a Brunch-and-Bound on top of NMAP (See 3.3.2) mapping technique to reach a better solution.

Heuristic search We can divide this class into two main categories: the transformative heuristics and the constructive heuristics. The former category starts from an existing mapping solution to arrive at better ones, whereas the latter starts from an empty solution and repeatedly extends the current solution until obtaining a complete solution. It is observed that constructive heuristics are normally much faster than the transformative heuristics Sahu and Chattopadhyay, 2013.

Concerning the transformative heuristics, the following contributions utilize the Genetic Algorithm (GA). This meta-heuristic is inspired from the natural selection

and uses biological inspired concepts such as mutation, crossover and selection to solve optimization problems (For more details, See Whitley, 1994).

Thus, the authors in Zhou, Zhang, and Mao, 2006 proposed a GA-based mapping solution that experience minimum average communication delay. Specifically, the initial solution is chosen randomly with the average waiting-time as a fitness function. Afterward, a new generation is issued from a multi-point crossover, in which the chromosome with a low waiting-time is selected and included in the crossover, then, the mutation operation is performed. Also, Ascia, Catania, and Palesi, 2004 proposed a pareto-based multiobjective GA that attempts to provide a task mapping with a trade-off between performance and energy-consumption. Similarly Bhardwaj and Jena, 2009 have proposed a multiobjective GA aiming at proving a task mapping with a minimal of energy consumption and the required bandwidth of NoC.

Besides that, the Particle Swarm Optimization (PSO) technique has been used to find a best task mapping. This collaborative technique is inspired by the social behavior of bird flocking (Further details in Poli, Kennedy, and Blackwell, 2007). Thus, The authors in Sahu, Venkatesh, et al., 2011 presented PSMAP, a PSO-based task mapping strategy both static and dynamic cost experienced by a 2D-mesh NoC, when they define the particle as a possible mapping cores to the routers. A prior work of J. Wang et al., 2011 used the Ant Colony optimization (ACO) for task mapping into NoC in order to reduce the bandwidth requirement.

In the constructive heuristics side, the NMAP technique has been proposed in Murali and De Micheli, 2004, which is a mapping technique coupled with minimum path routing that aims to minimize the average communication delay. basically, this technique begins by iteratively associating the cores having the most communication demand to the nodes having the maximum neighbors. Then, a core is randomly selected and mapped to the node that minimizes the communication cost with it (the core). This procedure is repeated until all cores are mapped. In Srinivasan and Chatha, 2005, a two phases heuristic has been proposed for a low energy 2D-mesh NoC. The first phase serves to map cores to different routers by using a graph bi-partitioning, while the second phase attempts to find a minimal path between source and destination to each communication.

3.4. Safety-critical real-time tasks allocation

The task allocation problem for safety-critical real-time constraints has been studied for different Network-on-Chip configurations as well as for different levels of criticality. However, during the design space exploration, only *schedulable* solutions are kept and considered, even though, other criteria such as energy consumption can be added to the objective. We illustrate in Figure 3.2 the logical chain of task mapping.

Hereafter, we address a related work that highlights the main contributions provided by the literature regarding the mapping of hard real-time applications on NoC.

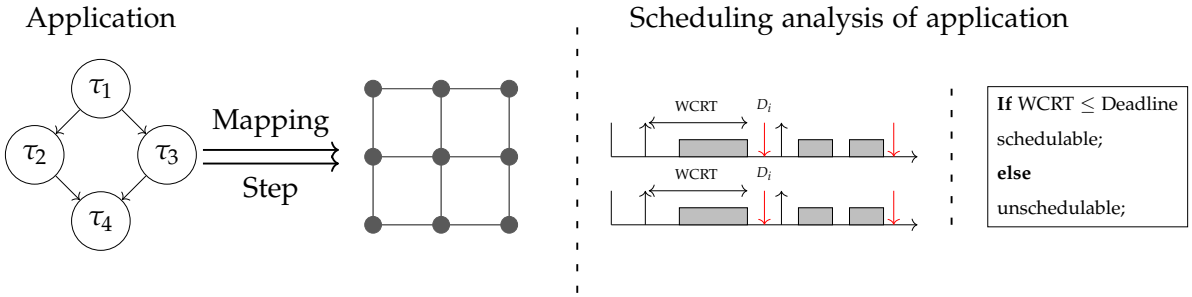


Figure 3.2.: Safety-critical systems application mapping workflow. In this schema, the second analysis phase relates only WCRT-based analysis, where other analysis methods, such as DBF can be applied.

Several works deal with priority-preemptive wormhole NoCs and are the main subject of the following papers. Some authors have proposed techniques based on Genetic Algorithms (GA) and they differ on optimization objectives. Indeed, they target a memory-aware mapping as in Still and L. S. Indrusiak, 2018 or using adaptive routing Norazizi Sham Mohd Sayuti, Hazwani Mohd Ridzuan, and Hilmi Abdullah, 2019 or even combined with efficient priority assignment technique for communication Sayuti and L. S. Indrusiak, 2015b, all based on GA technique. On the other hand, Authors of Jesse Barreto de Barros, Ayala-Rincon, and Quintero, 2019 used an adaptive GA that heuristic parameters may change during the computation, while Indrusiak et al. Bonilha, Santos, and L. Indrusiak, 2014 used the utilization factor as an optimization metric on adaptive GA.

Also, GA can be employed with many objectives at the same time. Thus, Bruch et al. Bruch et al., 2017 employed a multi-objective GA to reduce the deadline-miss ratio of tasks while optimizing the number of VCs used in communication. Likewise, Sayuti and L. S. Indrusiak, 2013 employed a multi-objective GA to reduce energy consumption with a fitness function.

Besides that, many works have used Integer Linear programming (ILP) combined with Simulated Annealing meta-heuristic to reduce energy consumption in heterogeneous NoC Huang et al., 2011a, or to find an energy-efficient mapping with minimal execution time Chatterjee et al., 2017. Additionally, Shi et al. Zheng Shi and Alan Burns, 2010 proposed a mapping with a greedy algorithm combined with an efficient priority assignment to communication flows, whereas Jessé Barreto de Barros, Sampaio, and Llanos, 2019 used Particle Swarm Optimization for the same purpose based on the utilization factor. Similarly Sayuti and L. S. Indrusiak, 2015a used a heuristic called *Constructive Task Mapping* which consists of task clustering and their allocation according to the utilization factor.

Few studies address the problem of real-time task mapping on TDMA/RR NoC. Cardona et al. Cardona et al., 2018 proposed a framework (NoCo) using ILP to map tasks and a heuristic to find the best NoC configuration regarding the routing

policy and an efficient slot assignment on VC. Perret et al. Perret et al., 2016a applied constraint programming to map task on *Kalray MPPA-256* NoC. However, applying exact techniques takes a long time on large dimension problems since task mapping is an NP-complete problem. Lo et al. Lo et al., n.d. experienced a static mapping of a common avionic system (Health Monitoring System) on *Kalray MPPA-256*. However, those previous works do not take into consideration the allocation of multiple applications (Multiple DAGs).

Another important aspect of task allocation is to handle the communication onto the NoC. In fact, when a sub-task finishes its execution, it often sends data through the network to the following sub-task which can be allocated on a different core. If several communications share the same path, this leads to congestion and therefore increases the latency. In general, congestion has a non-negligible impact on the response time. Harde et al. Harde et al., 2018 provided an exact budgeting solution for VC to respect the communication deadline. Also, Nikolic, Hofmann, and Ernst, 2019 proposed techniques for TDMA quantum assignment to minimize communication latency, and to find a near-optimal VC slot assignment using heuristics.

However, addressing a task mapping on the entire NoC as a unique region is time consuming because the complexity increases very rapidly with the number of tasks and processors.A better solution is to slice the NoC into sub-regions which contain a subset of processors. At the end, this reduces the allocation of a task for only a limited region and non for the entire NoC. Strategies to slice and choose the region size are called *core clustering*. Thus, Giannopoulou et al., 2016 proposed a simulated annealing-based algorithm for mixed-criticality task mapping onto a hardware-level clustered NoC.

3.5. Conclusion

In this chapter we explored the literature of task mapping on Network-on-Chip. Indeed, due to the important complexity involved by such problems, many meta-heuristics have been deployed as well as exact techniques to find near-optimal or optimal solutions. Also, as NoC are widely configurable, the contributions cited previously may be different.

The next part of the dissertation presents our contributions addressing the real-time task mapping on NoC architectures.

Ref	Proposed Technique	Optimization Goal	Arbitration Policy	Wormhole	Routing	Core Clustering
Still and L. S. Indrusiak, 2018	Genetic Algorithm	ET	Priority preemptive	x	Deterministic (XY)	
Norazizi Sham Mohd Sayuti, Hazwani Mohd Ridzuan, and Hilmi Abdullah, 2019	Genetic Algorithm	ET	Priority preemptive	x	Deterministic (XY)	
Sayuti and L. S. Indrusiak, 2015b	Genetic Algorithm	ET	Priority preemptive	x	Deterministic (XY)	
Jesse Barreto de Barros, Ayala-Rincon, and Quintero, 2019	Genetic Algorithm	ET	Priority preemptive	x	Deterministic	
Bonilha, Santos, and L. Indrusiak, 2014	Genetic Algorithm	ET	Priority preemptive	x	Deterministic (XY)	
Bruch et al., 2017	Multi-objective Genetic Algorithm	ET, EC	Priority preemptive	x	Deterministic (XY)	
Sayuti and L. S. Indrusiak, 2013	Multi-objective Genetic Algorithm	ET, EC	Priority preemptive		Deterministic (XY)	
Huang et al., 2011a	ILP + Simulated Annealing	ET, EC	Priority preemptive	x	Deterministic (XY)	
Chatterjee et al., 2017	ILP	ET, EC	Priority preemptive	x	Deterministic (XY)	
Zheng Shi and Alan Burns, 2010	Greedy Algorithm	ET	Priority preemptive	x	Deterministic (XY)	
Jessé Barreto de Barros, Sampaio, and Llanos, 2019	Particle Swarm Optimization (PSO)	ET	Priority preemptive	x	Deterministic (XY)	
Sayuti and L. S. Indrusiak, 2015a	Specific Heuristic	ET	Priority preemptive	x	Deterministic	
Cardona et al., 2018	ILP + Stochastic heuristic	ET	TDMA	x	Adaptive + Deterministic	
Perret et al., 2016a	Constraint Programming	ET	TDMA		Deterministic	x
Giannopoulou et al., 2016	Simulated Annealing	ET	TDMA	x	Deterministic	x

Table 3.1.: Summary of the main existing approaches considering real-time applications (ET: Execution Time, EC: Energy Consumption)

Part II.

Contributions

4. Comparative study: Priority Preemptive and Round-Robin Arbitration

Contents

4.1. Introduction

The Network-on-Chip architecture brings many advantages over the classical bus-based architecture especially by the path diversity offered by its topology. However, this diversity increases the complexity of communication time estimation while routers have own policy to handle the network traffic. When executing real-time tasks on a NoC-based architecture, the shared data has to be routed between PEs where communicating tasks are allocated. The needed time to route data from its source to its destination is called communication latency. Latency has to be bounded to ensure that each task instance has been executed without violating the real-time constraints (within its time window).

NoC components such as routers and network interfaces are designed to maximize network utilization without taking into account predictability and temporal behavior of communications, which make them not suitable to real-time systems. An arbitration schema is required to control the access of communication links between PEs, where this mechanism increases the complexity of the NoC. Several works in real-time community have proposed architectural modifications to reduce the worst case of latency bounds. However, these works are not properly compared against each other, due to a lack of tools (especially simulation).

This chapter presents an event-based simulator and analysis tool for periodic and sporadic real-time communications, allowing to compare the different approaches proposed in the literature against each other by simulation and analysis. We provide also a comparative study of fixed priority and time division arbitration protocols and their impact on latency and schedulability.

The remainder of this chapter is organized as follows: in the next section, we report NoC communication mechanisms. Section 4.3 is reserved to present architecture and communications models. Our Simulator is briefly described in Section 4.4. Section 6.5 presents the different approaches to analyze the behavior of fixed priority and TDMA arbitration protocols provided by our tool. Results are discussed in Section 4.6, we draw conclusion in Section 4.7.

4.2. NoC switching and routing mechanisms

Each communication consists of a message, communication source and destination. First, each message $\mathcal{M}_i$ is decomposed into a set of packets ($\mathcal{M}_i = \{\mathsf{P}_{i1}, \mathsf{P}_{i2}, \cdots\}$), further, packets are forwarded separately from a router to another.

Wormhole switching is the mechanism that describes how a packet moves forward from a router to another. In the wormhole switching, each packet P is broken into small pieces called FLITs[1], $\mathsf{P} = \{\mathsf{F}_1^P, \mathsf{F}_2^P, \cdots \mathsf{F}_n^P\}$.

[1] FLow control unITs

The first flit F_1^P, called the header flit, holds needed information to packet routing (for example, the destination address) and sets up the behavior of all other flits associated within the same packet. Final flit, F_n^P is called the tail flit. Between the header and the tail flit, flits are called body flits.

In wormhole switching, flits are stored in VCs[2]. Each VC is either idle or allocated to only one packet. A header flit can be forwarded to the next router if at least next router has one idle VC. The VC allocator decides where each packet is stored (selects the idle VC for the header flit). When the VC is selected, the header flit locks the VC. Body and tail flits can be forwarded to the same VC as the header, using a credit-based flow control. When the tail flit is routed, it frees the latest VC it has occupied.

In a NoC architecture where each router is composed by one VC per port, if two header flits or more are blocked in a circular dependency, it may lead to a deadlock. Thus, using multiple virtual channels allows to reduce wormhole blocking.

Routing is an operation performed in router to determine which is the next hop of packets. In this contribution, we focus only on XY routing. The packets are first transferred in X-direction and then in Y-direction in order to transfer them from the source router to the destination router.

4.3. System model

Network on Chip are tightly coupled with computing elements such processors, accelerators, etc. When executing real-time applications on NoC-based architecture, tasks are allocated onto cores such that all real-time requirements are respected. The respect of real-time constraints implies achieving real-time communications in a bounded time. In this contribution, we are not interested in task allocation, we focus only on real-time communications. In this section, we present hardware architecture design and task models used in the rest of this chapter.

4.3.1. Architecture model

NoC topology and architecture

We model a NoC architecture as a set of $m \times m$ routers. Routers are connected to each other in a 2D-mesh topology. Each Router is connected to its left, right, top, bottom neighbor except those on the edges.

$\mathcal{R}_{jm}$ denotes the router at row j and column m. For example $\mathcal{R}_{22}$ denotes the router in the second line and second column. It has for neighbor $\mathcal{R}_{21}$ on the left, $\mathcal{R}_{23}$ on the right, $\mathcal{R}_{12}$ on the top, $\mathcal{R}_{32}$ on the bottom. Routers are linked between

[2]Virtual Channels

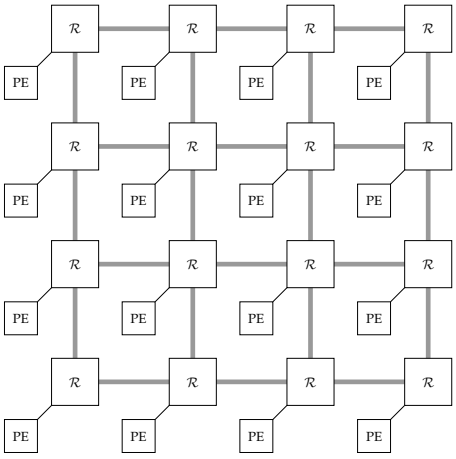


Figure 4.1.: 2D-mesh NoC architecture

them by an unidirectional links $\lambda_{(i,j)(m,n)}$, where this latter is communicating link from $\mathcal{R}_{ij}$ to $\mathcal{R}_{mn}$.

Router architecture

A router is the main unit in a network-on-chip. Mainly it has k ports, one for each neighbor. In 2D-mesh, each router has 5 in-ports and 5 out-ports connected to its neighbors. The fifth port is local port and connected to the local PE. Each router is composed of:

- In/Out-ports: are the physical media that links a router with its neighbor routers.
- Virtual Channels (VC): are message buffers. It can contain a fixed number of flit arriving from a neighbor, stored for a while, before being sent to its next destinations (routed). the number of VC per port denoted by |VC| allows a router to support multiple communications using the same port at the same time
- VC Allocator: is the entity responsible of selecting for a given packet, the VC where it is going to be stored.
- Route Computation: is the unit responsible to select the output port for any given packet. Here is implemented XY routing.
- Crossbar: is the unit able to route the non-conflicting communications. By conflicting communication, we denote the packets available at the same time in a given router and need to be routed using the same output port
- Arbiter: the unit that *schedules* outports for conflicting communications. It can be configured to by selecting an arbitration policy to ensure tighter bounds of latency for real-time communications.

4.3.2. Communication model

Real-time tasks are recurrent. Liu and Layland Liu and Layland, 1973 are the first to model recurrence in real-time systems by defining a real-time task by its deadline, period and offset. The Liu and Layland model is the most used in real-time community and industry. We use similar model for real-time communications. Let Γ denote a set of n communications $\Gamma = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_n\}$. Each communication is sporadic and can generate an infinite number of recurrent messages. It is characterized by $\mathcal{C}_i = (\mathcal{M}_i, \mathsf{D}_i, \mathsf{T}_i, \mathcal{R}_s, \mathcal{R}_d)$ where:

- $\mathcal{R}_s, \mathcal{R}_d$ represent the source and destination routers respectively.
- $\mathcal{M}_i$ is the message size sent from $\mathcal{R}_s$ to $\mathcal{R}_d$.
- T_i is the communication period. It represents the minimum arrival time between two communications. Thus, the communication $j+1$ can not start before at least T_i time from the arrival of communication j.
- D_i is the communication relative deadline. The j^{th} communication from $\mathcal{R}_s$ to $\mathcal{R}_d$ has to be finished within the time interval $[\mathsf{a}_{i,j}, \mathsf{a}_{i,j} + \mathsf{D}_i]$ where $\mathsf{a}_{i,j}$ is the time where communication $\mathcal{C}_i$ is requested from the router.

In our tool, task parameters are specified using YAML input file.

4.4. Real-time Communication simulator

Simulation tools allow faster exploration of design space and quick evaluation of the design choices performance. Recently, a lot of simulators Lis et al., 2010b; Vyas, Choudhary, and D. Singh, 2013; Binkert et al., 2011; Bolotin et al., 2004; Möller, L. S. Indrusiak, and Glesner, 2009; Fazzino, Palesi, and Patti, 2008; Benhaoua, AmitKumar Singh, Abou El Hassan Benyamina, et al., 2015; Ben-Itzhak et al., 2012b; Moraes et al., 2004; Hossain et al., 2007 have been proposed to explore design choices in NoC-based architectures at different abstraction and precision levels. For example, GPNoCSIM Hossain et al., 2007 and DynaMapNocSim Benhaoua, AmitKumar Singh, Abou El Hassan Benyamina, et al., 2015 are event-based simulators written in JAVA, the first focuses on communications, whereas the second focuses more on the task allocation, both at high level of abstraction. Hermes Moraes et al., 2004, is low-level simulation tool written in VHDL. It allows to emulate design choices on FPGA boards, however it is time consuming to explore design choices and evaluate their performances.

