Practical Strategies to Monitor and Control Contention in Shared Resources of Critical Real-Time Embedded Systems

Jordi Cardona

Universitat Politècnica de Catalunya

Computer Architecture Department

PhD Thesis

Doctoral program on Computer Architecture

February 2023

Practical Strategies to Monitor and Control Contention in Shared Resources of Critical Real-Time Embedded Systems

Jordi Cardona

February 2023

Universitat Politècnica de Catalunya
Computer Architecture Department

A Thesis submitted in fulfillment of
the requirements for the degree of
Doctor of Philosophy in Computer Architecture

Director: Francisco J. Cazorla, PhD, Barcelona Supercomputing Center
Codirector: Carles Hernández, PhD, Universitat Politècnica de València

Acknowledgements

First of all, I would like to deeply thank my Doctoral Thesis advisors Francisco J. Cazorla and Carles Hernández and also Jaume Abella and Enrico Mezzetti for their guidance, mentoring and for giving me the opportunity to undertake this Thesis in the Computer Architecture and Operating Systems group (CAOS) at Barcelona Supercomputing Center (BSC-CNS). I also want to thank the rest of the members of the CAOS group (the current ones but also the ones that have already left) as they have always been available to provide feedback on my work, give me good advice to improve it, and provided me help and courage when things didn't work at the first time.
Furthermore, I would like to acknowledge the BSC institution for financially supporting my Master's and Doctoral studies, the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P, and the HiPEAC Network of Excellence for partially funding this work.
Last but not least, I would like to deeply thank my entire family and especially my parents Lourdes and Ramon, my sister Núria, my partner Manu, as well as and my closest friends for their unconditional support during all these years. Without you, this Thesis would not have been possible. My most heartfelt thanks for the great confidence you have shown in me. *"Un trosset d'aquesta Tesi també és vostra"*.

Abstract

The performance needs in Critical Real-Time Embedded Systems (CRTES) in domains like automotive, avionics, railway and space have been steadily on the rise in the last decade. Unprecedented computational power is required to realize increasingly complex and performance-eager applications that are emerging as the key factors for the competitiveness of embedded products. To meet these computational needs, CRTES industry has been increasingly resorting to high-performance multicore and manycore processors that can cater for the required performance in a cost-efficient manner. Relatively simple bus-based multicore processor designs have been successfully deployed in general-purpose computing to provide good performance at low energy and area cost. However, these solutions quickly become ineffective with larger core counts as the bus becomes a major performance bottleneck. For this reason, bus-based designs have started migrating to Networks on Chips (NoCs) designs like trees, rings, meshes or torus topologies, which are being increasingly adopted in CRTES to offer multiple point-to-point connections and to allow exchanging several packets in parallel despite the new challenges that they entail.

Software timing is a paramount concern in the design and deployment of CRTES as correctness of the provided functions is typically not only determined by the delivered results, but also by the time at which those results are delivered. Domain-specific safety standards advocate for the adoption of software timing analysis techniques in the software development life-cycles as a necessary step to ascertain and guarantee all functions in the system execute timely and to prevent timing misbehavior to arise at run time. Timing concerns become particularly relevant to support three main phases in the CRTES development process: First, during the *timing verification phase* (budgeting), in the early stages of the software 'V' development cycle, worst-case timing analysis is meant to derive trustworthy upperbounds to tasks' execution time. Second, during the *timing validation* phase, in the late software development stages, timing properties are empirically assessed to confirm the correctness of the selected time budgets by gathering supporting evidence while running the whole system under analysis on the actual target. Finally, those phases are complemented by the design and deployment of techniques for the *enforcement of timing behavior* that put in place safety measures to prevent timing violations during system operation. These measures monitor the use of resources of the running tasks, either the CPU or shared hardware resources.

However, performing an efficient, industrial-quality timing analysis in the presence of complex multicore and manycore processors with complex hardware features like NoCs is complicated by the non-negligible timing interference arising when multiple cores contend in parallel to the same shared hardware resources through the interconnects. Multicore interference causes the execution time of a task to depend on the other tasks in the system, making it extremely difficult to characterize software timing in a trustworthy yet tight manner. The main challenges arise (i) in the budgeting phase, when deriving tight upper-bounds to the worst theoretical contention impact that requests can experience in traversing the NoC from their source to their destination, and (ii) in the validation phase, when tracking the actual multicore contention tasks generate on each other. Despite some real-time hardware accelerator designs, including NoCs, have been proposed to better factor multicore contention in Worst-Case Execution Time (WCET), such proposals entail high re-designing and re-verifying costs, making them unattractive for the CRTES market. For this same reason, Commercial-off-the-shelf (COTS) hardware solutions as wormhole NoCs (wNoCs) are typically adopted even in CRTES, though with some restrictions in the configuration as a trade-off between performance and timing predictability.

This Thesis addresses the problem of enabling efficient and effective contention analysis and characterization in NoC-based CRTES. While most of the works have focused on the verification phase, this Thesis addresses all three main steps in the overall software timing analysis in manycore processors. On the verification side, we propose EOmesh and NoCo solutions that optimize wNoCs' configuration to obtain tight and reduced WCET estimates while improving CRTES performance. On the validation phase, we propose a technique to breakdown the contention that cores generate each other in wNoC. To that end, we introduce a Golden Reference Value (GRV) on top of a PairWise Contention (PWC) metric that accurately identifies contention sources in wNoCs and provides a detailed multi-dimension contention breakdown, and a comparative analysis on the evolution of source contention identification in wNoCs. Finally, we cover a fundamental requirement in the enforcement phase by proposing MCCU, a software/hardware solution that performs fine-grain tracking of cores accesses in shared resources and enables, via configuration registers, to prevent cores to cause more interference on its contenders than budgeted by the system designer.

In summary, this Thesis focuses on facilitating the adoption of COTS NoC-based manycore processors in CRTES and presents a set of software and low-overhead solutions that contribute to optimizing software timing verification, validation and enforcement.