However, none of the simulators, cited above, offers a support for real-time communications, neither periodicity or recurrence in general. The latter are designed for non-critical systems and need lot of modifications to make them support real-time communication protocols. Thus, we propose a new simulation tool for real-time communication protocols.

Our simulator is modular, and extensible. A first version is available[3] and is still under continual upgrading and development to include extra-features. In this section, we describe how the simulator has been designed.

4.4.1. Packages

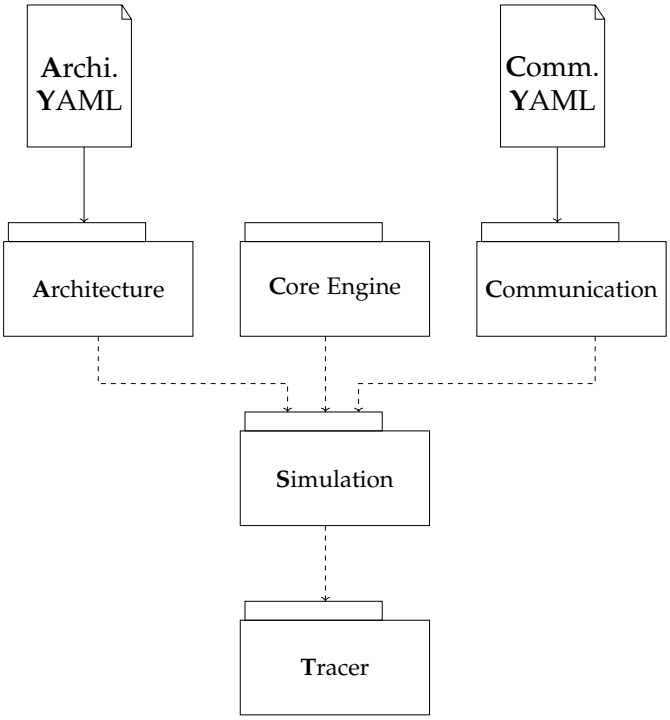


Figure 4.2.: Package diagram

Our simulation tool is compound of 5 packages, detailed in the follow :

Architecture : It contains all the classes and structures to define a NoC. We focus mainly on 2D Mesh topology. However, our design can be extended to specify other topologies like torus and ring.

Communication: This package defines the router communication structures and their parameters. Allowing to define periodic and aperiodic communications, message decomposition, structure and several extra-real-time parameters.

Core engine: Here are implemented all the algorithms that contributes in NoC functioning. They are mostly implemented by different interfaces (CROSS BAR, VC allocator, Arbiter, $\cdots$).

Simulation : The simulation package contains the simulation core. It is responsible for events and time management. It contains discrete event-based simulation engine. The simulation engine can be re-used for any other simulation purposes.

[3] https://github.com/chawki27000/retina-sim

Tracer : Responsible of registering at cycle level, all actions taken onto each router and the state of each VC at each time instance are save in a log file. It allows also to automatically generate formatted results and some predefined plots using PGF-plots.

4.4.2. NoC & Simulation Engines

Our simulator handles three types of events :

- MESSAGE_ARRIVAL : This event occurs to signal a communication between two routers. It starts from message splitting to reach flit granularity until generate next events.
- SEND_HEAD_FLIT : This event handles flit header forwarding, by defining the next hop router, reserving an idle VC, triggering arbitration (if conflict occurs) and generate the next events if all is done without errors.
- SEND_BODY_TAIL_FLIT : Finally, this event handles body or tail flit. It checks free space in the allocated VC, blocking flit sending if no space available and releasing VC if the current flit is tail.

Algorithm 1 Simulation

```
noc_f: YAML FILE
task_f: YAML FILE
parse_noc(noc_f)
create_events(task_f)
sort_events_time(task_f)
while (event_list ≠ ∅) do
    e = select_next_event()
    update_clock()
    switch e do
        case MESSAGE_ARRIVAL :
            Process_message(e)
        case SEND_HEAD_FLIT :
            send_header_flit(e)
        case SEND_BODY_TAIL_FLIT:
            send_body_tail_flit(e)
    if (sim_time_finished) then
        empty_event_list()
    end if
end while
```

Algorithm 1 shows different steps and main function calls of the simulator. It starts by parsing a NoC settings and communication scenario by instantiating all periodic or aperiodic communications. Further, it sorts all events and loops on them one by one. The clock is updated when the event is handled. The simulation ends when simulation time reaches the hyper-period or events list is empty.

4.5. Analysis

In this section, we present priority-based and TDMA-based arbitration mechanisms and their analysis.

4.5.1. Fixed priority

Shi et al. Z. Shi and A. Burns, 2008 propose to assign a priority to virtual channels. Therefore a communication in VC of priority p, is selected by the arbiter before any communication in all VCs of priority less than p. Moreover, if the communication in VC of priority p has already started, it can be interrupted by communications in VCs of higher priority, allowing preemptions. Once the high priority communication finishes, the low priority communication resume their forwarding in a classical preemption scheme. Authors in Z. Shi and A. Burns, 2008 provided worst case communication latency bounds analysis. Tighter bounds has been provided by Xiong et al. Xiong, Lu, et al., 2016 and Xiong, F. Wu, et al., 2017 by distinguishing two types of interference : *Upstream* and *Downstream*. In a 2D-NoC topology, Upstream interference is caused by conflicting messages arriving from the south port, whereas the downstream interference is caused by incoming communication from north port. Equation 4.1 have been proposed by the authors of Xiong, F. Wu, et al., 2017 to compute a communication latency bounds.

$$R_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i + J_j + I_{ji}^U}{T_j} \rceil (C_j + I_{ji}^D) + C_i \tag{4.1}$$

Equation 4.1 is iterative:

It starts by assuming $R_i^{(0)} = \sum_{\forall \tau_j \in S_i^D} (C_j + I_{ji}^D) + C_i$, where

- S_i^D is a set of messages that constitutes a direct interference;
- I_{ji}^U and I_{ji}^D are the set of conflicting messages belonging to Upstream and Downstream indirect interference respectively.

This equation converges if fixed point is found ($R_i^{(n+1)} = R_i^{(n)}$) or if latency is already greater than the deadline, resulting to a deadline miss ($R_i^{(n)} > D_i$).

Although this approach is *easy* to implement and analyze, it presents major limitations. First, preempting a communication can be a costly operation. In fact, the router is forced to create and schedule a tail FLIT for the preempted message so it can continue onward its routing and a new head FLIT for the FLITs that are still not yet forwarded. Moreover, a real implementation of such solutions requires as many VCs per input port as number of priorities (tasks). However, increasing the number of VC (which are mainly buffers) increases drastically the chip size and lead to heat dissipation and voltage problems. One solution

may be to limit the number of scheduled tasks or to manage the priorities in a hierarchical schedule scheme.

4.5.2. Time division multiple access

Abbreviated by TDMA, the second main approach aims to share output port between conflicting communications based on time sharing. Therefore, each VC has its own service time slots, where FLITs within that VC are forwarded. Several works have been interested in optimizing the time slot size and slot assignment. An exhaustive survey can be found in Hesham et al., 2017.

Under TDMA, each communication is achieved in isolation to the others. Its latency can be computed as shown in Equation 5.1.

$$R_i = \frac{L_i}{n_{slot}} \cdot \frac{\Delta}{\delta_i} + H_i \tag{4.2}$$

Where :

- L_i : number of flits in the message.
- n_{slot} : The amount of data sent in one slot (1 Flit by default).
- Δ / δ_i : The total number of slots in a *TDMA cycle* / the assigned slot number.
- H_i : Hop number between $\mathcal{R}_s$ and $\mathcal{R}_d$.

This approach is more complex and requires implementing timers and their synchronization mechanisms in the routers. However, it provides isolation of FLIT forwarding, therefore prevents "miss-behaving" communications from monopolizing the network. Furthermore, it does not require other modifications to VC structures, nor to arbitration protocols. However, communication-to-VC assignment mechanisms must be achieved offline.

4.6. Experiments

In this section, we present a wide set of synthetic simulations to study performances of fixed-priority-based approach against TDMA based approaches in terms of worst case latency bounds and resource augmentation. ReTiNAS is used to simulate the real-time task communication behavior. Communications latency change drastically when all conflicting communications are active at the same time. Therefore, we start by describing how conflicting communications are generated.

4.6.1. Conflicting communications generation

First, a communication com is selected between src and dst . Further, the route between src and dst is computed using XY-routing algorithm. Later, stressing communications which has a goal to create contention in either X-axis or Y-axis or both, are iteratively generated until reaching an input *c*ontention rate threshold.

The contention rate is computed as follows:

$$\sum_{\tau_i \in ConflictSet} \frac{C_i}{T_i} \tag{4.3}$$

where ConflictSet represents the higher priority tasks that share at least a link with the current task as depicted in Figure 4.3.

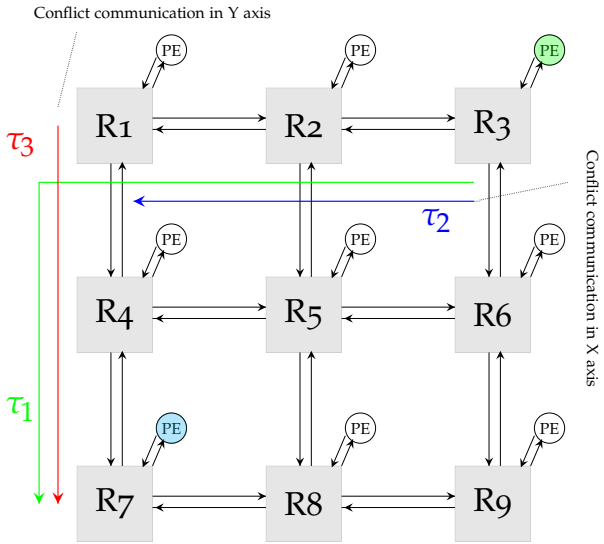


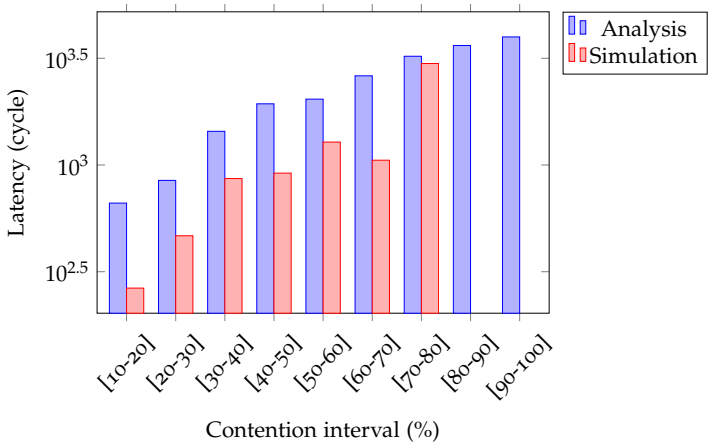
Figure 4.3.: Interference on the message path

4.6.2. Simulation

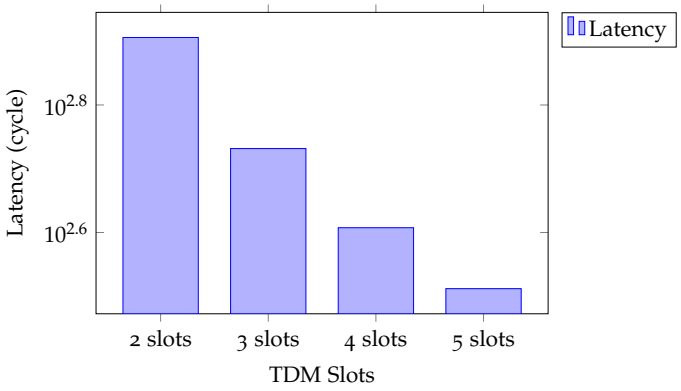
We perform a wide set of experiments using ReTiNAS, using a personal computer including Intel i5-7200U CPU and 8GB of RAM. All experiments where achieved on the NoC configuration summarized on Table 4.1.

Topology	4x4 2D-Mesh
VC per InputPort	6
Buffer size per VC	10 Flits
Periods (cycle)	[1000, 1500, 2000, 3000, 4000, 6000]
TDM Slots	[4, 2, 3, 5, 3, 3]
Message	8 Packets (10 Flits each)
Number of hops	5

Table 4.1.: NoC configuration and Communication details



(a) Simulation/Analysis in Preemptive Priority Arbitration



(b) Analysis by TDMA Arbitration

Figure 4.4.: Experimentation results in different arbitration mode

Figure 4.4a reports the worst case latency obtained by simulation against the one obtained by analysis as a function of contention rate for fixed priority approach. When the contention is low, the analysis tend to compute very large latency bounds compared to the measured latency using the simulator. In fact, the analysis assume always the worst case of task arrivals, therefore a congestion level that may never be reached when tasks execute. The more contention, the high latency is, in both simulation and analysis. In fact, in such scenario, the worst scenarios can often happen when all tasks are activated, therefore the simulation

worst bounds are close to the analytical ones. However, analysis is still slightly over-estimating the worst case latency bounds.

TDMA is a contention-free arbitration approach as all communications are executed in isolation. Therefore, it is not interesting to study the impact of congestion on latency itself. Thus, in Figure 4.4b, we report the latency using *TDMA* approach as a function of the time slot size. As expected, the bigger time slot size, the shorter latency is. However, communications may not be able to be served using unlimited time-slot size. The problem is to define the *e*xact time slot per time period for a given task to respect its real-time constraints.

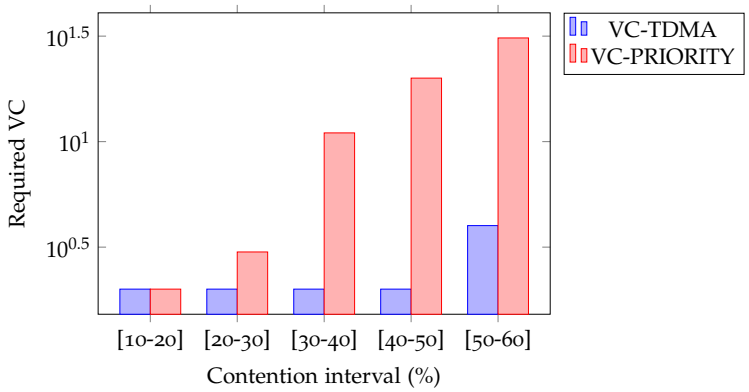


Figure 4.5.: NoC Resource Augmentation

Figure 4.5 represents the average VC number required, for fixed priority and TDMA, to respect deadlines for the same tasks as a function of contention rate. This allows us to compare the efficiency of each approach regarding the respect of real-time constraints. We highlight that we modify the time-slot size to have schedulable tasks for TDMA. Therefore, even if results reported here show the efficiency TDMA and fixed priority, it does not allow a fair comparison of TDMA against fixed priority.

When contention is low, we can see that TDMA approach need the same number of VCs as fixed priority. When the contention rate is increased, the congestion increases, therefore more of higher priority messages are scheduled and more VCs are needed to keep the latency less than the deadline. TDMA is not contention sensitive, therefore always requires less VCs compared to fixed priority. The gap between both keeps increasing as the contention is increased.

4.7. Conclusion

In this chapter, we presented the design and implementation of a real-time network-on-chip communication simulator and analysis tool. We provided also an overview of techniques to perform real-time communication in a NoC architectures. We presented a comparative study of TDMA and fixed priority approaches

as a function of worst case latency and resource augmentation bounds. In the next chapter, we introduce the computing tasks alongside communication tasks. Therefore, we present a preliminary allocation algorithm of the real-time DAG (Directed-Acyclic-Graph) tasks.

5. DAG tasks allocation on NoC

Contents

5.1. Introduction

When executing real-time systems onto a NoC-based architecture, the schedulability depends on (i) the task allocation, and (ii) inter-task communication. Each has received a particular attention in the real-time community. The task allocation has been widely addressed for bus-based architecture, and effective algorithms have been proposed to optimize several goals, such as energy H.-E. Zahaf, Abou El Hassen Benyamina, et al., 2017, resource utilization, etc. Regarding the communication, several research works have proposed NoC architectural modifications to reduce worst case inter-task communication time. Mainly they can be classified to: priority-based and TDMA-based. The previous chapter (4) reports an experimental and analytical comparison between arbitration protocols onto NoCs.

Few works only have focused on both allocation and communication issues at the same time for real-time systems. This problem is NP-hard in the strong sense. It is extremely time consuming to compute optimal solution as the design space is very large. Therefore, it is more convenient to design efficient heuristics to achieve fast and efficient design-space exploration. One way, to design allocation heuristics, is to assign intermediate artificial deadlines for every task, therefore allowing tasks to be analyzed independently from each other.

The artificial offset-and-deadline-assignment techniques have not been proposed for NoC-based architectures. In this chapter, we propose a new approach to allocate real-time tasks into NoC architecture by extending classical deadline assignment heuristics, as well as classical bin-packing heuristics for real-time DAG tasks. We first, propose an extension of bin-packing heuristics to the NoC-based multiprocessors. Further, we propose a new deadline assignment techniques for NoC based architecture, and provide analysis for allocating both communications and tasks.

The rest of the chapter is organized as follows: in the next section, we report the related work. System and architecture model are presented in Section 5.3. Further, we present independently real-time approaches for (i) bounding communication latency within NoCs, and (ii) deadline assignment for multiprocessor based architectures. Section 5.4.3 reports how real-time tasks can be executed within NoC-based multiprocessors using TDMA for communication and classical bin-packing heuristics for task allocation. Results of the both simulation and analysis and their discussions are presented in Section 5.5. We draw our conclusions in Section 6.7.

5.2. Related Work

Network-on-chip interconnection paradigm has many advantages compared to classical bus-based interconnections, such as its scaling capacity. However, as hundreds of computing units are embedded on the same chip, the task allocation

is a complex issues compared to classical bus-based architecture compound of few processors. Finding an optimal task allocation is an NP-complete problem and it has been the subject of several works. For non-real-time tasks, authors in Benhaoua, AmitKumar Singh, A. Benyamina, et al., 2014, Hansson, Goossens, and Rădulescu, 2005, Sahu, Shah, et al., 2013, Srinivasan and Chatha, 2005 have proposed offline (static), as well as on run-time mapping strategies (on-the-fly) for both task and communications. However, none of these proposals considers critical real-time system requirements.

In bus-based real-time systems, communication latency is analyzed and included in the task worst case execution time as it doesn't not much depend on the task allocation itself. However, such techniques can not be used in the context of NoC-based architecture, communication depends drastically on the task allocation, therefore it must be considered independently from the task worst-case execution time. Several techniques have been proposed that can be classified into two categories. The first uses *TDMA* (Time-division multiple access) to regulate the medium communication access, while the second assign a priority to each message, and a scheduling policy is applied. A comparative study of these techniques are reported in Benchehida, Benhaoua, H. E. Zahaf, et al., 2019 using simulation and analysis. Harde et al., 2018, Nikolic, Hofmann, and Ernst, 2019 proposed techniques for TDMA quantum assignment to minimize communication latency, i.e find a near-optimal VC slot assignment using heuristics. An exhaustive survey can be found in Hesham et al., 2017.

However, none of these works above combine task and communication assignment at the same time. In this chapter, we propose an allocation approach for both tasks and communication using TDMA for communication and classical bin-packing heuristics for allocations.

5.3. System Model

In this section, we present our architecture and our task models. First, we overview NoC architectures basic concepts. Further, we model real-time tasks using DAGs.

5.3.1. Architecture Model

The NoC is an interconnection paradigm which has been introduced in Benini and De Micheli, 2002 to overcome the bus-based interconnection limitations. It links the processors through an embedded network. Each processor is connected to a router via a network interface. Each couple of a router and processor is called a tile. As in classical networks, tiles can be arranged according to different topology. 2D-Mesh is the most used topology in NoCs. It aligns tiles in a square

matrix of $m \times m$. Let $\mathbb{M}$ denote a set of tiles, $\mathbb{M} = \{\mathcal{A}_1, \cdots, \mathcal{A}_{m \times m}\}$. Each tile δ_i is connected to typically 4 neighbors, except those at the network edge.

When exchanged between tiles, a message $\mathcal{M}_i$ is first split into several *Packets* ($\mathcal{M}_i = \{P_{i1}, \cdots, P_{ij}\}$). In Wormhole switching, each packet is broken into small data units, called *FLIT* (FLow control unIT). Within a tile, incoming FLITs are stored in local buffers, called virtual channels (VCs). Further, They are moved forward to their next destination according to a routing algorithm. We consider, in this study a XY routing. Flits are first moved to the next router in the X-axis (horizontal way) until reaching the destination router column, and then in the Y-axis (vertically) until reaching the target router. XY routing is simple, deadlock-free and deterministic. When two or more VCs are routed to the same output, they must be arbitrated, as the physical links support only one communication at same time. In this chapter, we use a TDMA-based arbitration mechanism. TDMA assigns for each VC a given number of slots, where it is served in a round robin fashion. That is, each VC has a guaranteed service time. TDMA ensures isolation between communications as well as a predictable contention free service time.

5.3.2. The DAG task model

A *DAG task* is a Directed Acyclic Graph, characterized by a tuple $\tau = \{T, D, \mathcal{V}, \mathcal{E}\}$, where: T is the task period, it represents the minimum inter-arrival time between two consecutive activation of task τ ; D is the relative deadline, all sub-tasks of τ must complete not later than D time units from its arrives; $\mathcal{V}$ is a set of nodes that represent *sub-tasks*. The set $\mathcal{E}$ is the set of edges of the graph $\mathcal{E} : \mathcal{V} \times \mathcal{V}$.

A sub-task $v \in \mathcal{V}$ is the basic computation unit. It represents the elementary chunk of the task code that be implemented by a thread and executed by any computing elements within $\mathbb{M}$ in our architecture. Every sub-task v is characterized by its worst-case execution time $C(v)$. An edge $e(n_i, n_j) \in \mathcal{E}$ models a precedence constraint (and related communication) between node n_i and node n_j and it is characterized by the maximum amount of the data (in FLIT number) exchanged by its source node n_i and destination node n_j, denoted by $\mathcal{M}(n_i, n_j)$.

The set of *immediate predecessors* of a node n_j, denoted by $\mathsf{pred}(n_j)$, is the set of all nodes n_i such that there exists an edge (n_i, n_j). The set of *predecessors* of a node n_j is the set of all nodes for which there exist a path toward n_j. If a node has no predecessor, it is a *source node* of the graph. In our model we allow a graph to have several source nodes. In the same way we define the set of *immediate successors* of node n_j, denoted by $\mathsf{succ}(n_j)$, as the set of all nodes n_k such that there exists an edge (n_j, n_k), and the set of *successors* of n_j as the set of nodes for which there is a path from n_j. If a node has no successors, it is a *sink node* of the graph, and we allow a graph to have several sink nodes.

Example 5.3.1. Let consider $\tau = \{T = 100, D = 80, \mathcal{V}, \mathcal{E}\}$ be a task. The dag, $\mathcal{V}, \mathcal{E}$, are reported in Figure 5.1.

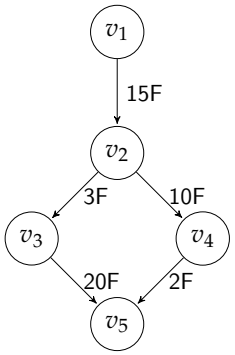


Figure 5.1.: An example of a dag task

The task has only one source node, therefore when the task arrives, sub-task v_1 is activated. Sub-task v_2 can not start its executions before v_1 finishes its execution and that v_2 has received completely 15 Flits sent by v_1. v_1 can not be activated again before at least 100 time units have elapsed. When v_2 finishes it sends 10 flits to v_4 (respectively 3 Flits to v_3). v_5 must finish its execution not later than 80 time units from the task arrival.

5.4. Real-time allocation and schedulability

Meeting timing constraints for a set of DAG real-time tasks requires allocating properly their sub-tasks and communications to different tiles. As these sub-tasks communicate, they are forced to respect an execution order dictated by the precedence constraints imposed by the graph structure. Therefore, every sub-task must wait for the completion of its immediate predecessors and their communications before it can start. Analyzing this behavior is complex, because of the large number of tiles to consider. The number of solutions is equal to $m \cdot m \cdot |VC| \cdot \sum_{\tau \in \mathcal{T}} |\mathcal{V}(\tau)|$ where $|VC|$ is the number of VCs per port, thus the design space is extremely large and cannot be completely explored to find optimal solutions in a reasonable time. In the following, propose an efficient methodology to allocate sub-tasks to PEs and communications to VCs to achieve fast design space exploration while meeting real-time constraints.

Our algorithm starts by allocating sub-tasks using classical bin-packing heuristics and by doing fast schedulability checking, as described in the next section. Later, it assigns to every communication a virtual channel (VC) where it will be served. Finally, to reduce the complexity of dealing with precedence constraints directly, we impose intermediate offsets and deadlines on each sub-task, i.e. for each sub-task, we compute an artificial activation time and an artificial deadline, and we ensure by analysis that if every sub-tasks is executed within its artificial activation time and artificial deadline, the task where it belongs will respect its deadline. In this way, precedence constraints are automatically respected. Further, the schedulability for every tile can be checked independently using classical single-core analysis, which can be found in the literature of real-time systems, for

both fixed priority and EDF. We detail every step in our allocation algorithm in the following sections.

5.4.1. Task allocation

This work uses classical bin-packing heuristics Best-Fit (BF) and Worst-Fit (WF) as disclosed in Algorithm 2.

It starts by sorting tasks according to order, that is either by deadline, or utilization (Line 3). Later, it selects the task on the top of the ordered task list, let it be τ. For every sub-task v in τ, the algorithm selects a sub-set of tiles where v is allowed for allocation (Line 6), (according to Definition 5.4.2 and Theorem 1). Further, the eligible tile list is sorted according to the bin-packing allocation heuristics (Line 7), BF for increasing utilization order and WF for a decreasing utilization order. A fast schedulability test is achieved to find the first tile allowing a schedulable allocation. If all eligible tiles have been investigated without finding an allocation that satisfies the schedulabilty test, the system aborts on fail. Otherwise, our algorithm moves to the next sub-task. When all sub-tasks have been allocated, our algorithm moves to the next task. When all tasks have been allocated, Algorithm 2 achieves deadline assignment for every task (Lines 20-22), by subtracting properly the communication latencies from the available slack time, as described in Section 5.4.3.

This procedure allows our algorithm to convert a complex allocation problem to multiple single-processor schedulability problem, for which well-known techniques can be found in the literature of real-time systems (Lines 23-27). If schedulability fails in a tile, the algorithm aborts on fail, otherwise the sub-task and communication assignment allows respecting timing constraints.

The function select_eligible_tiles (Line 6), returns a list of couples processor-VC where the sub-task and communication can be feasibility allocated according to the necessary schedulability test in Theorem 1.

5.4.2. Communication latency

We adopt for this contribution TDMA-based communication. Therefore, synchronously on all routers, one VC is served for its quantum. It provides isolation of FLIT forwarding, therefore prevents *miss-behaving* communications from monopolizing the network. However it requires synchronization mechanisms in the routers. Under TDMA, each communication latency between vertex v and v' $e(v, v')$ can be computed as shown in Equation (5.1).

$$\mathsf{lat}(VC, e) = \frac{TF(\mathcal{M}_i)}{n_{slot}} \cdot \frac{\Delta}{\eta_i} + H_i \tag{5.1}$$

Where :

Algorithm 2 Bin-packing allocation

```
1: input: Γ: set of tasks, alloc : TEXTBF or WF, order : D or U
2: output: A list of tuples (sub-task, Processor, VC)
3: sort_tasks_by(order)
4: for (τ ∈ Γ) do
5:     for (v ∈ τ) do
6:         eligible_list = select_eligible_tiles(v)
7:         sort_tiles(alloc, eligible_list)
8:         allocated = false
9:         for (p ∈ eligible_list) do
10:            if ((u(v) + U(Γ_p) ≤ 1) then
11:                add_sub − task_to_taskset(v, Γ_p)
12:                allocated = true
13:            end if
14:        end for
15:        if (allocated == false) then
16:            return FAIL
17:        end if
18:    end for
19: end for
20: for (τ ∈ Γ) do
21:    assign_deadlines_and_offsets(ø)
22: end for
23: for (p ∈ P) do
24:    if (check_schedulability(p) == FAIL) then
25:        return FAIL
26:    end if
27: end for
28: return success
```

- $TF(\mathcal{M}_i)$: Total number of Flits contained in the message.
- n_{slot} : The amount of data sent in one slot (1 Flit by default) in Virtual channel VC.
- Δ / η_i : The total number of slots in a *TDMA cycle* / the assigned slot number.
- H_i : Hop number between δ_{source} and $\delta_{destination}$.

Once the allocation of every sub-task is achieved, our algorithm computes all communications costs according to Equation (5.1) Lu and Jantsch, 2007. Further, schedulability can be checked. We describe how to isolate the schedulability is checked for each sub-task by the mean of deadline and offset assignment.

5.4.3. Deadlines and offsets assignment

Many authors have proposed techniques to assign intermediate deadlines and offsets to task graphs. In this chapter we report the two of the most used techniques, *proportional share* and *fair share*, reported in H.-E. Zahaf, Capodieci, et al., 2019.

Most of the deadline assignment techniques are based on the computation of the execution time of the critical path. A path $\pi_x = \{v_1, v_2, \cdots, v_l\}$ is a sequence of sub-tasks of task τ such that:

$$\forall v_l, v_{l+1} \in \pi_x, \exists e(v_l, v_{l+1}) \in \mathcal{E}.$$

Let $\Pi(\tau)$ denote the set of all possible paths of task τ. The critical path $\pi_{crit}(\tau) \in \Pi(\tau)$ is defined as the path with the largest cumulative execution time of the sub-tasks.

In contrast to classical deadline assignment techniques, We define the slack $\mathsf{Sl}(\pi, \mathsf{D})$ along path π as a function of the execution time of its sub-tasks and also of the communications latency that must be achieved between the sub-tasks of path π.

$$\mathsf{Sl}(\pi, \mathsf{D}) = \mathsf{D} - \sum_{v_l \in P} C(v_l) - \sum_{\substack{v_l \in \pi \\ v_{l+1} \in \pi}} \mathsf{lat}(\mathsf{VC}, v_l, v_{l+1})$$

where VC is the allocated virtual channel for communication between v_l and v_{l+1}.

The assignment algorithm starts by assigning an intermediate relative deadline to every sub-task along a path by distributing the path's slack as follows:

$$\mathsf{D}(v) = \mathsf{C}(v) + \mathsf{calculate\_share}(v, \pi)$$

The calculate_share function computes the slack for sub-task v along the path. This slack can be shared according to two alternative heuristics:

- **Fair distribution:** assigns slack as the ratio of the original slack by the number of sub-tasks in the path:

$$\mathsf{calculate\_share}(v, \pi) = \frac{\mathsf{Sl}(\pi, \mathsf{D})}{|\pi|} \tag{5.2}$$

- **Proportional distribution:** assigns slack according to the contribution of the sub-task WCET in the path:

$$\mathsf{calculate\_share}(v, \pi) = \frac{\mathsf{C}(v)}{\mathsf{C}(\pi)} \cdot \mathsf{Sl}(\pi, \mathsf{D}) \tag{5.3}$$

Once the relative deadlines of the sub-tasks along the critical path have been assigned, we select the next path in order of decreasing cumulative execution time, and assign the deadlines to the remaining sub-task by appropriately subtracting the already assigned deadlines. The complete procedure has been described in Y. Wu, Gao, and Dai, 2014, and is not reported here for space constraints.

Let $\mathsf{O}(v)$ be the offset of a sub-task with respect of the arrival time of the task's instance. The sum of the offset and of the intermediate relative deadline of a sub-task is called *local deadline* $\mathsf{O}(v) + \mathsf{D}(v)$, and it is the deadline relative to the arrival of the task's instance.

The offset of a sub-task is set equal to o if the sub-task has no predecessors; otherwise, it can be computed recursively as the maximum among the local deadlines of the predecessor sub-tasks.

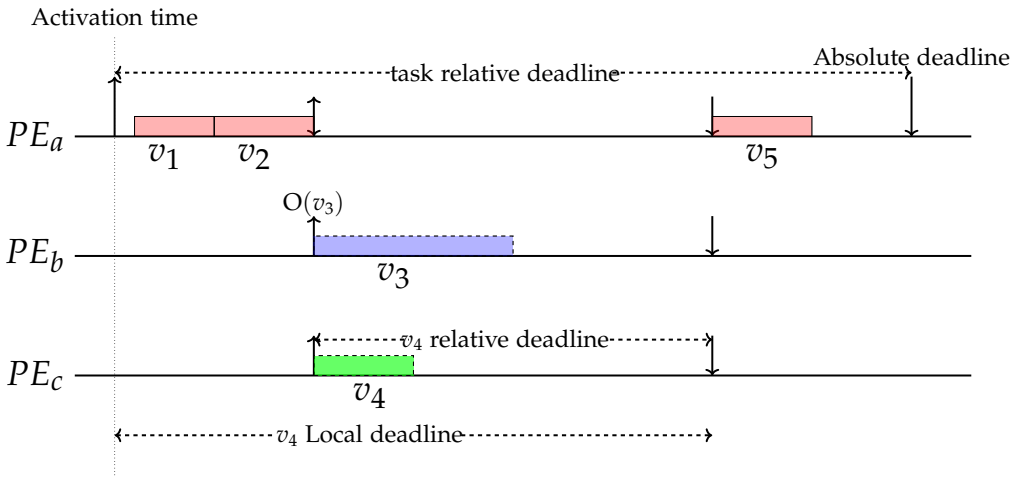


Figure 5.2.: Example of offset and local deadline

Figure 5.2 illustrates the relationship between the activation times, the intermediate offsets, relative deadlines and local deadlines of the sub-tasks of the task depicted in Figure 5.1. We assume that v_1, v_2, v_5 have been allocated on the same PE whereas v_3 and v_4 each on a different engine. The activation time is the absolute time of the arrival of the sub-task instance. The activation time of a source sub-task corresponds to the activation time of the task graph. The offset is the interval between the activation of the task graph and the activation of the sub-task. The local deadline is the interval between the task graph activation and the sub-task absolute deadline.

Definition 5.4.1. Sub-task $v \in \mathcal{V}_\tau$ is feasible if for each task instance arrived at a_j, sub-task v executes within the interval bounded by its arrival time $a(v) = a_j + \mathsf{O}(v)$ and its *absolute* deadline $a(v) + \mathsf{D}(v)$.

Lemma 1. *A task is feasible if all its sub-tasks are feasible.*

Proof. By the definition, the local deadline of the sink sub-tasks is equal to the deadline of the task D. Moreover, the offset of a sub-task is never before the local

deadline of a preceding sub-task. Therefore 1) the precedence constraints are respected, 2) if sink sub-tasks are feasible, then the task is feasible. □

Definition 5.4.2. Let p be a processor and v be a sub-task of task τ.

p is an illegible processor for v if :

$\forall \pi \in \Pi(\tau)$ such that $v \in \pi \implies \exists VC \in p$ that can be allocated to v and verifying condition $SI(\pi, D) \geq 0$

Theorem 1. *Let p be a processor and v be a sub-task.*

if p is not an illegible processor to v, than v can not be feasibility allocated to p.

Proof. The proof is done by counter example. Let assume that p is not illegible to v and that the system is schedulable.

By negating Definition 5.4.2, it exists at least one path where $SI(\pi, D) < 0$. Therefore, one or more sub-tasks will have an execution time greater to their deadlines, thus missing their deadlines and Lemma 1 can not be satisfied. □

The slack computation allows us to ensure that all communications will be achieved will not push a sub-task miss its deadline as they have their own reserved time that is not included in the distributed slack.

5.4.4. Single core schedulability analysis

Schedulability for FP can be checked using the test proposed in N. C. Audsley, 1991. The latter has a high complexity, therefore we allow also using the test proposed in Bate and Alan Burns, 1997 which is less complex but pessimistic. EDF schedulability can be checked using workload requirement using the schedulability test proposed in Sanjoy K. Baruah, Louis E. Rosier, and Rodney R. Howell, 1990b. This test has been extended for tasks presenting offsets, as follows :

$$dbf(\tau, t) = \max_{v \in \tau} \sum_{v' \in \tau} \lfloor \frac{t - \Theta(v') - D(v') + T(\tau)}{T(\tau)} \rfloor \tag{5.4}$$

where:

$$\Theta(v') = (O(v') - O(v)) \ mod \ T(\tau)$$

Thus, our approach has converted the task and communication allocation problem to a single core analysis problem.

5.5. Results and discussions

In this section, we evaluate the performances of our proposals on a large set of synthetic experiments. The taskset generation starts by invoking UUnifast algorithm to generate the utilization rate of n sub-tasks. Hence, DAG and inter-sub-tasks communications are generated by TGFF R. P. Dick, D. L. Rhodes, and W. Wolf, 1998. In the architecture side, we map these tasks on a 3×3 2D-Mesh NoC with routers, containing for each of them 6 VC with the following TDMA slot configuration : $[4, 2, 3, 5, 3, 3]$. To avoid untractable hyper-periods, the period of every task are randomly generated from a list of values between the interval of 1000 and 10000. Communications workload are flit-based quantified, i.e; for each communication, we assign a random number of flit in the range of 3 and 40.