Contents

List of Figures

List of Tables

List of Abbreviations

ACMNoC	Accurate Monitoring of Network on Chips contribution.
AD	Autonomous Driving.
ASIL	Automotive Safety Integrity Level.
BSC	Barcelona Supercomputing Center.
BW	Bandwidth.
CABA	Cycle Accurate Bit-Accurate.
CAOS	Computer Architecture and Operating Systems group.
COTS	Commercial-off-the-shelf.
CRTES	Critical Real-Time Embedded Systems.
CT	Contender Task.
DAG	Directed Acyclic Graph.
DAL	Development Assurance Level.
DMA	Direct Memory Access.
EOmesh	Even Odd Mesh contribution.
EPI	European Processor Initiative.
ES	Embedded Systems.
ET	Execution Time.
EVT	Extreme Value Theory.
flit	Flow Control Unit.
FPGA	Field-Programmable Gate Array.
FUB	Functional Unit Block.
GPP	General-Purpose Processor.
GRV	Golden Reference Value.
HoL	Head of Line.

HPC High-Performance Computing.

ILP Integer Linear Programming.
IR Injection Rate.
IRQ Interrupt Request.
ISR Interrupt Service Routine.

lrc local router contention.

MBPTA Measurement-Based Probabilistic Timing Analysis.
MBTA Measurement-Based Timing Analysis.
MCCU Maximum Contention Control Unit contribution.
MCU MicroController Unit.
MOET Maximum Observed Execution Time.
MPSoC Multi-Processor System-on-Chip.

NGMP Next Generation Microprocessor.
NIC Network Interface Communication.
NoC Network on Chip.
NoCo Network on Chip Optimization contribution.
nWCET normalized WCET.

phit Physical Unit.
PMC Performance Monitoring Counter.
PME Processor or Memory Element.
PMU Performance Monitoring Unit.
PWC PairWise Contention.
PWC-GRV PWC-GRV contribution.

QoS Quality of Service.

rrc remote router contention.
RTES Real-Time Embedded Systems.

SIL Safety Integrity Level.
SoC System on Chip.
SQME Software solutions Quota Monitoring and Enforcement.
STA Static Timing Analysis.

TDMA Time Division Multiple Access.
TILED MC Tile-based Manycore.

TuA	Task Under Analysis.
UBD	Upper-Bound Delay.
V&V	Verification & Validation.
VC	Virtual-Channel.
walloc	weighted allocation.
WCD	Worst-Case Delay.
WCET	Worst-Case Execution Time.
WCTT	Worst-Case Traversal Time.
wmesh	Weighted Mesh.
wNoC	wormhole NoC.
zll	Zero-Load Latency.

Part I

Introduction

Chapter 1

Introduction

Research in academy and industry has played a key role in the creation of computers and their evolution ever since one of the considered ”fathers of the computers” Charles Babbage created the differential and analytical machines between 1847 and 1849. From then on, research collaboration among multiple disciplines such as physics, chemistry, and mathematics have led to the creation of the ENIAC computer in 1945 using the Von Neumann architecture, the discovery of the transistor, and the first DRAM and microprocessor (i4004) creation in the 1970s. Since then, the advances in manufacturing, integration technologies, and many other technological improvements have popularized the use of computers making them easy to use, small, fast, and affordable. While first computers were created mainly for military or mathematical purposes, now, computers have been adopted from recreational or gaming uses to research objectives, transportation (e.g. cars, plains), industry and health care. Even though all these computers share many design concepts, every computer system type has evolved to better fit a specific set of requirements depending on their target. When we talk about computer systems, normally we think of general-purpose desktops or laptops, or high-end computers for servers or supercomputers – High-Performance Computing (HPC) – with high computing capabilities. However, since many decades ago, Embedded Systems (ES) have gained relevance in our lives and keep growing. According to the BBC Research report, the ES' market is expected to grow from $207.3 billion in 2020 to $267.3 billion by 2025 [4].

Since their conception, ES have been specific self-contained computing systems with low computational power that can operate with low energy within a larger mechanical or electronic system. These systems have been widely popularized for their reliability, low energy consumption, low cost, and a relatively simple Verification & Validation (V&V) process. Nevertheless, over the years, the ES field has been steadily expanding as the computational power demands have increased and keep increasing, being comparable to the ones provided by high-performance chips [5, 6, 7]. At the same time, concepts such as reliability, energy consumption, and amenability to validation continue to be relevant in these systems' targets [8, 9]. For instance, examples of ES in a car include those responsible for oil temperature control, or even much more complex tasks, like the automatic braking system.

Real-Time Embedded Systems (RTES) are a subset of ES in which both the time and functional dimensions play a key role in their correct functioning. At a coarse level, we can classify these systems into different groups according to the criticality[1] of the provided function, the real-time computing requirements, or their architectural setups. Based on tasks criticality, we can distinguish between: *hard real-time systems*, where a system failure can lead to catastrophic consequences, including loss of human lives, and hence deadline misses must be prevented and corrective actions must be in place in case they occur (e.g. flight management system in an airplane); *firm real-time systems*, where deadline misses do not have catastrophic consequences and infrequent timing violation can be deemed acceptable (e.g. assembly line defects); *soft real-time systems* where timing violations lead to Quality of Service (QoS) degradation but does not imply any type of system corruption (e.g. streaming); or *non-critical real-time systems* where the system correctness is not bound to their timing behavior (e.g. a statistics unit that sends information about performance to a company server). At a finer level, each RTES domain has its own safety criticality level classification according to its reference standard. For instance, automotive systems build on 4 Automotive Safety Integrity Levels (ASILs) A to D, where ASIL-A represents the lowest criticality and ASIL-D the highest one, to characterize each subsystem risk of exposure, severity, and controllability of automotive hazards defined in the ISO-26262 [10] standard. Avionics systems, instead, refer to Development Assurance Levels (DALs), which go from DAL-E (less critical) to DAL-A (most critical), to classify the effects that systems' failures have on the aircraft, crew, and passengers specified in the DO178B/C [11] standard. Finally, the railway domain identifies 4 Safety Integrity Levels (SILs) with levels ranging from 0 to 4 (most critical level). Even though the integrity level concept is very similar in various industrial domains, there is no direct equivalence between criticality levels of the different standards as each domain has its own specific trait. In this Thesis, we focus on Critical Real-Time Embedded Systems (CRTES) that mainly fit into the hard real-time ES group.