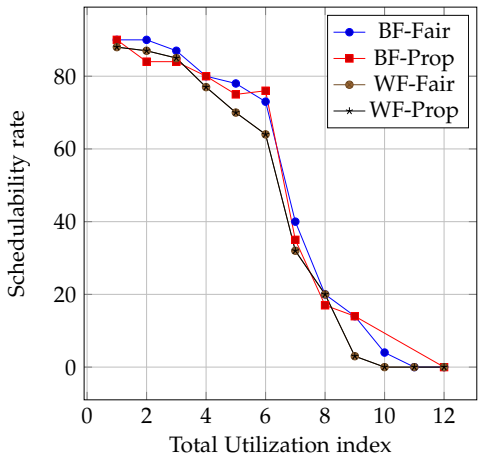


Figure 5.3.: Heuristics schedulability rate by tasks utilization rate

The code has been executed on a regular laptop with Intel Core i5-7200U processor (2×2.5 GHz) and 8 GB of ram. The results are shown in Figure 5.3 and are presented as follows. Each experience takes a heuristic allocation and artificial-deadline assignment method as detailed in 5.4.3. The results are reported by a function measuring the evolution of the schedulability rate by varying the task utilization rate. Thus, we notice BF heuristic combinations dominate the WF heuristic. This can be explained by observing that BF tries to pack the maximum number of sub-tasks into the minimum of engines, and this allows more flexibility to schedule heavy tasks on other engines.

5.6. Conclusion

In this chapter, we provide real-time tasks support into NoC-based multiprocessor. Our approach converts an extremely complex task and communication allocation problems to a set of classical scheduling problem, for which efficient algorithms exist, while preserving allocation problem timing properties. We used bin-packing

heuristics to allocate tasks on cores and we provided also promising methods for offset and deadline assignment.

In the next chapter, we extend the model to include the communication with off-chip memories. Furthermore, we deploy sophisticated technique to find an optimal task allocation.

6. Processor-Memory co-scheduling on NoC

Contents

6.1. Introduction

The DAG model is suitable to describe communicating tasks. In this chapter, we aim at presenting a more expressive model as we would like to include two types of communication: task-to-task and task-to-main-memory communication. Indeed, we introduce a DRAM (Dynamic Random Access Memory) as a part of our architecture model while extending the task model by the AER model Durrieu et al., 2014.

The latter models the execution of a sub-task into three distinct and separated phases: in a first phase (*Acquisition*) the sub-task reads the needed data from main memory into a local memory; in a second phase (*Execution*) the sub-task executes using data only present in the local memory; in the third phase (*Restitution*) the sub-task writes the produced data back to the main memory.

Thus, three questions need to be addressed to provide real-time support to NoC-based platform :

- (i) Where a given sub-task might execute?
- (ii) How to ensure that inter-task communication are bounded?
- (iii) how to ensure that main-memory data read and write-back are correctly achieved?

while meeting real-time constraints. More explicitly, it is necessary to define the core where a given task might execute, which path shall a communication take within a network and how memory controllers are designed, to follow the system design constraints. The system design space is exponential to the number of tasks, cores, communication paths, and the number of memory controllers. Exploring all allocation possibilities to find the best solution is an extremely time consuming operation. Indeed, smaller instances of such problems are proven to be NP-complete in strong the sense.

In this chapter, we define a methodology to tackle this problem using a *fast* implementation of a multi-objective implementation of simulated-annealing meta-heuristic. We aim to "direct" the metaheuristic space exploration, to explore potentially good solutions by the mean of NoC-resources clustering. We consider task to core allocation, task-to-main-memory and inter task communications to the different communication medias. We consider partitioned scheduling; i.e. once a sub-task and communication is allocated on a resource, all activation of the same task and communication will be allocated to the same resource during the system lifetime.

Further, we prove the efficiency of the meta-heuristic by a large set of synthetic simulations, showing a significant improvement to the state of art.

The rest of this chapter is organized as follows. We present in Section 6.2 prior contributions related to the current work. Section 6.3 details the architecture and the AER task model. We introduce the task mapping algorithm in Section 6.4. Section 6.5 relates how the schedulability analysis of the system is performed.

Experiments and results are presented in Section 6.6. Finally, we draw a conclusion in Section 6.7.

6.2. Related work

6.2.1. Real-time tasks allocation

Network-on-chip interconnection paradigm has been introduced by the seminal paper of Benini et al. Benini and De Micheli, 2002. As the task allocation problem is a high combinatorial problem, many studies are reported on the literature providing techniques to tackle it efficiently with a less computational complexity. Thus, concerning the non-real time applications, the authors in Hansson, Goossens, and Rădulescu, 2005, Sahu, Shah, et al., 2013, Srinivasan and Chatha, 2005 have proposed offline (static) as well as on-line (on-the-fly) mapping strategies for both tasks and communications.

On studies related to bus-based multiprocessors addressed to the real-time systems, the on-chip communication latency is analyzed and included as a part of the worst case execution time of a task and it doesn't much depend on the allocation schema. Obviously, such analysis is not tailored to NoC-based architecture since the communication depends drastically on the task allocation, and therefore, it must be considered independently from the task worst-case execution time. Several techniques have been proposed to allocate real-time tasks onto multicore architecture H. Zahaf et al., 2019, ZAHAF, Lipari, Niar, et al., 2020, Fonseca et al., 2015, Bhuiyan et al., 2018, as well as communications on NoC. The NoCs can be classified following two categories according to their strategy on network traffic handling. Indeed, congestion may occur when two communication flows would like to reserve a path through the network. The first category concerns the use of *TDMA* (Time-division multiple access) to share the communication medium among communication flows by time quantum reservation, whereas the second category consists of assigning priority to flows, and therefore, the arbitration follows the priority map Zheng Shi and Alan Burns, 2010. A comparative study of both strategies is reported in Benchehida, Benhaoua, H.-E. Zahaf, et al., 2020 through the simulation and analysis of several scenarios. Moreover, Harde et al., 2018, Nikolic, Hofmann, and Ernst, 2019 proposed heuristics to find the optimal TDMA quantum assignment for a minimum communication latency. An exhaustive survey addressed to real-time support on NoC is reported in Hesham et al., 2017.

6.2.2. Off-chip memory sharing

Regarding the task-to-main-memory communications, many studies have included an off-chip memory, such as DRAM with the NoC model. In such way, Jang and Pan, 2011 proposed a design of a DRAM-sensitive NoC router consisting

of a strategy to handle the communications to the main memory since the packets issued at this purpose have a high priority and are routed first. On the other hand, the authors in Daneshtalab et al., 2010 proposed a mechanism to handle multiple DRAM coupled to a NoC. Thus, they proposed a solution to sort an out-of-order arrival requests from the NoC to different main memories by an algorithm that re-orders those requests and routes them to the specified DRAM. However, all those previous work consider a NoC with 2D-Mesh topology. Likewise, Jin et al. Jin et al., 2011 proposed a mapping technique in a hierarchical tree-based NoC where bridges are deployed to connect directly the routers to the DRAM controller. Thus, they proposed a task allocation algorithm to avoid the congestion since only one task has the access to the off-chip memory through one bridge and therefore, the tasks are sequentially re-ordered.

Nonetheless, the previous studies do not consider applications with real-time constraints. In the real-time off-chip memory-aware field, Giannopoulou et al. Giannopoulou et al., 2016 have proposed a Simulated Annealing-based mapping technique for mixed-criticality tasks on Karlay MPPA NoC with a DRAM memory. Moreover, Gomony et al. Gomony, Akesson, and Goossens, 2014 proposed a middleware to adapt any TDMA-NoC with a main-memory to support real-time systems. In fact, they proposed an adaptation of the main-memory controller by computing a network interface (NI) bandwidth and cores operational frequency to optimize the NoC energy consumption. Besides that, Perret et al. Perret et al., 2016b proposed a mapping technique using constraint programming while the budgeting of real-time applications is calculated *a priori*, i.e they evaluate the computing power need as well as the number of memory access of an application.

6.3. System Model

6.3.1. Network-on-Chip design

We consider an architecture compound of N tiles. A tile $\mathcal{A}$ is composed of : (i) a processing engine PE, (ii) a router R and network interface NI. Typically, the calculation tasks that execute on the processing engines are allowed to send/receive data via the NoC by means of their adjacent routers. The latter are wired between them by a set of unidirectional links. In this work, tiles are arranged in a square matrix of row and column count equal to m, i.e $\mathsf{N} = m^2$. Therefore, every tile is denoted by its x and y position within the square matrix, i.e. tile $\mathcal{A}(x, y)$ (respectively its router $\mathcal{R}(x, y)$ and processing engines $\mathsf{PE}(x, y)$) is located at row x and column y, in a 2D mesh topology, as demonstrated in Figure 6.1. Each tile is directly wired to its four neighbors located on its: North, South, East, West side, except those located at the edge and the corner of the topology which have three and two neighbors respectively. Moreover, a group of tiles can be gathered to constitute a so-called *virtual cluster*.

Router

The router remains the main in-tile component responsible for handling the network traffic coming from other routers and passing through it, as well as the traffic-load coming from its processing engine. Additionally, edge routers may have a direct connection with I/O components in order to communicate with external platform elements. A router is composed of the following elements:

- *Input ports*: they are the router entries. Each port is assigned a set of *Virtual Channels*. A virtual channel VC is a buffer that stores temporarily a message during its traversal, when the requested output is not immediately available. In a typical design, the number of virtual channels per port is the same for all input ports for any router, and the number of input ports is equal to the number of neighbors, to which we add the input port ensuring the connectivity with the local PE of the same tile.
- *Output ports*: they are the exit point of the messages from the router. A given communication flow takes one of router outputs' to move forward, according to a *routing* policy. When more than an entry desire the same out port, an *arbitration* policy is required to elect the communication that will be served. The number of output ports is equal to the number of input ports.
- *Crossbar switch*: it is responsible for physically connecting the input ports to the output ports.

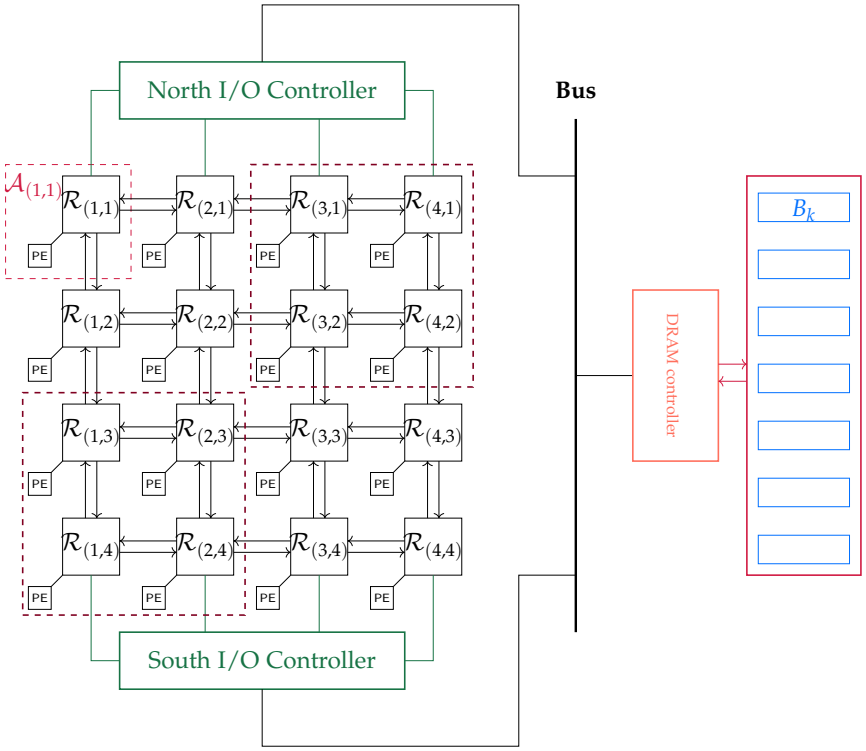


Figure 6.1.: The execution platform architecture

The performances of a NoC design highly depend on *routing and arbitration policies*, and the *flow control protocol*.

Routing policy: We use DOR-XY[1] routing policy. A message is forwarded from a router to another continuously on the X-axis until it reaches the destination router column. Afterward, the communication follows the Y-axis until its destination. This routing protocol is deterministic, i.e. a given communication will always follow the same path at runtime.

Flow control protocol: It defines how a communication moves from a router to its neighbor. We consider the *wormhole switching* control flow protocol. A message $\mathcal{M}$ is divided into multiple packets P. The different packets are scheduled on the network resources independently. Each packet is sliced to a set of transferable units, called Flits, i.e. $P = \{F_h, F_1, F_2, \cdots, F_n, F_t\}$, where F_h and F_t denote special flits called header and tail. During its traversal, a packet locks the virtual channel it uses, i.e it has an exclusive access. The *header* first moves to the next router, gets a free virtual channel and locks the VC to its packet. When a *tail* flit leaves the VC, it releases the VC, therefore the latter becomes available to another communication.

Arbitration policy The arbitration policy defines how a router should handle the communication flows conflict. Indeed, during their traversal, the messages may would like to reserve the same output port, when this resource is assumed for a message at time. Thus, the arbiter decides which among message is allowed to use it. We denote several policies such as priority-based or shared in time policy. In this contribution, we use the latter policy, also named TDMA (Time Division Multiple Access) when each VC has a fixed number of slots to exclusively take the output port.

Processing engine

The processing element is the unit responsible for achievinqg computations. Each processing engine PE has its own registers and a *scratchpad memory* (abbreviated by SMP) which serves as its local memory for data storage coming from on-chip or off-chip communications or local processing.

6.3.2. DRAM organization

The DRAM is platform main memory. It is considered as an off-chip memory since it is located outside of NoC. Therefore, The NoC is connected to the DRAM by means of a dedicated bus. The DRAM controller is a unit responsible to organize and handle the requests coming from other architecture elements. At this fact, DRAM and its controller forms the *memory sub-system*. We illustrate in Figure 6.1 the platform. Typically, during the system execution, the processing

[1] Direct Ordering Routing

engine may request data from the DRAM and load them into its local memory, which emits a signal to the nearest *I/O cluster* to achieve it. We describe further such operations.

The DRAM architecture is composed of several *banks*. The bank represents a two-dimensional matrix of *cells* arranged in *rows* and *columns*. Each cell holds a bit. Additionally, several banks can form a virtual entity called *rank*. As a result, the DRAM has a tree-dimensional structure where each cell is identified by a 3-tuple coordinate <bank, row, column>. Such organization is depicted in Figure 6.2. We describe in the following the steps of read and write operations that occur between the NoC and the DRAM.

Read operation: It is performed in demand of a processing engine $\mathsf{PE}(x, y)$ as follows:

1. $\mathcal{A}(x, y)$ initiates the read operation parameters, and sends the request to its network interface NI.
2. $\mathsf{R}(x, y)$ is asked from its network interface NI to forward the request to an I/O cluster to achieve this memory operation, let it be $A^m(x', y')$.
3. The request is routed between $\mathsf{R}(x, y)$ and $\mathcal{A}^m(x', y')$ using the mechanisms described above.
4. The I/O cluster sends the request to main controller through the dedicated bus.
5. DRAM controller computes the data location and sends back data to $\mathcal{A}^m(x', y')$.
6. Data are routed back from $\mathcal{A}^m(x', y')$ to $\mathcal{R}(x, y)$ using the routing protocol.

Write operation: The write operation is very similar to the read operation. Instead of sending request and wait for the reply, the request and the data are sent sequentially between $\mathcal{R}(x, y)$ and $\mathcal{A}^m(x', y')$.

As we would like to make the connection between NoC and DRAM as much predictable as possible, we associate the access of a given bank B_i exclusively from a prior known router $\mathcal{R}(x, y)$. The purpose behind such mechanism is to provide a unique path from a PE requester to a bank B_i. Moreover, we aim at sharing fairly the off-chip communications among routers throughout these associations of $< \mathcal{R}(x, y), B_i >$.

In a nutshell, we can describe the controller as a supervisor which satisfy external requests by applying commands on the DRAM.

- **Row Activate (ACT)** It is the first operation applied on a row, which basically moves the row on a dedicated region called *row buffer* in order to fetch it. After the operation, the row is denoted as *opened*.
- **Read (RD):** It reads a column from an opened row and retrieves the bit stored in the cell.
- **Write (WR):** It writes on a column that belongs to an opened row.

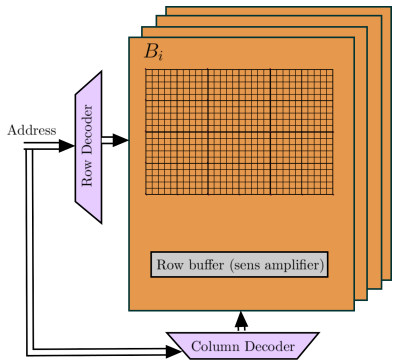


Figure 6.2.: DRAM architecture

- **Precharge (PRE):** It closes an open row. This operation moves the row from the row buffer to the bank. It involves the end of the read/write operation on the row.

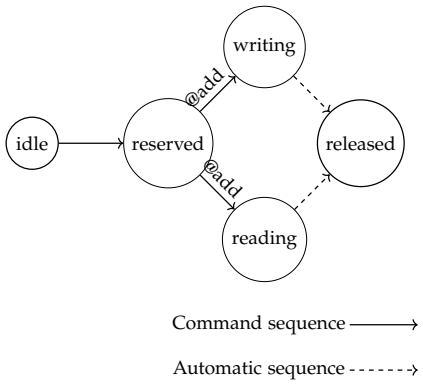


Figure 6.3.: Graph state of a bank

Thus, during the system execution, a bank goes through several states, as depicted by the graph in Figure 6.3.

Memory Controller Model

We integrate the DRAM controller model RLDRAM (Reduced Latency DRAM Controller) proposed by Hassan Hassan, 2018 as a suitable solution to real-time systems. Compared to the classical DDR-DRAM it experiences less latency and provides predictability, mainly due to:

- the activation and prechage phases are relegated to the RLDRAM. Consequently, the controller orders only the Read/Write commands;
- the data bus of DRAM is bidirectional and the entire bandwidth is used to either read or write operation, and therefore, switching delay between read/write impacts the latency. However, the delay of RLDRAM is less important than DDRx-DRAM;
- all data addresses are provided in one-step, with the non-multiplexed mode

Further details are provided in Casini et al., 2020.

The previous features allows to determine with high simplicity the effective cost related to DRAM access and data copy. Besides that, we assume a *bank partitioning* onto the DRAM, which strictly assign a bank exclusively to a specific PE, therefore, reducing interference among PEs.

Example 6.3.1. Let consider PE_3 assigned to bank B_5 triggers a read request. After arriving to the controller, the RLDC decodes the request by the two block in order to determine the request type (in our case, read request) and which cells to get data through the array of (bank, row, column). Afterward, the request is placed by its arrival order into its right per PE buffer in order to be elected by the arbiter. We have to keep in mind that the timing checker behaves like a request regulator which ensure a minimum between between two commands toward a bank. When a request is elected, it is immediately applied into the RLDRAM.

Further details about the data latency calculation are provided in Section 6.5.

6.3.3. AER DAG task model

We describe the system by a task set Γ of sporadic DAG tasks, where a task τ is composed by the tuple: $\tau_i = \{\mathcal{V}_i, \mathcal{E}_i, \mathsf{D}_i, \mathsf{T}_i\}$. We outline each element of a task as follows:

- $\mathcal{V}_i$: A set of sub-tasks v_j which represent a computational part of a task, such as an parallel chuck of code
- $\mathcal{E}_i$: A set of edges $e(v_j, v_k)$ which models a directed communication between 2 sub-tasks.
- D_i: Is the relative end-to-end deadline of a task. The latter should finish its execution no more than this timing constraint.
- T_i: It represents the minimum inter-arrival time of the task instance.
- P_i: The priority of the task. Obviously, the most a task is priority, the most it is privileged compared to the lower priority tasks. Moreover, the sub-tasks inherit the priority of the task that composes them.

A sub-task $v_j \in \tau_i$ presents the following characteristics:

- C_i: represents the worst-case execution time of the sub-task on the platform.
- O_i: determines the task offset, i.e: a late starting time of a task due to a non-ready data (e.g: off-chip data copy operation).
- J_i: refers to maximum delay that a sub-task can suffer after its activation due for example the arrival for an external event.
- r_i: its the worst-case response time of a sub-task.
- $\pi(v_j)$: represents the processing engine's index on which the sub-task v_j is allocated.

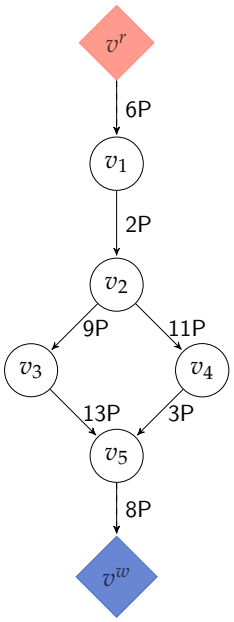


Figure 6.4.: The AER task model

We highlight in Figure 6.5 the different parameters of a sub-task (previously described in Section 6.3.3).

The Directed Acyclic Graph (DAG) is an expressive model that represents a task composed of sub-tasks (vertices) and their communications (edges). Thus, a sub-task v_j of a task τ_i represents a chuck of code that performs calculation. It can starts its execution immediately after the task release, if there is no precedence constraint on it, e.i: it has a predecessor sub-task that has not finished its execution. Consequently, each sub-task has a set of direct successors $succ(v_j)$ of sub-tasks in which there exists a relation, defined by :

$$\{v_k \in succ(v_j) | e(v_j, v_k) \in \mathcal{E}_i\}$$

Similarly, a sub-task has a set of direct predecessors $pred(v_j)$, defined as follows:

$$\{v_k \in pred(v_j) | e(v_k, v_j) \in \mathcal{E}_i\}$$

We call a sub-task that has no predecessors $pred(v_j) = \emptyset$ a root sub-task, whereas we call a sub-task with no successors $succ(v_j) = \emptyset$ a leaf sub-task. Moreover, two sub-tasks that belong to the same task can be executed in parallel (simultaneously) if there is no precedence constraint between them by allocating them on different tiles.

In this work, we aim at include explicitly the off-memory access in the task model. Following that, we extend the DAG task model as proposed by Baruah et al. Sanjoy Baruah et al., 2012 by the AER model which include special sub-tasks, called virtual sub-tasks. Such sub-tasks constitute, more precisely, an off-chip memory copy engine mechanism. In the AER model, each task has one virtual

sub-tasks at the beginning of the task to copy data from the DRAM to the local core memory (SPM) and one virtual sub-tasks at the end of the task to write-back data to the main-memory.

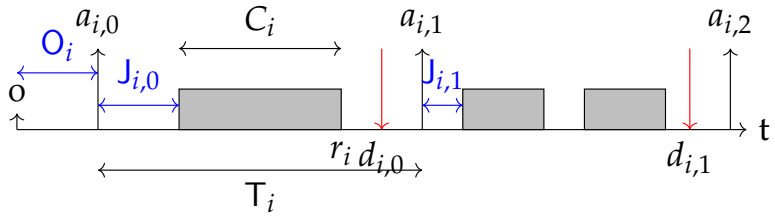


Figure 6.5.: The sub-task parameters

Example 6.3.2. Let consider $\tau = \{\mathcal{V}_i, \mathcal{E}_i, D = 120, T = 120\}$ a DAG task, depicted in Figure 6.4.

When the task is activated, it starts by the read memory sub-task v^r to perform a data copy from the main-memory. v_1 begin its execution directly after the data has been copied entirely (6 packets) to the local memory of its PE. After v_1 ends its execution, it sends data to v_2 through the network since v_2 must wait the end of communication. v_3 and v_4 are allowed to run in parallel as soon as they receive the packets (9 and 11 packets respectively). As v_5 has two predecessors, it must waits until both of its predecessors terminate their execution. Finally, the sub-task v^w achieves the write-back operation. The latter must finish no later than the task deadline 120 time units from the task activation.

6.4. MO-SA DAG Task Allocation

Typically, meta-heuristics are suitable when the problem is highly combinatorial. Compared to exact optimization methods, they don't converge to the optimum, but they provide solutions closer to the optimum with relatively less time.

In this section, we introduce the multi-objective simulating annealing (MO-SA) Bandyopadhyay et al., 2008. This method is derived from the metal annealing procedure in which the temperature heating and cooling of a material is controlled to increase its strength. This derived model is appreciated for its ability to escape from a local solution that could be considered as optimal on its scope. To perform such, this heuristic makes possible the control of the following annealing parameters:

- The initial temperature (t_{init}): determines the highest temperature at which a metal is first heated.
- The minimal temperature (t_{min}): refers to the lowest temperature the metal can reach. Arriving at this value, the annealing process is stopped.
- The current temperature (t_{cur}): refers to the temperature of the current metal state.

- The lowering coefficient (t_α): defines how the temperature is decreased from a current temperature to a more lower one. The state is updated by: $t_{cur} \leftarrow t_{cur} \times t_\alpha$.

Before going further, we define the concept of a move: the action of shifting on the solution space from a position to another. Concretely, a move is associated to a solution, and can be defined as *bad* if the solution provided is less-optimal than the previous move. Likewise, we define a *good* move if it provides an improvement since the previous move solution.

The annealing temperature mission is twofold: determines the depth of the solution space exploration and the threshold tolerance of the explored solutions. More explicitly, the temperature is used to determine whether a solution is acceptable or not. In fact, during the solution space exploration, a bad move doesn't intrinsically involve a wrong exploration path, but it can be accepted according to the current temperature value. Thanks to this mechanism, the algorithm explores the space extensively, seeking a *global* solution without falling into a *local minima*.

The more the cooling temperature is high, the more the solution space is explored exhaustively. This is explained by the fact that the acceptance ratio of a bad move is high when the current temperature is high. Likewise, a bad move has less chance to be accepted when the current temperature is low. Thus, we define in Equation 6.1 how a bad solution is conditionally either accepted or refused. The acceptance is determined by comparing a random generated value ($random\_float()$) against a calculated number that takes as parameters: the gap between the previous and current solution and the current temperature.

Definition 6.4.1. A bad neighboring solution S is accepted or not according the following function:

$$ACC(S) = \begin{cases} true, & e^{\frac{\Delta E}{t_{cur}}} > random\_float() \\ false, & \text{otherwise} \end{cases} \tag{6.1}$$

Algorithm 3 describes our heuristic to find an optimal task mapping. We have to keep in mind that we model a multi-objective problem as we consider each DAG response time as a single objective. At the end, the algorithm returns a set of solutions $\{S_0, \cdots, S_n\}$ – We assume that the set contains only incomparable solutions. For instance, S_i and S_j are incomparable if they provide for τ_1: $r_{i,1}$ and $r_{j,1}$, and $r_{i,2}$ and $r_{j,2}$ for τ_2 respectively, when: $r_{i,1} > r_{j,1}$ and $r_{i,2} < r_{j,2}$.

Consequently, the algorithm returns a *front pareto* that includes all accepted solutions, and it remains to the responsibility of the designer to choose which among the proposed solution is most suited to the specification needs.

Algorithm 3 describes the MO-SA. We can roughly divide the algorithm into three steps: (i) find a feasible (schedulable) initial solution, (ii) explore neighbor solutions by moving, (iii) decide if the solution is acceptable or not.

Algorithm 3 Multi-objective Simulated Annealing

1: **input:** Γ: Tasket, t_{cur}: initial temp, t_{min}: minimal temp, t_{low}: lowering coefficient
2: **output:** Ω: Pareto Frontier
3: $\Omega \leftarrow \emptyset$
4: $S_O \leftarrow generate\_mapping(\Gamma)$
5: **while** $t_{cur} \geq t_{min}$ **do**
6: $\quad S_N \leftarrow next\_neighborhood(\Gamma)$
7: $\quad \Delta\ E \leftarrow diff(S_N, S_O)$
8: $\quad$ **if** $\Delta\ E < 0$ **then** ▷ Good move
9: $\quad\quad S_O \leftarrow S_N$
10: $\quad\quad update\_pareto(\Omega, S_N)$
11: $\quad$ **else** ▷ Bad move
12: $\quad\quad$ **if** $e^{\frac{\Delta\ E}{t_{cur}}} > random\_float()$ **then**
13: $\quad\quad\quad S_O \leftarrow S_N$
14: $\quad\quad\quad update\_pareto(\Omega, S_N)$
15: $\quad\quad$ **end if**
16: $\quad$ **end if**
17: $\quad t_{cur} \leftarrow t_{cur} * t_{low}$
18: **end while**
return Ω

The algorithm starts by initializing the *front pareto* set and generate a random initial solution mapping (Line 3-4). This operation is repeated until a given mapping is determined as schedulable. Afterward, a block is repeated while the current temperature t_{cur} is higher than the minimum temperature t_{min} (Line 5). Then, it starts to explore the next near solution (Line 6) through a function described by Algorithm 4. After the next neighbor move has been done, we evaluate the effectiveness of the move by comparing it to previous solution by calculating the tardiness (Equation (6.2)). We describe the tardiness of a task by the amount of time it takes by a task to complete its execution after exceeding its deadline. Nonetheless, the tardiness is equal to zero if the task respects its deadline ($r_i \leq \mathsf{D}_i$). At this moment, the algorithm has to take a decision of either accepting or refusing this after-move mapping schema. If the latter is considered as a bad move (Line 11), it will be evaluated (Line 12) following the acceptance function (Equation (6.1)). After, the current temperature is decreased by the lowering coefficient (Line 17). At the end, the algorithm returns the front pareto which contains incomparable solutions. We provides in Figure 6.6 an example of a front pareto set.

$$tardiness(\tau_i) = \max\{0, r_i - \mathsf{D}_i\} \tag{6.2}$$

```
S1: {
  Task_1: 79, Task_2: 182, Task_3: 67, Task_4: 121, Task_5: 70
}
S2: {
  Task_1: 127, Task_2: 88, Task_3: 82, Task_4: 97, Task_5: 55
}
S3: {
  Task_1: 127, Task_2: 88, Task_3: 82, Task_4: 97, Task_5: 55
}
S4: {
  Task_1: 127, Task_2: 88, Task_3: 82, Task_4: 97, Task_5: 55
}
```

Figure 6.6.: An example of front pareto of incomparable Solutions

6.4.1. Exploring the neighbor solutions

As the solution space is large, it requires a strategy to explore it efficiently. We consider in Algorithm 4 that a move from current mapping to another is firstly performed by reallocating a random sub-task (Line 4-6). Afterward, we test if the sub-task's communication is placed on a VC which has the highest slot (Line 7). If true, the sub-task is reallocated to another core and its communication takes a random VC (Line 8-9). Otherwise, we proceed to reallocate only the communication to another VC on the same core (Line 11). This action is repeated until finding a schedulable mapping (Line 6-13). Finally, we update the mapping schema according to the changes and return it (Line 13-14).

Algorithm 4 Next Neighborhood exploration

1: **input:** Γ: Taskset
2: **output:** N_{map}: New allocation schema
3: $N_{map} \leftarrow allocation(\Gamma)$
4: $\tau_i \leftarrow get\_random\_task(\Gamma)$
5: **repeat**
6: $(v_j, e_j) \leftarrow get\_random\_subtask(\tau_i)$
7: **if** $highest\_slot(VC(e_j)$ **then**
8: $subt\_to\_core\_alloc(v_j)$
9: $random\_reservation(e_j)$
10: **else**
11: $reallocation(e_j, VC(e_j))$
12: **end if**
13: **until** $is\_schedulable(\tau)$
14: $N_{map} \leftarrow update\_allocation(\tau)$
return N_{map}

6.5. AER tasks scheduling analysis and memory latency

This section explains how the scheduling analysis of the task set is performed. In fact, the system schedulability analysis can be distinguished by three parts: (i) the on-chip communication analysis, (ii) the off-chip memory copy analysis, and (iii) the analysis of the task response time on the processing engines. Firstly, we detail these three parts independently from each other for the sake of clarity, and we give at the end a holistic analysis of the entire system.

6.5.1. Virtual sub-tasks and memory latency cost

We determine the worst-case memory access delay (abbreviated by WCMD) of a bank's DRAM by Equation (6.3). This cost of packet copy concerns exclusively the delay between the I/O cluster to the RLDRAM regardless the cost due to the on-chip routing.

For the sake of simplicity, we assume the late arrival of each request into the buffer which takes the (N)th position, and therefore, it will be elected after by the Round-Robin scheduler has satisfied the (N-1) requests that precede it. We also assume any delay on the bus that relies the NoC with the DRAM controller. Table 6.1 resumes the RLDRAM parameters delay.

$$WCMD_i(\mathsf{P}) = (N-1) \times tRC + tCL \tag{6.3}$$

Parameter	Description	Cycle cost
tRC	The minimum delay between two commands	6
tWL	The minimum delay to start copy after W	14
tRL	The minimum delay to start copy after R	13

Table 6.1.: RLDRAM memory access costs

6.5.2. On-chip communication latency analysis

We describe the interference between two or more communication flows when they would like to reserve the same output port. We aim at deploying a conflict-free scheme for routers since no arbitration unit is required on the architecture. Instead, the router follows the TDMA table to share the output medium. Therefore, we calculate the latency due to an on-chip communication $e_i(v_{src}, v_{dst})$ between two sub-tasks in a given VC_j as following:

$$\mathsf{lat}(VC_j, e_i) = \frac{TF(\mathcal{M}_i)}{n_{slot}} \cdot \frac{\Delta}{\eta_j} + H_i \tag{6.4}$$

Where:

- $TF(\mathcal{M}_i)$: total number of Flits contained in the message;
- n_{slot}: the amount of data send by 1 slot of communication. We suppose by default 1 Flit;
- Δ/η_j: refers to the bandwidth assigned to a VC, which represents the ratio of: the total number of slots in a *TDMA cycle* / number of slots assigned to a VC;
- H_i: distance between source and distance tile in which the two sub-tasks are allocated. It is expressed by number of hop that separate the two tiles.

By this, we assume that the VC will be served at the end of the TDMA cycle.

6.5.3. Preemption-cost and AER tasks response time

The scheduling analysis of the task set is based on the task response time, while we consider a given task schedulable when the following inequation remains true: $r_i \leq \mathsf{D}_i$. The calculation of task response time takes into account two aspects:

- the interference (also called contribution) from higher priority tasks occurred only in the sub-task jobs $\mathcal{J}_p \in \{p_0 \cdots p_n\}$ instantiated on its busy period w.
- Considering the tasks with *dynamic offsets*. As the sub-tasks may delay their activation until the arrival of the message, which depends on its time release and the response time of the previous sub-task (precedence constraint).

We assume in the task model an implicit deadline distribution ($\mathsf{T}_i = \mathsf{D}_i$) with Rate Monotonic (RM) priority assignment. We base ourselves on the schedulability test proposed by Palencia et al. Palencia and Gonzalez Harbour, 1998. However, we adapt it for DAG tasks, while the initial contribution is addressed only for tasks modeled as transactions.

Worst-case response time of the DAG task

Initially, we have to keep in mind that a given sub-task $v_{ab} \in \tau_i$, during its execution can be preempted by a task τ_k with a high priority, which generates a delay imposed to the preempted sub-task v_{ab}. However, it's difficult to predict which among sub-tasks of τ_k will preempt v_{ab}, and therefore, will contribute to its worst-case response time (WCRT). More specifically, we must find the sub-task of τ_k that creates the worst-case busy period of v_{ab}; The busy period w of a task denotes the time interval starting from the critical instant until the response time Y. Wang and Saksena, 1999, where the critical instant itself is defined as the arrival time that produces the largest response time. Simply, it coincides with the activation of sub-task a $v_{ik} \in \tau_k$:

$$W_{ik}(v_{ab}, t) = \sum_{j \in hp(v_{ab})} \left(\left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{t - \varphi_{ijk}}{T_i} \right\rceil \right) C_{ij} \tag{6.5}$$

Where, φ_{ijk} determines the sub-task phase of v_{ab} aligned with the arrival time of v_{ik}, such that its critical instant is initiated with the activation of v_{ik} :

$$\varphi_{ijk} = (T_i - (O_{ik} + J_{ik} - O_{ij})) \bmod T_i \tag{6.6}$$

Thus, the calculation of the WCRT of v_{ab} goes through checking all possible variations and by taking the maximum of all possible interferences from the sub-tasks of τ_k (Equation (6.7)).

Obviously, we assume in our analysis that the task offset might exceed its period. We also consider the release time of a sub-task v_i might be occurred in the interval of $[t_0 + O_{ij}, t_0 + O_{ij} + J_{ij}]$.

$$W_i^*(v_{ab}, w) = \max_{k \in hp(v_{ab})} W_{ik}(v_{ab}, w) \tag{6.7}$$

With:

$$hp(v_{ab}) = \{v_j \in \tau_i \mid P_j \geq P_{ab} \wedge \pi(v_j) = \pi(v_{ab})\} \tag{6.8}$$

For space constraints, we don't explicitly outlined the calculation of the busy period's w_{ab} length of sub-tasks L_{ab}. More details are available in a dedicated section in Palencia and Gonzalez Harbour, 1998.

$$R_{ij} = C_{ij} + \max_{k \in hp(v_{ij})} \left[\max_{p = p_0 \cdots p_L} (R_{ij}(p)) \right] \tag{6.9}$$

Where:

$$R_{ij}(p) = w_{ijk}(p) - \varphi_{ijk} - (p-1)T_i + O_{ij} \tag{6.10}$$

6.5.4. Offsets and jitters assignment

We use for analysis purpose the Worst-case Dynamic Offset (WCDO) algorithm Palencia and Gonzalez Harbour, 1998. Since the tasks are modeled following the DAG, offsets are assigned to impose the precedence constraint between sub-tasks. Thus, we ensure the strict respect of time causality, since a sub-task cannot be activated prior to the complete reception of a message (or signal) from its predecessor.