1.1 Critical Real-Time Embedded Systems

CRTES require, in addition to functional correctness, *timing correctness*. While functional correctness implies that the system works according to functional specification, timing correctness refers to the execution of the corresponding functionalities in a timely manner, that is before specific time boundaries called *deadlines*. To ensure that deadlines will be met during the task's execution, CRTES need to go through a strict V&V process before being deployed. In this process, manufacturers need to provide evidence that both functional and non-functional requirements are met. To guide this process, certification standards provide specific guidelines and recommendations depending on the targeted domain (e.g. the ISO-26262 [10] for automotive or ECSS-Q-ST-80C [12] and ECSS-E-ST-40C [13] for the space domain).

[1]Criticality is a term strongly related to the consequence of what would happen if a system fails. The notion of criticality depends on the domain and how is defined in each different standard.

Providing evidence of the correctness of systems' components, especially in the timing domain, is not trivial. Certification standards share the prescription of V&V activities in view of obtaining trustworthy guarantees on the timing behavior of the system. However, determining the execution time that a task will exhibit at run time or whether a given task set is schedulable without missing any deadline can be very complex even in the simplest processor architectures as it depends on a lot of factors (e.g. hardware architecture, input data, compiler, system state, etc.). That makes that even a simple operation can take between a few and hundreds of cycles. Since deriving or observing the real Worst-Case Execution Time (WCET) of a task is practically unfeasible, timing analysis is in charge of deriving execution time bounds (estimates) that are required to be trustworthy and tight. Those estimates are ultimately used to determine if all tasks in a set can be scheduled without incurring timing violations.

1.1.1 Trends in CRTES

In the last decade, the CRTES industry has been experiencing an increasing demand for more computing power in several domains (i.e. automotive, aerospace, space, etc.) due to the emergence of new software applications that implement complex functionalities and manage huge amounts of data [14]. In the automotive field, artificial intelligence algorithms, object detection algorithms, and many other complex programs required in an Autonomous Driving (AD) system [15, 16], have already surpassed the amount of 100 million lines of code [17] and performance requirements are expected to grow by 100x up until 2024 [18].

Manycore processors are able to satisfy, in a cost-efficient manner, the computing needs of embedded real-time industry [8, 9, 19]. In this line, building as much as possible on manycore solutions deployed in the Commercial-off-the-shelf (COTS), high-performance (mainstream) market [6, 20] contributes to further reducing non-recurrent costs and increasing availability. The other side of the coin is that high-performance resources like shared caches, interconnection networks, and shared memories hamper timing V&V and in particular the derivation of tight and trustworthy WCET estimates [21, 22]. Indeed, the improper management of hardware shared resources (e.g. caches, interconnection networks, and memory) in multicores or manycores, can translate into worse average performance and WCET estimates than when run in isolation, hence defying the benefits of manycores in CRTES [19]. Interestingly, interconnection networks have been shown to be one of the shared resources with a larger impact on performance when cores access the shared memory hierarchy.

On small multicores, monolithic solutions such as buses implementing static arbitration algorithms were enough to interconnect cores with cache memories and provide good performance (i.e. AMBA bus [23]). Nevertheless, as the cores' complexity increased (i.e. increasing the petition Injection Rate (IR) in the bus) and so did the number of cores connected to the bus, buses became a bottleneck. On this account, distributed Network on Chips (NoCs) topologies starting from low complexity and reduced point-to-point links (e.g. trees or rings [24]) to more complex and more point-to-point connected NoCs (e.g. meshes [25] or torus), routing algorithms, arbitration policies, etc. have been proposed as a solution to the scalability problems provided by buses. In fact, multiple processors targeting CRTES already deploy NoCs (e.g. to connect 10 to 20 nodes), such as the Kalray MPPA 256 SoC [7] (e.g. a 4x4 mesh) and the Xilinx VERSAL Multi-Processor System-on-Chip (MPSoC) [26]. This is also the case of ongoing products like the SiPearl Rhea processor that will build on Zeus Cores [27] and an Arm CMN 600/700 based NoC [28], and that is being produced in the scope of the European Processor Initiative (EPI) [29].

1.2 Challenges to CRTES timing analysis

The adoption of high-performance NoC-based manycore processors in CRTES markets [8, 9] offers some obvious benefits like good performance, a simpler core design, and the capability to exploit parallelism, improving single applications' performance and scheduling multiple applications in the same core, hence maximizing hardware utilization. However, as introduced in the previous section, the use of high-performance hardware in CRTES introduces important challenges in the CRTES development process and jeopardizes, in particular, timing analyzability. These challenges arise at the different stages of the software engineering process, namely: verification, validation, and safety measures (enforcement) depicted in Figure 1.1.[2]

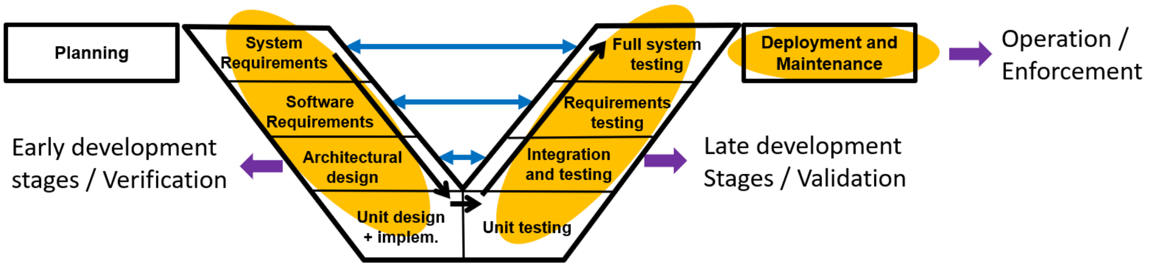


Figure 1.1: Standard V-model software development process.

1.2.1 Timing Verification (budgeting)

During the early stages of the software 'V' development cycle (see Figure 1.1), execution time budgets[3] (i.e. worst-case estimates) are associated with each task based on their worst-case timing requirements so that, at run time, tasks will complete within their allocated schedule slot. WCET analysis [30, 31, 32, 33, 34] is crucial for reasoning on timing correctness in a system as it provides the necessary inputs to the schedulability analysis [35, 36] that allows testing whether a task set is schedulable (e.g. all their budgets are met) under a given scheduling algorithm.

[2]Is worth noting that we use the terms V&V with the meaning they are provided in the automotive domain. In other domains (e.g. avionics), terms may be used differently.

[3]Timing budgeting is part of the verification process when estimates and checks are performed with the aim of building a dependable system. Validation, instead, is mostly based on testing activities once system components are implemented to identify design and verification flaws.

Determining the WCET of tasks is a challenging endeavor due to the increasing complexity of hardware (e.g. deep caches hierarchies or buffers) and software deployed on CRTES systems, which make WCETs often arising as a rare corner case considering multiple worst-case scenarios to occur at the same time. Two main families of timing analysis techniques are considered in industry and academia for deriving WCET estimates: Static Timing Analysis (STA) and Measurement-Based Timing Analysis (MBTA). STA has been the most used technique in CRTES, and it derives the WCETs mathematically and using abstract models. To achieve that, STA [37, 38] builds a timing model of the hardware making an abstraction of all potential hardware states and transitions. That allows deriving the worst-cases scenario without actually executing the task. Despite achieving mathematically sound WCET bounds, STA requires deep knowledge of the architectural design (which is not always available) and suffers from scalability issues, when dealing with hardware state explosion, already in relatively simple architectures. Those limitations can lead to unuseful, overly-pessimistic WCET estimates [39, 40]. In contrast, MBTA [30, 41, 42] executes the tasks on the real hardware to collects execution task measurements. The challenges in this methodology lie in triggering the worst-case scenarios that can possibly happen at run time, which can be particularly difficult or even impossible if the worst-case HW scenario cannot be willingly enforced by the programmer. As an attempt to overcome the limitations of consolidated timing analysis methods, Measurement-Based Probabilistic Timing Analysis (MBPTA) approaches [43, 44] have been proposed that builds on statistical tools to quantifying the *representativeness* of the collected measurements.

Timing analysis of multicores and manycore processor architectures is possibly even more challenging. The derivation of high-quality WCET bounds is hampered by the massive hardware resource sharing in those systems, which has been unequivocally identified as one of the main challenges for timing analysis [45, 46]. When multiple cores try to simultaneously access the same hardware resource (e.g. shared caches, interconnection networks, memory controllers, etc.), an arbiter prioritizes requests causing tasks to experience variable access latencies. This, in turn, introduces variability on tasks' execution time, typically referred to as multicore timing interference or contention impact. The impact of contention in tasks' execution time is non-negligible: it has been observed to cause more than 20x performance degradation with respect to execution in a single core [22]. The latency one task has when accessing a shared resource depends on the contention created from other cores when accessing the same shared resource (i.e. frequency, accesses interleaving, etc.). The (potential) variability stemming from contention makes it more difficult to obtain reasonably tight WCET estimates as simply accounting for the theoretical worst-case contention scenario may easily lead to overly-pessimistic and unusable bounds. Conceptually, the execution time of a task in a manycore can be broken down into two main components:

- *Isolation time*: time that a task takes to execute when running in isolation in the system (i.e. as if in single core).
- *Contention time*: additional delay created by co-running tasks in the other cores due to resource contention.

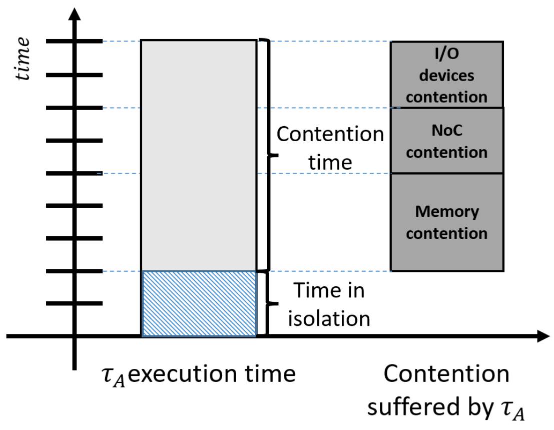


Figure 1.2: Execution task decomposition in manycores.

Figure 1.2 provides an example breakdown of a task execution time in the two components, with further classification of contention based on the shared resource it stems from. Deriving tasks WCET estimates in manycores is complex as tasks' execution time is a combination of the contention that occurs in all shared resources and circularly depends on the other co-runner tasks. Modifying even a single task in the system may require to re-iterate over the timing analysis effort for all tasks in the system to capture new contention scenarios, with evident repercussions on development costs.

To ease the WCET derivation and enable a more incremental task integration process, *time composability* is a desired property for WCET estimates. That means WCET estimates of a task are meant to account for the worst-possible contention scenario, regardless of the actual set of co-running tasks in the system [47]. For instance, time-composable WCET estimates can be obtained by upper-bounding the maximum access time to a shared resource (when being affected by the interferences) and later, applying this Upper-Bound Delay (UBD) to each access to the shared resource [34]. Nevertheless, accounting for the worst possible contention scenario will inherently lead to overly pessimistic and scarcely useful results, especially in high-performance systems that are not optimized for worst-case scenarios [39]. Research in this topic has been done mostly in two directions: A lot of research effort has been devoted to deriving efficient approaches to deal with contention impact. The explored solutions can be broadly classified in two main approaches: devising real-time specific designs [19, 48, 49] which allow controlling by design the amount of timing interference tasks can suffer, or making use of COTS high-performance processors [5, 20, 32, 50, 51] with contention-aware software solutions and (to a minor extent) hardware changes to make these systems more time predictable while still delivering high systems' performance.