As we consider a distributed system in which processors are linked by the NoC, the offset O_i of a given sub-task constitutes the best-case scenario of both time execution and the minimum communication latency of the sub-task that precede it, whereas, the jitter J_i signifies the maximum of both execution and communication time from the previous sub-task.

Consequently, we assume the best case execution time of a sub-task to zero $(C_i = 0)$.

We calculate the offset and jitters for tasks following the Worst-case Dynamic Offset (WCDO) algorithm. At first, the variables are initiated by: $J_{ij} = 0$ and $O_{ij} = \{k \in pred(v_{ij}) \mid R_{ik}\}$.

$$O'_{ij} = \min_{k \in pred(v_{ij})} R^l_k + \underline{e}_{k,j} \tag{6.11}$$

$$J'_{ik} = \max_{k \in pred(v_{ij})} R_k + \bar{e}_{k,j} \tag{6.12}$$

Hence, we calculate the task offset by the lower bound of the best case response time of its predecessor, whereas, the jitter is computed by the upper bound of the worst case response time of the predecessor. By using these parameters we re-calculate the response time of a sub-task for each update iteratively until we obtain the same result in two successive iterations:

$$r^{n+1}_{ik} = r^n_{ij} \qquad \forall i, \forall j \tag{6.13}$$

Where:

$$r_{ij} = R_{ij} + J_{ij} \tag{6.14}$$

6.6. Experimental Results

This section discloses the results of an experimental evaluation that compares the proposed mapping technique, the MO-SA algorithm (Section 6.4) against the bin-packing mapping heuristics. We detail in this section the implementation of the algorithm as well as the execution environment.

6.6.1. Task set generation

We use UUniFast Discard algorithm Emberson, Stafford, and Davis, 2010 to generate a range of processor utilization U_i (Equation (6.15)) since it determines the CPU workload. We use the algorithm to generate utilization sets several times meaning testing many utilization values while the maximum tolerable utilization rate in order to produce a schedulable task set is $U_i = 1$ on one processor. Each set contains the utilization of all processors on the execution platform.

As the AER task model is derived from the DAG model, we must define the composition of the tasks: (i) the number of sub-tasks, and (ii) the edges that link them.

$$U_i = \frac{C_i}{T_i} \tag{6.15}$$

The DAG structure is generated randomly by TGFF tool Robert P Dick, David L Rhodes, and Wayne Wolf, 1998 following our own generation parameters. We determine *a priori* the number of DAG tasks which is by default assigned to 5. After, we determine randomly the number of sub-task that compose a task into the range of [5,10] whereas the graph degree is randomly assigned by a value into [1, 3] and [1, 2] for the incoming and outgoing edges respectively. After creating the DAG tasks structure, we assign to each of them a period T_i picked randomly from a range [1000, 100000] by step of 1000 in order to avoid intractable hyper-periods. Each task receives an utilization factor U_i taken from the utilization set, and then, we generate a local utilization set specific for sub-task by means of UUniFast. Consequently, we calculate the worst-case execution time of the sub-tasks: $C_i = U_i \times T_i$.

Regarding the on-chip traffic workload, we consider 100% the maximum network charge when all router VCs are full when a lower value considers a less important charge of the VCs. Thus, according to desired network charge we distribute equally the communication workload to the task's edges, then, we calculate the message sizes as following: number of Flits by task = (desired communication workload $\times$ network full capacity) / number of tasks. Afterward, we distribute the flits randomly among the task's edges. We also include the first edge originating from a DRAM copy of the distribution.

6.6.2. Bin-packing heuristics

The bin-packing heuristics are useful when comparing them with other mapping methods. Thus, we compare our proposed method against: First-Fit (FF) and Worst-Fit (WF). However, as the allocation on NoC depends drastically on the distance between tasks, we adapt the heuristics to take into account the distance parameter that separates the sub-tasks.

Typically, Worst-Fit aims at maintaining all the processors at the same level of utilization, since it allocates each sub-task so that creating an equal workload for processors. Thus, the processors are sorted by their utilization by increasing order and are placed on the eligibility list. At each allocation, we try to allocate the sub-task to the closest processor to its predecessor (if obviously, is not the root sub-task) while still considering the eligible list.

Likewise, the First-Fit algorithm tries to push the maximum of sub-tasks while respecting the utilization threshold of $(U_i \leq 1)$. After jumping to the next processor, we ensure to choose the ones from the eligible list that experience minimum distance from the predecessor of the current sub-task considering creating the minimum latency.

6.6.3. Platform specifications & experiments protocol

The performance of the proposed method has been evaluated by an experimental study implying a set of synthetic experiments. We ran the simulation scenarios on a workstation that on-boards an Intel Xeon E5-2670 v3 of 12 cores clocked at 2.3 GHz each with 128 Go of RAM. All the simulations have been carried out as follows: We produced two experiences based on different simulated annealing (SA) parameters. The first experience differs from the second by the in-depth level of solution space exploration. In fact, we provided a lower minimum temperature and a higher lowering coefficient than the second experience to better contrast the importance of annealing parameters. For each SA setting, we vary the communication workload while varying the processors' utilization. We evaluate each experience by the schedulability rate calculated by the mean of 100 simulation scenarios each. The mapping algorithms have been tested on a NoC platform in non-clustered mode (also denoted by regular mode) as well as in its clustered mode (C-XX denotes experiences performed on clustered NoC). In fact, in the former configuration, the sub-tasks could be allocated in any processing engine onto the NoC, whereas the latter mode presents more constraints since tasks are assigned to a specific cluster and their sub-tasks cannot be allocated outside of their region.

The Network-on-Chip settings are detailed in Table 6.2. As we use bank partitioning, we determine randomly during the simulations the banks B_i assigned to each processing engine PE_j. Moreover, we assign tasks to clusters in the following way: we sort the tasks by decreasing order according to their utilization, then, we assign in turn a task to a cluster, until the last.

Topology	4×4 2D-Mesh
Clusters	$4 \times (2 \times 2)$
VC per InputPort	8
Buffer size per VC	10 Flits
TDM Slots distribution	[4, 6, 3, 5, 3, 3, 2, 4]
Message	8 Packets (10 Flits each)

Table 6.2.: Network-on-Chip configuration

6.6.4. Simulation results and discussions

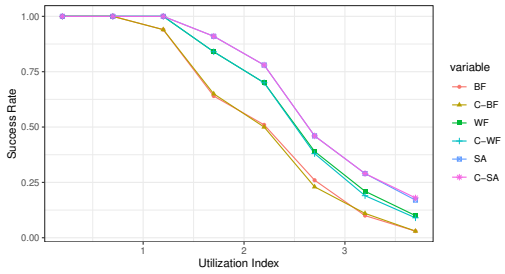
The simulation results are presented separately regarding the simulated annealing parameters. The following Figures (6.7a, 6.7b, 6.7c, 6.7d) and (6.8a, 6.8b, 6.8c, 6.8d) show the dominance of the SA algorithm against the bin-packing algorithms. Indeed, the success schedulability ratio is more important than WF and BF in both clustered and non-clustered modes. On the other hand, we notice that in high network charge, the clustered mode provides more effectiveness than the regular mode as in Figure 6.7d. This is mainly due to the fact that the clustered mode offers short distance communication when the distances could be long in regular NoCs, and therefore, generate high latencies. This observation is more supported by the figures (6.7e, 6.7f, 6.7g, 6.7h) when we compare the number of feasible solutions provided by SA in the two modes. With all network charge ratios, the C-SA outperforms the SA which highlights the fact that core clustering remains more flexible by offering a large variety of mapping schemes.

The second observation lies the SA parameters. The more simulated annealing explores deeply the solution space, the more SA and C-SA become closer as drawn is figures (6.8a, 6.8b, 6.8c, 6.8d). This is proven by the fact that clustered mode is a generalization of the non-clustered mode. Thus, when the solution space is explored by SA it may include all the portion of solutions that belong to C-SA, when eventually, both techniques will be equal. However, the strict difference remains in the number of solutions provided by them. Moreover, we can observe that SA and C-SA begin at proving less solution proportionally with the enhancement of utilization ratio.

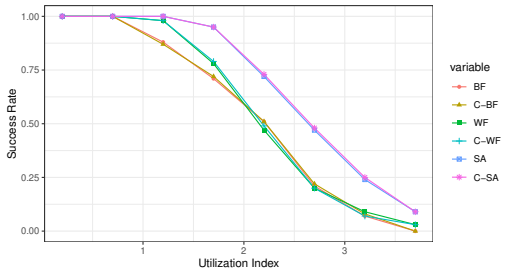
As the initial feasible solution is determined randomly, we plot in Figure 6.9 the time spent to explore the space until the stop condition of the algorithm, which is calculated by the average time of 100 simulations per utilization ratio. We observe fast delays of C-SA against SA. However, we notice a decrease in SA delays in high utilization since we stop the initial solution search at 2000 iterations, and the algorithm is aborted after trying to find an initial mapping. Therefore, it proves that C-SA is efficient at first steps.

6.7. Conclusion

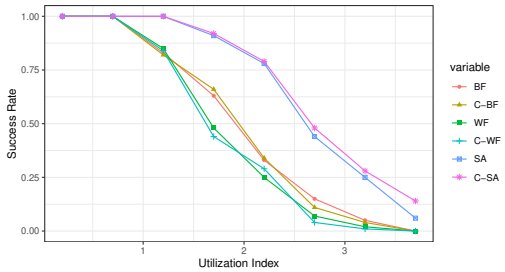
In this chapter, we presented an allocation algorithm of AER tasks, which are able to model both on-chip and off-chip communications. The complexity of the Network-on-Chip execution platform coupled with a DRAM pushed to privilege meta-heuristics technique for fast solution space exploration. We demonstrate through this chapter the effectiveness of our approaches against bin-packing heuristics. Moreover, we offer a perspective of the usefulness of core clustering in NoC platform, while typically the architecture itself is capable of including thousands of computing elements. We also privileged predictable architecture by deploying TDMA algorithm in in-router arbiters. The performance of the multi-objective simulated annealing algorithm has been proven throughout a variety of simulations.



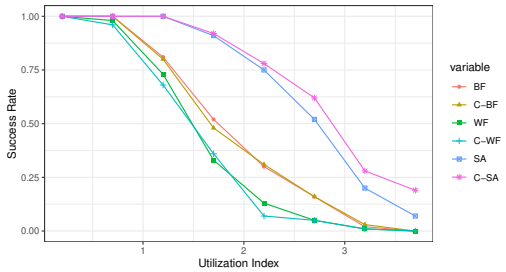
(a) 20% of comm. workload



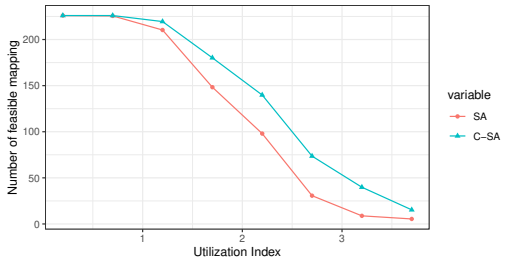
(b) 40% of comm. workload



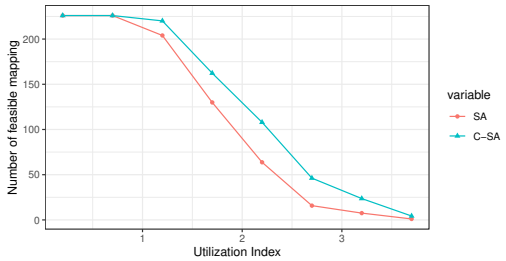
(c) 80% of comm. workload



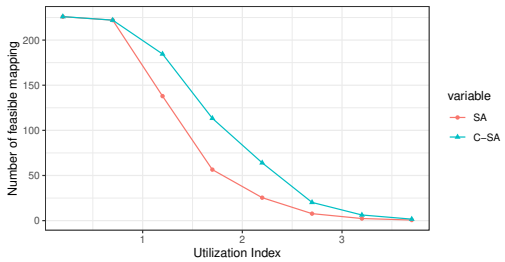
(d) 100% of comm. workload



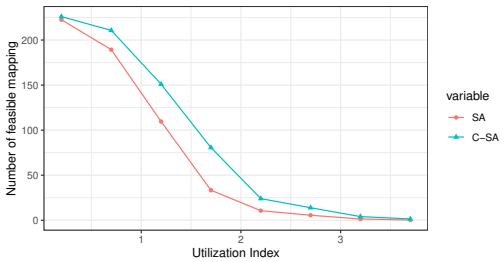
(e) 20% of comm. workload



(f) 40% of comm. workload

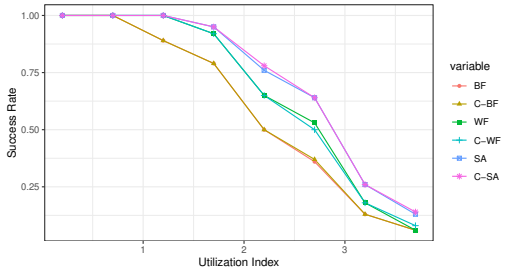


(g) 80% of comm. workload

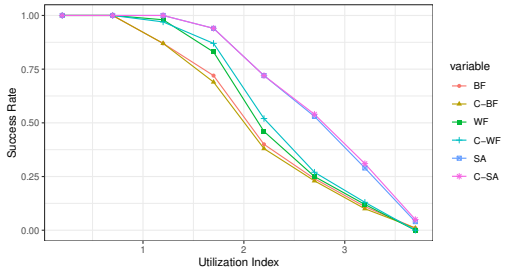


(h) 100% of comm. workload

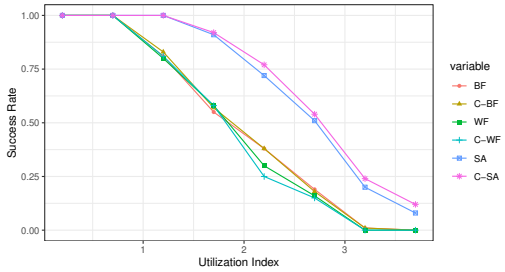
Figure 6.7.: Schedulability ratio of 100 task set by utilization. SA-parameters (t_{init} = 1.0, t_{min} = 10^{-3}, t_{α} = 0.90)



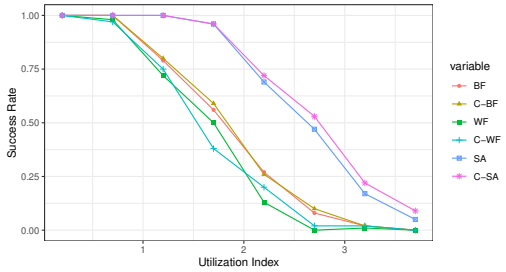
(a) 20% of comm. workload



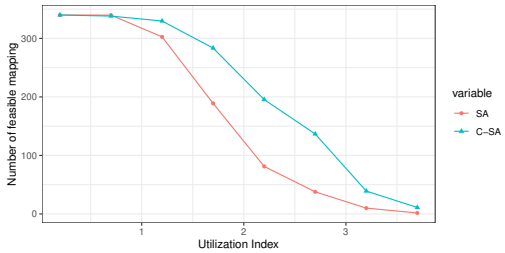
(b) 40% of comm. workload



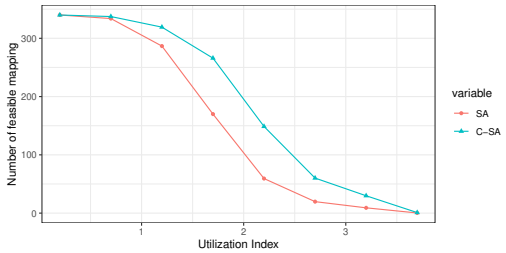
(c) 80% of comm. workload



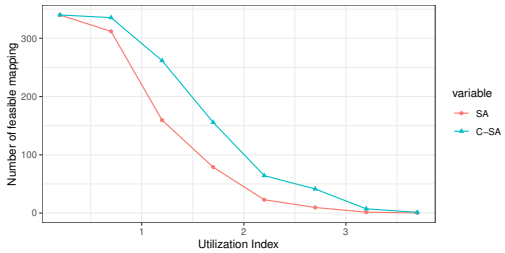
(d) 100% of comm. workload



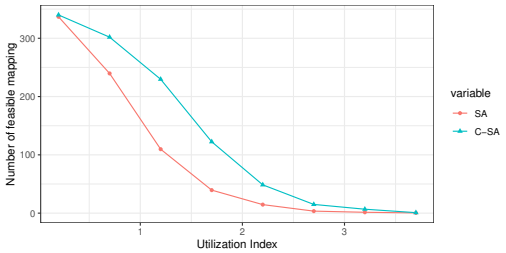
(e) 20% of comm. workload



(f) 40% of comm. workload

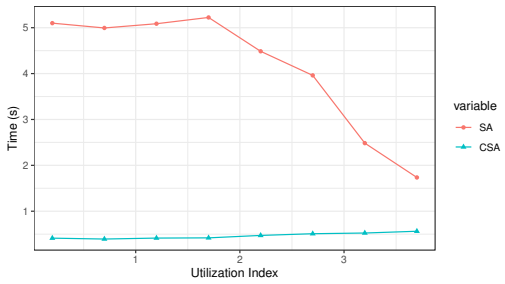


(g) 80% of comm. workload

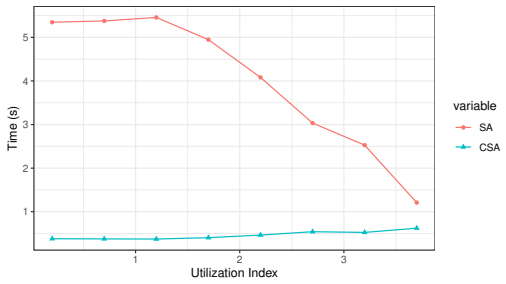


(h) 100% of comm. workload

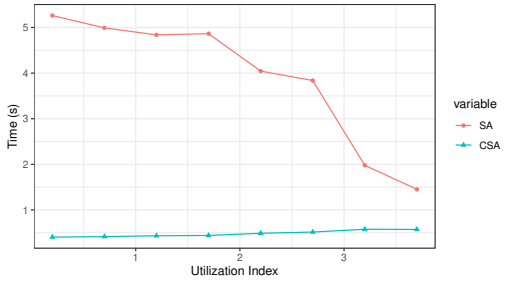
Figure 6.8.: Schedulability ratio of 100 task set by utilization. SA-parameters (t_{init} = 1.0, t_{min} = 10^{-6}, t_{α} = 0.95)



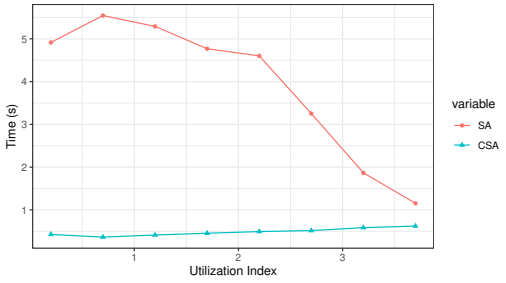
(a) 20% of comm. workload



(b) 40% of comm. workload



(c) 80% of comm. workload



(d) 100% of comm. workload

Figure 6.9.: Time related to find the initial solution of both SA and C-SA

Conclusion & Outlook

Conclusion

In this thesis, we tackled the mapping problem technique on different sides. This problem remains NP-hard as it is considered highly combinatorial. Indeed, combining the task allocation with the communication resource reservation makes the solution space significantly wide when finding an optimal solution with a relatively reasonable amount of time requires advanced techniques.