Multicore timing interference is not a specific concern for NoC-based systems as it already arises in relatively simple bus-based solutions. However, NoCs possibly exacerbate the problem by bringing their own complexity in how and where contention can actually happen at run time. The main challenge in timing analysis of NoC-based systems lies in accurately estimating the latency that each packet will experiment in the NoC. From the multicore interference standpoint, it translates into determining the maximum contention that packets can suffer from the others, accounting not only for the contention that occurs in each traversed router (i.e. packet waiting for winning the arbitration round) but also for other complex packet blocking effects [52, 53, 54] like buffer backpressure [55, 56] (i.e. a packet cannot proceed because the packets ahead are also stalled). Real-time NoC designs have been proposed to minimize or even reduce contention in NoCs [49, 57, 58, 59]. However, similarly to other custom real-time designs, these solutions are generally not adopted for the high production and validation costs they incur. Several techniques have been proposed to derive more or less accurate bounds to the worst-case contention [52, 55, 60] and classify it into *direct* or *indirect* contention depending on whether contending flows share resources over their paths in the NoC or not [61]. Other works, instead, focus on the available COTS solutions from the high-performance market and propose methods to exploit the existing hardware support to improve time predictability [53, 62, 63, 64].

1.2.2 Timing Validation (testing)

Validation, which happens in the late development stages, relies on extensive testing intended to cause extreme – yet possible – behavior. The absence of timing failures, i.e. software components overrunning their time budget, serves as an argument to sustain the correctness of the software timing behavior.

Timing V&V heavily relies on extensive testing campaigns aiming at spotting latent timing failures in the system. Focusing just on end-to-end observed execution times against the estimated budgets is generally considered inadequate, especially in multicore and manycore systems, as observations can easily hide undesired contention behavior, decreasing the confidence in the collected evidence. This approach relates to current practice in specific domains such as automotive and avionics, in which end-users need to provide evidence of freedom from interference or to show how interference due to the use of shared hardware resources has been controlled or mitigated. For avionics, CAST-32A [65], now A(M)C 20-193 [66], explicitly calls for the identification and bounding of interference channels in multicores. In the case of the automotive domain, although certification practice is less strict (i.e. no certification authority exists yet), ISO26262 [10] asks for providing evidence of freedom from interference which, in practice, implies proving that tasks cannot exceed their allocated time budgets because of interference from other tasks. However, achieving full freedom from interference is challenging and, typically, interference that occurs in some shared resources can only be mitigated but not fully removed without replicating hardware, which will bring to a Functional Unit Blocks (FUBs) re-verification, or entail a huge average performance drop. Despite the majority of the works focus on how to achieve systems' timing isolation [22, 67, 68], an equally relevant issue in contention analysis is related to the capability, during testing, to identify the actual sources of contention, to detect which task causes timing overruns and propose, and to apply corrective actions. To that end, fine-grained metrics (e.g. shared resources' utilization and Worst-Case Delays (WCDs), collected by Performance Monitoring Counters (PMCs) in Performance Monitoring Units (PMUs), allow breaking down task execution at the level of CPU time and contention delay, thus enabling the detection of timing failures more accurately and promptly take corrective actions [39, 69].

When focusing on the NoCs behavior, the solutions proposed for deriving WCET and contention bounds, needed during verification [52, 61], cannot be applied for validation purposes to track execution (and contention) over observed measurements. The reason is that those solutions have been specifically designed to account for the worst possible delay packets can suffer in the NoC (bound) which does not necessarily correspond to detecting the actual delay of packets. The challenge lies in identifying the main sources of contention for a specific run in distributed NoCs as 2D meshes and locating where contention occurs. In fact, due to the distributed nature of the interconnects, contention can occur in multiple locations at the same time and can be propagated along the NoC. For these reasons, which add to the very limited fine-grain information available when monitoring or debugging NoCs, novel fine-grained metrics and tracing methodologies are advocated as key elements for easing the validation process and improving the trustworthiness of its results [39].

1.2.3 Safety Measures (enforcement)

Safety measures are required to prevent timing violations to occur during system operation. Typically, deployed solutions build on monitoring the use of resources of the running tasks to intercept system misbehaviors. During the validation stage, mechanisms are deployed to monitor the execution time of tasks so that corrective actions can be taken in case of a timing budget exhaustion (e.g. watchdogs [10]). The number of accesses to shared hardware resources (together with other resource usage metrics) is monitored and exploited [33] to take corrective actions when specific usage thresholds are exceeded.

Software solutions Quota Monitoring and Enforcement (SQME) techniques have been proposed to handle multicore contention in more generic processors with limited hardware support for time predictability [33, 67]. In general, SQME approaches build on limiting per task (core) maximum shared resources' utilization. To that end, the operating system monitors tasks' activities using the available hardware PMCs and suspends tasks execution when their assigned budget is exhausted.

Most solutions for timing-related safety measures have been designed and deployed on relatively simpler bus-based systems. When considering NoCs, as far as we know, no enforcement techniques based on quotas monitoring enforcement exist for distributed NoCs. For these systems, precisely ascribing the contention suffered by one task to another task is not trivial, despite the availability of similar hardware PMCs that can be tracked by the operating system as that available in bus-based NoCs. In fact, finer-grain contention information must be obtained and conveniently collected among all the nodes in the NoC, as we briefly introduced in Section 1.2.2.

Fortunately, it is possible to leverage some of the techniques that have been recently introduced to support mixed-criticality requirements in CRTES manycore systems [70, 71]. The solutions adopted in CRTES manycores to support the execution of tasks with different criticality/priority (i.e. mixed-criticality systems), can be also used to limit contention caused/suffered between tasks. Approaches for controlling or limiting how interference can arise in the NoC are particularly promising, such as priority-based flit-level preemption mechanisms [70, 71], where packets from higher criticality/priority tasks can preempt packets from lower criticality/priority tasks to advance in the NoC, or weighted arbitration mechanisms [64], that can directly prioritize certain ports and buffers favoring the advancing rate of packets coming from specific tasks.

1.3 Contributions

In this Thesis, we aim at designing new techniques to monitor and control contention in CRTES. We focus on the improvement and development of methods to achieve tighter WCET estimates and control contention in bus-based and NoC-based real-time systems while accounting for CRTES requirements such as time predictability, high performance, low area and energy utilization, and their V&V.

The contributions in this Thesis cover 3 major topics that correspond to the 3 software development activities with a bearing on timing verification, validation, and enforcement of safety measures. Specific contributions are enumerated below with 1 and 2 belonging to the Verification stage (e.g. techniques to improve WCET estimates), contributions 3 and 4 to the Validation stage (e.g. techniques to improve testing and detect timing failures), and contribution 5 to the Safety measures stage (e.g. enforcement techniques).

1. **EOmesh.** Reducing WCET estimates while improving average performance becomes challenging in 2D mesh NoC-based CRTES. While wmeshes have been proposed to alleviate NoCs pathological Bandwidth (BW) imbalance, they fail to fairly distribute BW in high contention scenarios. Thus, we propose even/odd mesh (EOmesh), an XY-YX predictable routing solution that delivers near-optimal BW across cores. To that end, EOmesh makes locally imbalanced arbitration decisions to achieve globally balanced BW distribution within XY-YX deterministic routing. We evaluate the performance increase, WCET reduction in a single thread and parallel applications, and different implementation possibilities with low hardware overhead.
2. **NoCo.** Finding the optimal NoC configuration that minimizes the WCD of packets and tasks' WCET is a multidimensional problem that requires optimizing tasks' mapping, routing allocations, and arbitration simultaneously. We propose NoCo, a stochastic ILP optimization that finds minimal WCD NoC configurations. We analyze the main wormhole NoC (wNoC) parameters that cause variability in WCD, we model their interactions and decompose the WCD reduction multidimensional problem into a stochastic-based optimization and an ILP formulation. Finally, we evaluate the effectiveness of our contribution with respect to other strategies that only focus on optimizing a subset of the NoC parameters.

Table 1.1: Thesis contributions organized by topic and focus.

Topic	**Subtopic**	**Name**	**Focus**	**Publication**
1. Verification (Budgeting)	Distributed NoCs	EOmesh	WCET estimates, High-Performance and NoC parameters optimization	TCAD'18
2. Verification (Budgeting)	Distributed NoCs	NoCo	WCET estimates, High-Performance and NoC parameters optimization	RTSS'18
3. Validation (Testing)	Distributed NoCs	PWC & GRV	Contention metrics, Actual contention and Overruns diagnosis	TODAES'23
4. Validation (Testing)	Distributed NoCs	ACM NoC	Contention metrics, Actual contention and Accuracy analysis	(To be submitted)
5. Safety Measures (Enforcement)	Centralized NoCs	MCCU	WCET estimates, Performance guarantees and Enforcement	DATE'19

3. **PWC-GRV.** Measuring and analyzing the actual contention that tasks generate to each other for diagnosis and validation purposes is challenging in manycore systems. While determining the sources of contention in centralized interconnection systems, it becomes more challenging in distributed NoCs as 2D meshes where contention information is distributed along the NoC nodes. To solve that, we define a metric (PWC) that captures and bounds the slowdown the packets generated by a task suffer from packets of other tasks, formulate a criterion (GRV) that classifies packets contention distinguishing the source and the location where contention takes place and propose an implementation via an off-line method for timing validation. We evaluate the metric results in a wide variety of traffic scenarios and our implementation scalability in different NoC sizes.
4. **ACMNoC.** In this contribution, we analyze the challenges associated with resource planning, offline testing, and online monitoring in NoC-based manycore by providing evidence for certification, system optimization, and overruns diagnosis. We show the need for timing V&V support in NoC-based manycores for CRTES and define a workflow to accurately estimate contention. We analyze how the accuracy of results depends on the NoC traffic nature and the information available from the NoC to perform the contention analysis.

We compare the methodology proposed in our third contribution with respect to other NoC metrics and evaluate the accuracy of the obtained results, showing that contention estimates and breakdown are key to achieving sufficient accuracy, as needed for V&V, system optimization, and overrun diagnosis at operation.

5. **MCCU.** MCCU improves current techniques that monitor and control contention a task suffers from other co-runner tasks in bus-based real-time ES. Existing techniques track and bound the number of requests to hardware shared resources that each task is allowed to perform. However, software-only solutions only work well when there is only one shared resource and type of request to track and bound, but do not scale to large number of shared resources and types of requests. In MCCU, we handle this general case and propose low-overhead hardware support that performs fine-grain tracking preventing a core to cause more interference on its contenders than budgeted. We evaluate the proposal in a simulator and we also quantify the hardware implementation overhead that does not require re-designing or re-verifying the existing FUBs.

Furthermore, Table 1.1 summarizes the target topic, the challenges tackled, the proposed technique and focus, and the publications associated with each of contributed item.

1.4 Thesis Organization

The Thesis is organized as follows:

- **Chapter 1 - Introduction.** Motivates our work by introducing the main topics and concepts of this Thesis. It contextualizes the evolution of CRTES and introduces the current status and the main challenges in the software timing engineering process.
- **Chapter 2 - Background.** Explains the background concepts, terminology and previous work targeting the challenges posed by the adoption of manycores with centralized (e.g. Bus-based) and distributed interconnections (e.g. NoCs at large and wormhole 2D mesh NoCs in particular) onto the timing Verification, Validation and Enforcement activities within the CRTES software development process.
- **Chapter 3 - Experimental Setup.** Presents the experimental setups used in the Thesis to evaluate our contributions. More in detail, it describes the methodology followed, the reference architectures used in the Thesis and their peculiarities, the way these architectures have been modeled, and other additional tools, benchmarks and applications that have been used to evaluate the contributions.

- **Chapters 4 to 8 - Contributions.** Each of those chapters presents a different Thesis contribution. The chapter structure is the same across all chapters: first a brief introduction to the topic and main points of the contribution are provided, followed by specific background information and the problem statement. Each chapter is then developing a detailed explanation of the contribution, the evaluation of the proposed technique, and finally some concluding remarks.
- **Chapter 9 - Conclusions and Future work.** Concludes the work done in this Thesis, discusses the impact and possible extensions to the proposed contributions, and identifies promising future research directions.