The on-chip communications themselves have been studied separately. In fact, The Network-on-Chip routers offer several modes of arbitrations. The priority-preemptive is refereed as efficient with the real-time applications. However, such arbitration mode is not predictable as the community simply provides an upper-bound of latency estimation. On the other hand, TDMA protocol is known to be predictable and safe as we can compute the communication latency *a priori*, and therefore, avoid unpredictable behavior of critical systems at run-time. In the first contribution (4^{th} chapter), we contrasted both techniques with both worst-case analysis and simulations.

In the 5^{th} chapter, we demonstrated a promising technique to reduce the analysis complexity of DAG tasks through the use of partitioned scheduling and the intermediate deadline assignment. Thanks to the DAG model and its ability to express the calculation tasks as well as their communications. We used EDF as it is considered as an optimal scheduler on single core, while TDMA has been used on purpose to provide conflict-less schema on routers' arbitration.

In the 6^{th} chapter, we moved toward a more complete architecture that includes a main memory. The latter is denoted as an off-chip memory as it is located outside in the point-of-view of the NoC. At this fact, the NoC is featured by dedicated units to exchange with the main-memory, denoted by I/O clusters. Moreover, we introduced the particularity of gathering a subset of cores to constitute calculation clusters in order to provide isolation, and we showed the benefits of core clustering to a set of synthetic experiments. We extended the DAG model to the AER model to express both on-chip and off-chip communications, and we proposed a meta-heuristic-based mapping technique for fast solution space exploration.

Outlook

Adaptive Clustering & heterogeneous platforms

In this thesis, we defined the core clustering arbitrarily regardless the task set and performance requirements, when we assumed an identical cores platform. However, as modern real-time systems require heterogeneous cores, it is better to define the clusters size and shape following the tasks requirements. Thus, we plan to propose advanced clustering techniques on heterogeneous NoC platforms.

Mixed-criticality Systems

Throughout the thesis, we studied the CPS applications considering them as hard real-time tasks. In such systems, violating the time deadline is forbidden, when it is more tolerable concerning soft real-time systems. Beyond that, modern CPS applications tend to cohabit both systems onto the same execution platforms aiming at optimizing the resources when executing them separately remains extremely costly. In this perspective, it is interesting to study the mapping of mixed-criticality tasks onto Network-on-Chip.

Global scheduling

The global scheduling by opposite of partitioned scheduling allows the task migration between processors, and therefore, considered as work-conserving. We plan to deploy G-EDF as task scheduler onto NoC's computing engines.

Personal publications

Journals

- Chawki Benchehida, Mohammed Kamel Benhaoua, Houssam-Eddine Zahaf, Giuseppe Lipari. "Memory-Processor co-scheduling for Real-time Tasks on Network-on-chip manycore architectures". Int. J. of High Performance Systems Architecture
- Chawki Benchehida, Mohammed Kamel Benhaoua, Houssam-Eddine Zahaf, and Giuseppe Lipari. "An analysis and simulation tool of real-time communications in on-chip networks: a comparative study". SIGBED Rev. 17, 1 (February 2020), 5–11.

Conferences & Workshops

- Chawki Benchehida, Mohammed Kamel Benhaoua, Houssam-Eddine Zahaf, Giuseppe Lipari. "Task and Communication Allocation for Real-time Tasks to Networks-on-Chip Multiprocessors". Second International Conference on Embedded & Distributed Systems (EDiS), Oran, Algeria, 2020, pp. 9-14.
- Chawki Benchehida, Benhaoua Mohammed Kamel, Houssam Eddine Zahaf, Giuseppe Lipari. "An analysis and Simulation Tool of Real-Time Communications in On-Chip Networks". The Embedded Operating Systems Workshop EWiLi'19, New York, USA.
- Chawki Benchehida, Benhaoua Mohammed Kamel, Houssam Eddine Zahaf, Giuseppe Lipari. "Real-time Communications in On-Chip Networks". The 12th Junior Researcher Workshop on Real-Time Computing JRWRTC 2018, Poitiers, France.

Bibliography

Al Faruque, Mohammad Abdullah, Rudolf Krist, and Jórg Henkel (2008). "Adam: run-time agent-based distributed application mapping for on-chip communication." In: *2008 45th ACM/IEEE Design Automation Conference*. IEEE, pp. 760–765 (cit. on p. 38).

Andersson, Björn, Konstantinos Bletsas, and Sanjoy Baruah (2008). "Scheduling arbitrary-deadline sporadic task systems on multiprocessors." In: *2008 Real-Time Systems Symposium*. IEEE, pp. 385–394 (cit. on p. 19).

Andersson, Björn and Jan Jonsson (2000). "Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition." In: *Proceedings Seventh International Conference on Real-Time Computing Systems and Applications*. IEEE, pp. 337–346 (cit. on p. 18).

Andersson, Björn and Eduardo Tovar (2006). "Multiprocessor scheduling with few preemptions." In: *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*. IEEE, pp. 322–334 (cit. on p. 19).

Ascia, Giuseppe, Vincenzo Catania, and Maurizio Palesi (2004). "Multi-objective mapping for mesh-based NoC architectures." In: *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 182–187 (cit. on p. 40).

Audsley, Neil et al. (1993). "Applying new scheduling theory to static priority pre-emptive scheduling." In: *Software engineering journal* 8.5, pp. 284–292 (cit. on p. 14).

Audsley, Neil C (1991). *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. Citeseer (cit. on p. 70).

Bandyopadhyay, Sanghamitra et al. (2008). "A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA." In: *IEEE Transactions on Evolutionary Computation* 12.3, pp. 269–283. DOI: 10.1109/TEVC.2007.900837 (cit. on p. 83).

Barros, Jesse Barreto de, Mauricio Ayala-Rincon, and Carlos Humberto Llanos Quintero (Nov. 2019). "Application of an Adaptive Genetic Algorithm for Task Mapping Optimisation on a Wormhole-based Real-time Network-on-Chip." en. In: *2019 IX Brazilian Symposium on Computing Systems Engineering (SBESC)*. Natal, Brazil: IEEE, pp. 1–7. ISBN: 978-1-72816-318-5. DOI: 10.1109/SBESC49506.2019.9046081. URL: https://ieeexplore.ieee.org/document/9046081/ (visited on 07/13/2020) (cit. on pp. 41, 43).

Barros, Jessé Barreto de, Renato Coral Sampaio, and Carlos Humberto Llanos (2019). "An adaptive discrete particle swarm optimization for mapping real-time applications onto network-on-a-chip based MPSoCs." en. In: *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design - SBCCI '19*.

São Paulo, Brazil: ACM Press, pp. 1–6. ISBN: 978-1-4503-6844-5. DOI: 10.1145/3338852.3339835. URL: http://dl.acm.org/citation.cfm?doid=3338852.3339835 (visited on 05/05/2020) (cit. on pp. 41, 43).

Baruah, S. and N. Fisher (2005). "The partitioned multiprocessor scheduling of sporadic task systems." In: *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, 9 pp.–329. DOI: 10.1109/RTSS.2005.40 (cit. on p. 18).

Baruah, Sanjoy et al. (2012). "A Generalized Parallel Task Model for Recurrent Real-time Processes." In: *2012 IEEE 33rd Real-Time Systems Symposium*, pp. 63–72. DOI: 10.1109/RTSS.2012.59 (cit. on p. 82).

Baruah, Sanjoy K, Louis E Rosier, and Rodney R Howell (1990a). "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor." In: *Real-time systems* 2.4, pp. 301–324 (cit. on pp. 15, 16).

Baruah, Sanjoy K., Louis E. Rosier, and Rodney R. Howell (Nov. 1990b). "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor." In: *Real-Time Systems* 2.4, pp. 301–324. ISSN: 1573-1383. DOI: 10.1007/BF01995675. URL: https://doi.org/10.1007/BF01995675 (cit. on p. 70).

Bate, Iain and Alan Burns (1997). "Schedulability analysis of fixed priority real-time systems with offsets." In: *Proceedings Ninth Euromicro Workshop on Real Time Systems*. IEEE, pp. 153–160 (cit. on p. 70).

Bell, Shane et al. (2008). "Tile64-processor: A 64-core soc with mesh interconnect." In: *2008 IEEE International Solid-State Circuits Conference-Digest of Technical Papers*. IEEE, pp. 88–598 (cit. on p. 33).

Ben-Itzhak, Yaniv et al. (2012a). "HNOCS: Modular open-source simulator for Heterogeneous NoCs." In: *2012 International Conference on Embedded Computer Systems (SAMOS)*, pp. 51–57. DOI: 10.1109/SAMOS.2012.6404157 (cit. on p. 34).

Ben-Itzhak, Yaniv et al. (2012b). "HNOCS: modular open-source simulator for heterogeneous NoCs." In: *Embedded Computer Systems (SAMOS), 2012 International Conference on*. IEEE, pp. 51–57 (cit. on p. 51).

Benchehida, Chawki, Mohammed Kamel Benhaoua, Houssam Eddine Zahaf, et al. (2019). "An analysis and Simulation Tool of Real-Time Communications in On-Chip Networks: A Comparative Study." In: *EWILI'19* (cit. on p. 63).

Benchehida, Chawki, Mohammed Kamel Benhaoua, Houssam-Eddine Zahaf, et al. (July 2020). "An Analysis and Simulation Tool of Real-Time Communications in on-Chip Networks: A Comparative Study." In: *SIGBED Rev.* 17.1, pp. 5–11. DOI: 10.1145/3412821.3412822. URL: https://doi-org.ressources-electroniques.univ-lille.fr/10.1145/3412821.3412822 (cit. on p. 75).

Benhaoua, Mohammed Kamel, AmitKumar Singh, Abou El Hassan Benyamina, et al. (2015). "DynMapNoCSIM: A Dynamic Mapping SIMULATOR for Network on Chip based MPSoC." In: *Journal of Digital Information Management* 13.1 (cit. on p. 51).

Benhaoua, Mohammed Kamel, AmitKumar Singh, AEH Benyamina, et al. (2014). "Heuristic for accelerating run-time task mapping in NoC-based heterogeneous MPSoCs." In: *Journal of Digital Information Management* 12.5, p. 293 (cit. on p. 63).

Benini, Luca and Giovanni De Micheli (2002). "Networks on chips: A new SoC paradigm." In: *computer* 35.1, pp. 70–78 (cit. on pp. 24, 63, 75).

Bhardwaj, Kshitij and Rabindra K Jena (2009). "Energy and bandwidth aware mapping of IPs onto regular NoC architectures using multi-objective genetic algorithms." In: *2009 International Symposium on System-on-Chip*. IEEE, pp. 027–031 (cit. on p. 40).

Bhuiyan, Ashikahmed et al. (2018). "Energy-efficient real-time scheduling of DAG tasks." In: *ACM Transactions on Embedded Computing Systems (TECS)* 17.5, pp. 1–25 (cit. on p. 75).

Binkert, Nathan et al. (2011). "The gem5 simulator." In: *ACM SIGARCH Computer Architecture News* 39.2, pp. 1–7 (cit. on p. 51).

Bolotin, Evgeny et al. (2004). "QNoC: QoS architecture and design process for network on chip." In: *Journal of systems architecture* 50.2-3, pp. 105–128 (cit. on p. 51).

Bonilha, Iae Santos, Osmar Marchi dos Santos, and Leandro Indrusiak (Nov. 2014). "Heuristics for Mapping Real-Time Applications to NoC-Based Architectures Using Genetic Algorithms." en. In: *2014 Brazilian Symposium on Computing Systems Engineering*. Manaus, Amazonas, Brazil: IEEE, pp. 144–149. ISBN: 978-1-4799-8559-3. DOI: 10.1109/SBESC.2014.33. URL: http://ieeexplore.ieee.org/document/7091181/ (visited on 05/01/2020) (cit. on pp. 41, 43).

Bruch, J.V. et al. (Nov. 2017). "Deadline, Energy and Buffer-Aware Task Mapping Optimization in NoC-Based SoCs Using Genetic Algorithms." en. In: *2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)*. Curitiba: IEEE, pp. 86–93. ISBN: 978-1-5386-3590-2. DOI: 10.1109/SBESC.2017.18. URL: https://ieeexplore.ieee.org/document/8116564/ (visited on 05/05/2020) (cit. on pp. 41, 43).

Cardona, Jordi et al. (Dec. 2018). "NoCo: ILP-Based Worst-Case Contention Estimation for Mesh Real-Time Manycores." en. In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. Nashville, TN: IEEE, pp. 265–276. ISBN: 978-1-5386-7908-1. DOI: 10.1109/RTSS.2018.00043. URL: https://ieeexplore.ieee.org/document/8603219/ (visited on 05/06/2020) (cit. on pp. 41, 43).

Carpenter, John et al. (n.d.). *A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms.* (Cit. on p. 16).

Carvalho, Ewerson, Ney Calazans, and Fernando Moraes (2007). "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs." In: *18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP'07)*. IEEE, pp. 34–40 (cit. on p. 38).

Casini, Daniel et al. (2020). "A Holistic Memory Contention Analysis for Parallel Real-Time Tasks under Partitioned Scheduling." In: *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 239–252. DOI: 10.1109/RTAS48715.2020.000-3 (cit. on p. 81).

Catania, Vincenzo et al. (2015). "Noxim: An open, extensible and cycle-accurate network on chip simulator." In: *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 162–163. DOI: 10.1109/ASAP.2015.7245728 (cit. on p. 34).

Ceng, J. et al. (2008). "MAPS: An Integrated Framework for MPSoC Application Parallelization." In: *Proceedings of the 45th Annual Design Automation Conference*. DAC '08. Anaheim, California: Association for Computing Machinery, pp. 754–759. ISBN: 9781605581156. DOI: 10.1145/1391469.1391663. URL: https://doi.org/10.1145/1391469.1391663 (cit. on p. 36).

Chandra, Rohit et al. (2001). *Parallel programming in OpenMP*. Morgan kaufmann (cit. on p. 20).

Chatterjee, Navonil et al. (Mar. 2017). "Deadline and energy aware dynamic task mapping and scheduling for Network-on-Chip based multi-core platform." en. In: *Journal of Systems Architecture* 74, pp. 61–77. ISSN: 13837621. DOI: 10.1016/j.sysarc.2017.01.008. URL: https://linkinghub.elsevier.com/retrieve/pii/S1383762117300498 (visited on 05/01/2020) (cit. on pp. 41, 43).

Chen, Guangyu, Feihui Li, and Mahmut Kandemir (2007). "Compiler-directed application mapping for noc based chip multiprocessors." In: *ACM SIGPLAN Notices* 42.7, pp. 155–157 (cit. on p. 37).

Chou, Chen-Ling and Radu Marculescu (2007). "Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels." In: *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pp. 161–166 (cit. on p. 38).

Chou, Chen-Ling and Radu Marculescu (2008). "User-aware dynamic task allocation in networks-on-chip." In: *2008 Design, Automation and Test in Europe*. Ieee, pp. 1232–1237 (cit. on p. 38).

Chou, Chen-Ling, Umit Y Ogras, and Radu Marculescu (2008). "Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.10, pp. 1866–1879 (cit. on p. 38).

Dally, William J et al. (1992). "Virtual-channel flow control." In: *IEEE Transactions on Parallel and Distributed systems* 3.2, pp. 194–205 (cit. on pp. 31, 32).

Daneshtalab, Masoud et al. (2010). "A Low-Latency and Memory-Efficient On-chip Network." In: *2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*. Grenoble, France: IEEE, pp. 99–106. ISBN: 978-1-4244-7085-3. DOI: 10.1109/NOCS.2010.19. URL: http://ieeexplore.ieee.org/document/5507556/ (cit. on p. 76).

Davis, Robert I and Alan Burns (2011). "A survey of hard real-time scheduling for multiprocessor systems." In: *ACM computing surveys (CSUR)* 43.4, pp. 1–44 (cit. on pp. 12, 19).

Dick, R. P., D. L. Rhodes, and W. Wolf (Mar. 1998). "TGFF: task graphs for free." In: *Proceedings of the Sixth International Workshop on Hardware/Software Codesign. (CODES/CASHE'98)*, pp. 97–101. DOI: 10.1109/HSC.1998.666245 (cit. on p. 71).

Dick, Robert P, David L Rhodes, and Wayne Wolf (1998). "TGFF: task graphs for free." In: *Proceedings of the Sixth International Workshop on Hardware/Software Codesign.(CODES/CASHE'98)*. IEEE, pp. 97–101 (cit. on p. 91).

Dinechin, Benoit Dupont de et al. (2013). "A distributed run-time environment for the kalray mppa®-256 integrated manycore processor." In: *Procedia Computer Science* 18, pp. 1654–1663 (cit. on p. 33).

Durrieu, Guy et al. (2014). "Predictable flight management system implementation on a multicore processor." In: *Embedded Real Time Software (ERTS'14)* (cit. on p. 74).

Emberson, Paul, Roger Stafford, and Robert I Davis (2010). "Techniques for the synthesis of multiprocessor tasksets." In: *proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, pp. 6–11 (cit. on p. 91).

Fazzino, Fabrizio, Maurizio Palesi, and David Patti (2008). "Noxim: Network-on-chip simulator." In: *URL: http://sourceforge. net/projects/noxim* (cit. on p. 51).

Ferrandiz, Thomas, Fabrice Frances, and Christian Fraboul (July 2009). "A method of computation for worst-case delay analysis on SpaceWire networks." en. In: *2009 IEEE International Symposium on Industrial Embedded Systems*. Lausanne, Switzerland: IEEE, pp. 19–27. ISBN: 978-1-4244-4109-9. DOI: 10.1109/SIES.2009.5196187. URL: http://ieeexplore.ieee.org/document/5196187/ (visited on 03/04/2020) (cit. on p. 30).

Fonseca, José Carlos et al. (2015). "A multi-dag model for real-time parallel applications with conditional execution." In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pp. 1925–1932 (cit. on p. 75).