1.5 List of Publications

1.5.1 Conference Papers and Journal Articles

- **EOmesh: Combined Flow Balancing and Deterministic Routing for Reduced WCET Estimates in Embedded Real-Time Systems.** Jordi Cardona, Carles Hernandez, Jaume Abella, and Francisco J. Cazorla. *In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. Embeedded Systems Week (ESWEEK'18) Conference. Oct 2018, Torino, Italy.*
- **NoCo: ILP-based Worst-Case Contention Estimation for Mesh Real-Time Manycores.** Jordi Cardona, Carles Hernandez, Enrico Mezzetti, Jaume Abella and Francisco J. Cazorla. *In 39th IEEE Real-Time Systems Symposium (RTSS'18). Dec 2018, Nashville (Tennesse), USA.*
- **Maximum-Contention Control Unit (MCCU): Resource Access Count and Contention Time Enforcement.** Jordi Cardona, Carles Hernandez, Jaume Abella, and Francisco J. Cazorla. *Design, Automation and Test in Europe 2019 (DATE'19) Conference, Florence, Italy.*
- **Accurately Measuring Contention in Mesh NoCs for Timing Validation and Optimization** Jordi Cardona, Carles Hernandez, Jaume Abella, Enrico Mezzetti and Francisco J. Cazorla. *ACM Transactions on Design Automation of Electronic Systems 2023 (TODAES 2023).*

To be Submitted

- **Software Timing Verification and Diagnosis on NoC-based Manycores** Jordi Cardona, Carles Hernandez, Jaume Abella, Enrico Mezzetti and Francisco J. Cazorla.

1.5.2 Posters and other publications

- **Hardware Support for Enforcing Performance Guarantees in Multicore Processors** Jordi Cardona, Carles Hernández, Jaume Abella and Francisco J. Cazorla. *In Design, Automation Conference 2018 (DAC'18 Young Fellows program).*
- **Computing Worst-Case Contention Delays for Networks on Chip** Jordi Cardona, Carles Hernández, Jaume Abella and Francisco J. Cazorla. *In the 7th Barcelona Supercompuing Center International Doctoral Symposium, Spring 2020.*
- **Monitoring and Controlling Interconnect Contention in Critical Realtime Systems** Jordi Cardona. *In Design, Automation and Test in Europe (DATE'21 PhD Forum), Virtual Conference and Exhibition.*

Chapter 2

Background and Related work

This section provides the necessary background on the main topics covered in this Thesis including timing analysis practice, and the properties required from the platform used to improve time predictability. This section also covers the NoC designs considered in this Thesis and the Weighted Mesh (wmesh) upon which we build our contributions.

2.1 Timing Analysis

CRTES' safety standards require providing evidence of the correct system's behavior to pass the certification process. To that end, timing analysis contributes to the timing validation and verification process, mandatory in CRTES (1) providing estimates of tasks WCETs and (2) providing guarantees on those estimates so that tasks can meet their deadlines and produce schedulable task sets.

As we already introduced in Section 1.1, estimating the execution time of tasks can be very challenging in part due to the impact that jitter has on hardware shared resources during tasks' execution. Determining and reducing the sources of jitter can be very complex or even impossible to achieve in some cases since part of the factors causing this jitter (e.g. system status) cannot be controlled by the end user. A highly relevant source of jitter, and hence, execution time variability, relates to resource sharing between tasks (i.e. contention) in multicores and manycores.

As obtaining a deterministic execution time is practically unfeasible, timing analysis is in charge of deriving execution time bounds (estimates). These estimates will be later used as input for a scheduler, which will be in charge of doing schedulability analysis and, if possible, scheduling tasks in a way that deadline misses are avoided. In Figure 2.1, we show the typical distribution of the execution times of a task. We observe that the potential execution times of a task are comprised between the Best-Case Execution Time (BCET) and WCET. Notice that the observed execution times (between the minimal and the maximal observed execution time) only show a subset of all the potential execution times that the task can have. That is why it is so important to obtain good WCET estimates that tightly upper bound all the potential execution times of a task. In order to be useful, WCET estimates need to

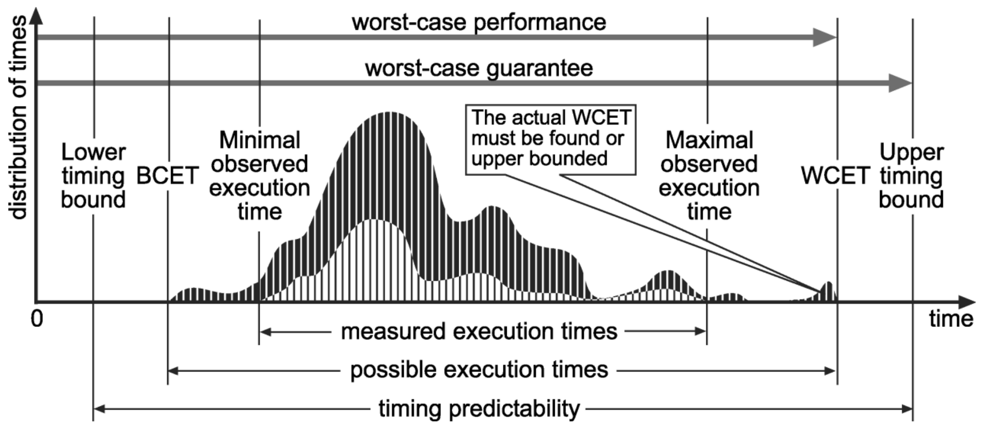


Figure 2.1: Execution time distributions [1].

be as tight as possible to the real WCET value (i.e. need to tightly upper-bound all potential task execution times). For instance, guaranteeing that the braking system of a car would take no more than 5 seconds to react is useless in practice because such a value is higher than the real maximum, and is not usable from a system perspective.

Until recently, CRTES built upon relatively-simple software running on relatively low-performance (and low-complexity) hardware. For instance, many avionics systems still build upon single-core processors with in-order execution and without cache memories. The advantage of those systems is that timing verification is relatively simple since execution time variability is low. Hence, the WCET can be tightly estimated using timing models of the system or by collecting measurements and adding a safety margin over the highest execution time observed to cover the impact of 'unobserved' effects.