Ghosh, Pavel, Arunabha Sen, and Alexander Hall (2009). "Energy efficient application mapping to NoC processing elements operating at multiple voltage levels." In: *2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*. IEEE, pp. 80–85 (cit. on p. 39).

Giannopoulou, Georgia et al. (July 2016). "Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources." en. In: *Real-Time Systems* 52.4, pp. 399–449. ISSN: 0922-6443, 1573-1383. DOI: 10.1007/s11241-015-9227-y. URL: http://link.springer.com/10.1007/s11241-015-9227-y (visited on 11/09/2020) (cit. on pp. 42, 43, 76).

Gomony, Manil Dev, Benny Akesson, and Kees Goossens (2014). "Coupling TDM NoC and DRAM controller for cost and performance optimization of real-time systems." en. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*. Dresden, Germany: IEEE Conference Publications, pp. 1–6. ISBN: 978-3-9815370-2-4. DOI: 10.7873/DATE.2014.062. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?%20arnumber=6800263 (cit. on p. 76).

Hansson, Andreas, Kees Goossens, and Andrei Rădulescu (2005). "A unified approach to constrained mapping and routing on network-on-chip architectures." In: *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, pp. 75–80 (cit. on pp. 63, 75).

Harde, Tim et al. (2018). "Configurations and Optimizations of TDMA Schedules for Periodic Packet Communication on Networks on Chip." In: *RTNS*, pp. 202–212 (cit. on pp. 42, 63, 75).

Hassan, Mohamed (Dec. 2018). "On the Off-Chip Memory Latency of Real-Time Systems: Is DDR DRAM Really the Best Option?" en. In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. Nashville, TN: IEEE, pp. 495–505. ISBN: 978-1-5386-7908-1. DOI: 10.1109/RTSS.2018.00062. URL: https://ieeexplore.ieee.org/document/8603238/ (cit. on p. 80).

Hesham, Salma et al. (May 2017). "Survey on Real-Time Networks-on-Chip." en. In: *IEEE Transactions on Parallel and Distributed Systems* 28.5, pp. 1500–1517. ISSN: 1045-9219. DOI: 10.1109/TPDS.2016.2623619. URL: http://ieeexplore.ieee.org/document/7728147/ (visited on 04/04/2019) (cit. on pp. 25, 55, 63, 75).

Hossain, H. et al. (Mar. 2007). "Gpnocsim - A General Purpose Simulator for Network-On-Chip." In: *2007 International Conference on Information and Communication Technology*, pp. 254–257. DOI: 10.1109/ICICT.2007.375388 (cit. on p. 51).

Hu, Jingcao and Radu Marculescu (2003a). "Energy-aware mapping for tile-based NoC architectures under performance constraints." In: *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, pp. 233–239 (cit. on p. 39).

Hu, Jingcao and Radu Marculescu (2003b). "Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures." In: *2003 Design, Automation and Test in Europe Conference and Exhibition*. IEEE, pp. 688–693 (cit. on p. 39).

Hu, Jingcao and Radu Marculescu (2005). "Energy-and performance-aware mapping for regular NoC architectures." In: *IEEE Transactions on computer-aided design of integrated circuits and systems* 24.4, pp. 551–562 (cit. on p. 39).

Huang, Jia et al. (Feb. 2011a). "Energy-Aware Task Allocation for Network-on-Chip Based Heterogeneous Multiprocessor Systems." en. In: *2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*. Ayia Napa, Cyprus: IEEE, pp. 447–454. ISBN: 978-1-4244-9682-2. DOI: 10.1109/PDP.2011.10. URL: http://ieeexplore.ieee.org/document/5739050/ (visited on 05/01/2020) (cit. on pp. 41, 43).

Huang, Jia et al. (2011b). "Energy-aware task allocation for network-on-chip based heterogeneous multiprocessor systems." In: *2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*. IEEE, pp. 447–454 (cit. on p. 39).

Jang, Wooyoung and David Z. Pan (Oct. 2011). "Application-Aware NoC Design for Efficient SDRAM Access." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30.10, pp. 1521–1533. ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2011.2160176. URL: http://ieeexplore.ieee.org/document/6022008/ (cit. on p. 75).

Jerger, Natalie Enright and Li-Shiuan Peh (Jan. 2009). "On-Chip Networks." en. In: *Synthesis Lectures on Computer Architecture* 4.1, pp. 1–141. ISSN: 1935-3235, 1935-3243. DOI: 10.2200/S00209ED1V01Y200907CAC008. URL: http://www.morganclaypool.com/doi/abs/10.2200/S00209ED1V01Y200907CAC008 (visited on 01/29/2018) (cit. on p. 29).

Jin, Xi et al. (Aug. 2011). "Memory Access Aware Mapping for Networks-on-Chip." en. In: *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*. Toyama, Japan: IEEE, pp. 339–348. ISBN: 978-1-4577-1118-3. DOI: 10.1109/RTCSA.2011.31. URL: http://ieeexplore.ieee.org/document/6029862/ (visited on 05/01/2020) (cit. on p. 76).

Leung, Joseph Y-T and Jennifer Whitehead (1982). "On the complexity of fixed-priority scheduling of periodic, real-time tasks." In: *Performance evaluation* 2.4, pp. 237–250 (cit. on p. 14).

Lis, Mieszko et al. (June 2010a). "DARSIM: a parallel cycle-level NoC simulator." In: *MoBS 2010 - Sixth Annual Workshop on Modeling, Benchmarking and Simulation*. Ed. by Lieven Eeckhout and Thomas Wenisch. Saint Malo, France. URL: https://hal.inria.fr/inria-00492982 (cit. on p. 34).

Lis, Mieszko et al. (2010b). "DARSIM: a parallel cycle-level NoC simulator." In: *MoBS 2010-Sixth Annual Workshop on Modeling, Benchmarking and Simulation* (cit. on p. 51).

Liu, C. L. and James W. Layland (Jan. 1973). "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment." In: *J. ACM* 20.1, pp. 46–61. ISSN: 0004-5411. DOI: 10.1145/321738.321743. URL: http://doi.acm.org/10.1145/321738.321743 (cit. on pp. 14, 19, 51).

Lo, Moustapha et al. (n.d.). "IMPLEMENTING A REAL-TIME AVIONIC APPLICATION ON A MANY-CORE PROCESSOR." en. In: (), p. 11 (cit. on p. 42).

Lu, Zhonghai and Axel Jantsch (2007). "Slot Allocation Using Logical Networks for TDM Virtual-Circuit Configuration for Network-on-Chip." In: *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*. ICCAD '07. San Jose, California: IEEE Press, pp. 18–25. ISBN: 1424413826 (cit. on p. 67).

Mandelli, Marcelo, Alexandre Amory, et al. (2011). "Multi-task dynamic mapping onto NoC-based MPSoCs." In: *Proceedings of the 24th symposium on Integrated circuits and systems design*, pp. 191–196 (cit. on p. 38).

Mandelli, Marcelo, Luciano Ost, et al. (2011). "Energy-aware dynamic task mapping for NoC-based MPSoCs." In: *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, pp. 1676–1679 (cit. on p. 38).

Mehran, Armin, Ahmad Khademzadeh, and Samira Saeidi (2008). "DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip." In: *IEICE Electronics Express* 5.13, pp. 464–471 (cit. on p. 38).

Möller, Leandro, Leandro Soares Indrusiak, and Manfred Glesner (2009). "NoCScope: A graphical interface to improve Networks-on-Chip monitoring and design space exploration." In: *Design and Test Workshop (IDT), 2009 4th International*. IEEE, pp. 1–6 (cit. on p. 51).

Moraes, Fernando et al. (2004). "HERMES: an infrastructure for low area overhead packet-switching networks on chip." In: *INTEGRATION, the VLSI journal* 38.1, pp. 69–93 (cit. on p. 51).

Murali, Srinivasan, Luca Benini, and Giovanni De Micheli (2005). "Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees." In: *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pp. 27–32 (cit. on p. 39).

Murali, Srinivasan and Giovanni De Micheli (2004). "Bandwidth-constrained mapping of cores onto NoC architectures." In: *Proceedings design, automation and test in Europe conference and exhibition*. Vol. 2. IEEE, pp. 896–901 (cit. on p. 40).

Nikolic, Borislav, Robin Hofmann, and Rolf Ernst (2019). "Slot-Based Transmission Protocol for Real-Time NoCs-SBT-NoC." In: *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (cit. on pp. 42, 63, 75).

Norazizi Sham Mohd Sayuti, M., Farida Hazwani Mohd Ridzuan, and Zul Hilmi Abdullah (June 2019). "Task mapping and routing optimization for hard real-

time Networks-on-Chip." en. In: *Bulletin of Electrical Engineering and Informatics* 8.2, pp. 414–421. ISSN: 2302-9285, 2089-3191. DOI: 10.11591/eei.v8i2.1395. URL: http://beei.org/index.php/EEI/article/view/1395 (visited on 05/02/2020) (cit. on pp. 41, 43).

Ostler, Chris and Karam S Chatha (2007). "An ILP formulation for system-level application mapping on network processor architectures." In: *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, pp. 1–6 (cit. on p. 39).

Ozturk, Ozcan, Mahmut Kandemir, and Seung W Son (2007). "An ilp based approach to reducing energy consumption in nocbased CMPS." In: *Proceedings of the 2007 international symposium on Low power electronics and design*, pp. 411–414 (cit. on p. 39).

Palencia, J.C. and M. Gonzalez Harbour (1998). "Schedulability analysis for tasks with static and dynamic offsets." en. In: *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*. Madrid, Spain: IEEE Comput. Soc, pp. 26–37. ISBN: 978-0-8186-9212-3. DOI: 10.1109/REAL.1998.739728. URL: http://ieeexplore.ieee.org/document/739728/ (cit. on pp. 88–90).

Perret, Quentin et al. (2016a). "Mapping hard real-time applications on many-core processors." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pp. 235–244 (cit. on pp. 42, 43).

Perret, Quentin et al. (2016b). "Mapping hard real-time applications on many-core processors." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pp. 235–244 (cit. on p. 76).

Pham, Phi-Hung et al. (2010). "Design and implementation of backtracking wave-pipeline switch to support guaranteed throughput in network-on-chip." In: *IEEE transactions on very large scale integration (VLSI) systems* 20.2, pp. 270–283 (cit. on p. 25).

Poli, Riccardo, James Kennedy, and Tim Blackwell (2007). "Particle swarm optimization." In: *Swarm intelligence* 1.1, pp. 33–57 (cit. on p. 40).

Rhee, Chae-Eun, Han-You Jeong, and Soonhoi Ha (2004). "Many-to-many core-switch mapping in 2-D mesh NoC architectures." In: *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.* IEEE, pp. 438–443 (cit. on p. 39).

Sahu, Pradip Kumar and Santanu Chattopadhyay (Jan. 2013). "A survey on application mapping strategies for Network-on-Chip design." In: *Journal of Systems Architecture* 59.1, pp. 60–76. ISSN: 13837621. DOI: 10.1016/j.sysarc.2012.10.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S1383762112000902 (visited on 04/02/2020) (cit. on p. 39).

Sahu, Pradip Kumar, Tapan Shah, et al. (2013). "Application mapping onto mesh-based network-on-chip using discrete particle swarm optimization." In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.2, pp. 300–312 (cit. on pp. 63, 75).

Sahu, Pradip Kumar, Putta Venkatesh, et al. (2011). "Application mapping onto mesh structured network-on-chip using particle swarm optimization." In: *2011 ieee computer society annual symposium on vlsi*. IEEE, pp. 335–336 (cit. on p. 40).

Sayuti, M. Norazizi Sham Mohd and Leandro Soares Indrusiak (Aug. 2013). "Real-time low-power task mapping in Networks-on-Chip." en. In: *2013 IEEE*

Computer Society Annual Symposium on VLSI (ISVLSI). Natal, Brazil: IEEE, pp. 14–19. ISBN: 978-1-4799-1331-2. DOI: 10.1109/ISVLSI.2013.6654616. URL: http://ieeexplore.ieee.org/document/6654616/ (visited on 05/01/2020) (cit. on pp. 41, 43).

Sayuti, M. Norazizi Sham Mohd and Leandro Soares Indrusiak (Dec. 2015a). "A constructive task mapping algorithm for hard real-time embedded NoCs." en. In: *2015 IEEE Conference on Systems, Process and Control (ICSPC)*. Bandar Sunway, Malaysia: IEEE, pp. 123–128. ISBN: 978-1-4673-7655-6. DOI: 10.1109/SPC.2015.7473571. URL: http://ieeexplore.ieee.org/document/7473571/ (visited on 05/02/2020) (cit. on pp. 41, 43).

Sayuti, M. Norazizi Sham Mohd and Leandro Soares Indrusiak (Mar. 2015b). "Simultaneous Optimisation of Task Mapping and Priority Assignment for Real-Time Embedded NoCs." en. In: *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. Turku, Finland: IEEE, pp. 692–695. ISBN: 978-1-4799-8491-6. DOI: 10.1109/PDP.2015.84. URL: http://ieeexplore.ieee.org/document/7092794/ (visited on 05/02/2020) (cit. on pp. 41, 43).

Shi, Z. and A. Burns (Apr. 2008). "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching." In: *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, pp. 161–170. DOI: 10.1109/NOCS.2008.4492735 (cit. on p. 54).

Shi, Zheng and Alan Burns (Dec. 2010). "Schedulability analysis and task mapping for real-time on-chip communication." en. In: *Real-Time Systems* 46.3, pp. 360–385. ISSN: 0922-6443, 1573-1383. DOI: 10.1007/s11241-010-9108-3. URL: http://link.springer.com/10.1007/s11241-010-9108-3 (visited on 05/01/2020) (cit. on pp. 41, 43, 75).

Singh, Amit Kumar, Wu Jigang, et al. (2009). "Mapping algorithms for noc-based heterogeneous mpsoc platforms." In: *2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*. IEEE, pp. 133–140 (cit. on p. 38).

Singh, Amit Kumar, Thambipillai Srikanthan, et al. (2010). "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms." In: *Journal of Systems Architecture* 56.7, pp. 242–255 (cit. on p. 38).

Srinivasan, Krishnan and Karam S Chatha (2005). "A technique for low energy mapping and routing in network-on-chip architectures." In: *Proceedings of the 2005 international symposium on Low power electronics and design*, pp. 387–392 (cit. on pp. 40, 63, 75).

Srinivasan, Krishnan, Karam S Chatha, and Goran Konjevod (2006). "Linear-programming-based techniques for synthesis of network-on-chip architectures." In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.4, pp. 407–420 (cit. on p. 39).

Stankovic, John A. (1988). "Misconceptions about real-time computing: A serious problem for next-generation systems." In: *Computer* 21.10, pp. 10–19 (cit. on p. 8).

Still, Lloyd Robert and Leandro Soares Indrusiak (Mar. 2018). "Memory-Aware Genetic Algorithms for Task Mapping on Hard Real-Time Networks-on-Chip." en. In: *2018 26th Euromicro International Conference on Parallel, Distributed and*

Network-based Processing (PDP). Cambridge: IEEE, pp. 601–608. ISBN: 978-1-5386-4975-6. DOI: 10.1109/PDP2018.2018.00101. URL: https://ieeexplore.ieee.org/document/8374524/ (visited on 05/05/2020) (cit. on pp. 41, 43).

Tosun, Suleyman (2011). "Cluster-based application mapping method for Network-on-Chip." In: *Advances in Engineering Software* 42.10, pp. 868–874 (cit. on p. 39).

Vangal, Sriram et al. (2007). "An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS." In: *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*. IEEE, pp. 98–589 (cit. on p. 33).

Vyas, Kartika, Naveen Choudhary, and Dharm Singh (2013). "NC-G-SIM: A Parameterized Generic Simulator for 2D-Mesh, 3D-Mesh & Irregular On-chip Networks with Table-based Routing." In: *Global Journal of Computer Science and Technology* (cit. on p. 51).

Wang, Jian et al. (2011). "Bandwidth-aware application mapping for NoC-based MPSoCs." In: *Journal of Computational Information Systems* 7.1, pp. 152–159 (cit. on p. 40).

Wang, Yun and Manas Saksena (1999). "Scheduling fixed-priority tasks with preemption threshold." In: *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306)*. IEEE, pp. 328–335 (cit. on p. 88).

Whitley, Darrell (1994). "A genetic algorithm tutorial." In: *Statistics and computing* 4.2, pp. 65–85 (cit. on p. 40).

Wu, Yifan, Zhigang Gao, and Guojun Dai (2014). "Deadline and activation time assignment for partitioned real-time application on multiprocessor reservations." In: *Journal of Systems Architecture* 60.3. Real-Time Embedded Software for Multi-Core Platforms, pp. 247–257. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2013.11.011. URL: http://www.sciencedirect.com/science/article/pii/S138376211300266X (cit. on p. 69).

Xiong, Qin, Zhonghai Lu, et al. (2016). "Real-Time Analysis for Wormhole NoC: Revisited and Revised." en. In: *Proceedings of the 26th edition on Great Lakes Symposium on VLSI - GLSVLSI '16*. Boston, Massachusetts, USA: ACM Press, pp. 75–80. ISBN: 978-1-4503-4274-2. DOI: 10.1145/2902961.2903023. URL: http://dl.acm.org/citation.cfm?doid=2902961.2903023 (visited on 02/01/2019) (cit. on p. 54).

Xiong, Qin, Fei Wu, et al. (Sept. 2017). "Extending Real-Time Analysis for Wormhole NoCs." en. In: *IEEE Transactions on Computers* 66.9, pp. 1532–1546. ISSN: 0018-9340. DOI: 10.1109/TC.2017.2686391. URL: http://ieeexplore.ieee.org/document/7884964/ (visited on 04/04/2019) (cit. on p. 54).

Zahaf, H. et al. (2019). "The Parallel Multi-Mode Digraph Task Model for Energy-Aware Real-Time Heterogeneous Multi-Core Systems." In: *IEEE Transactions on Computers* 68.10, pp. 1511–1524 (cit. on p. 75).

Zahaf, Houssam-Eddine, Abou El Hassen Benyamina, et al. (2017). "Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms." In: *Journal of Systems Architecture* 74, pp. 46–60. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2017.01.002. URL: http://www.sciencedirect.com/science/article/pii/S138376211730019X (cit. on p. 62).

Zahaf, Houssam-Eddine, Nicola Capodieci, et al. (2019). "A C-DAG task model for scheduling complex real-time tasks on heterogeneous platforms: preemption matters." In: *arXiv preprint arXiv:1901.02450* (cit. on p. 68).

ZAHAF, Houssam-Eddine, Giuseppe Lipari, Smail Niar, et al. (2020). "Preemption-Aware Allocation, Deadline Assignment for Conditional DAGs on Partitioned EDF." In: *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, pp. 1–10 (cit. on p. 75).

Zhou, Wenbiao, Yan Zhang, and Zhigang Mao (2006). "An application specific NoC mapping for optimized delay." In: *International Conference on Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006.* IEEE, pp. 184–188 (cit. on p. 40).