2.1.1 Timing Analysis Paradigms

Several timing analysis techniques have been developed to derive WCET estimates (e.g. STA, MBTA, Probabilistic Timing Analysis (PTA), MBPTA, etc.). STA [37, 38] builds a timing model of the hardware making a Task Under Analysis (TuA) abstraction and modeling all potential hardware states and transitions at instruction or execution cycle level. After analyzing all possible transitions and states (i.e. at the end of the task execution), it selects the final state with a higher execution time that determines the WCET. However, STA becomes infeasible when the number of potential states grows exponentially due to multiple branches outcomes, memory addresses uncertainty, etc [40]. Although is it true that it is possible to decrease the states' complexity of the model making pessimistic assumptions (e.g. cache accesses always miss) to merge several states, it is not enough for making this type of analysis feasible for complex tasks or hardware due to the pessimism introduced. Furthermore, STA requires accurate hardware details that, in many cases, are protected by manufacturers' Intellectual Property [39]. MBTA [30, 43, 44], in contrast, builds upon execution time measurements of a task running on the target system, and does not require a timing model or system abstraction. The focus of this technique is on relating the representativeness of the execution times observed during evaluation with the ones that can occur during operation. In fact, MBTA usually uses the Maximum Observed Execution Time (MOET) plus a safety margin as the WCET estimate to account the unknowns. In the last years, other techniques like MBPTA [43, 44], also based on the measurement-based model, have emerged, and bound the WCET estimate to a probability of exceeding this number. The goal of MBPTA is to replace the arbitrary safety margin used in MBTA with a quantifiable margin that can be obtained using statistical methods such as Extreme Value Theory (EVT) [72].

2.1.2 Multicores and manycores

The adoption of high-performance resources like shared caches, interconnection networks and shared memories, hamper timing V&V and, in particular, deriving tight and trustworthy timing WCET estimates [21, 22]. That is explained by the impact that interferences have on tasks when running in parallel and try to access a given hardware shared resource at the same time. This contention effect causes high execution time variability as it makes tasks' execution time dependent on the use that other co-runner tasks' do of the systems' shared resources, which is hard to predict and model. Indeed, multicore timing interferences impact translates to a WCET and task response time increase, but also leads to overly pessimistic WCET estimates.

Focusing on the interconnect shared resource, given that it is one of the shared resources that has more impact in CRTES manycores, we can distinguish mainly two types of interconnects: centralized and distributed interconnects.

Centralized interconnects, like buses, are broadly used in high-performance manycores with a moderate number of cores as they provide good performance, with low power and area consumption. Furthermore, they are also commonly used in the CRTES domain (e.g. NXP T2080 in Figure 2.2 and T1040) as they have good timing predictability when they work under some restrictions (i.e. static arbitration), which allows tightly upper-bounding packets worst-case latency, hence easing the obtaining of WCET estimates and making them time-composable. Indeed, a lot of research has been done to compute safe contention bounds [32, 73] and derive tight WCET estimates [74, 75, 76, 77, 78] for centralized interconnects.

Regrettably, centralized interconnects can provide limited performance when running memory demanding applications that make buses to be a system bottleneck. To overcome this limitation, distributed NoCs like trees, rings or meshes have been created to provide more BW by sending multiple packets simultaneously taking advantage of their multiple point-to-point connections. As explained in Section 1.2, CRTES have also embraced the use of distributed NoCs to reach the high-performance computation levels demanded by the new functionalities despite complicating the V&V process. Two examples of that are Neoverse N1 (see Figure 2.3) and E1 processors that use the ARM CMN-600 Mesh NoC also used in some contributions in this Thesis. Academia and industry are putting a lot of effort to make these distributed topologies more time-predictable and to ease WCET estimation. In this direction, some works (e.g. rings [79], trees [80] and meshes [81, 82]) have been presented.

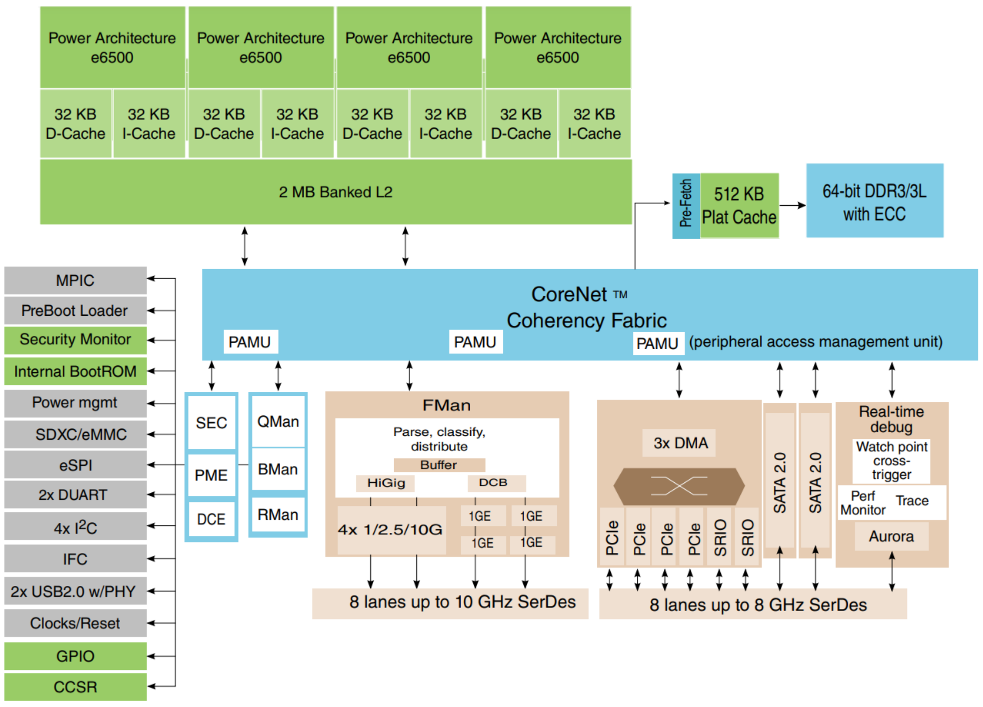


Figure 2.2: Block diagram of the quad-core T2080 processor using a centralized NoC [2].

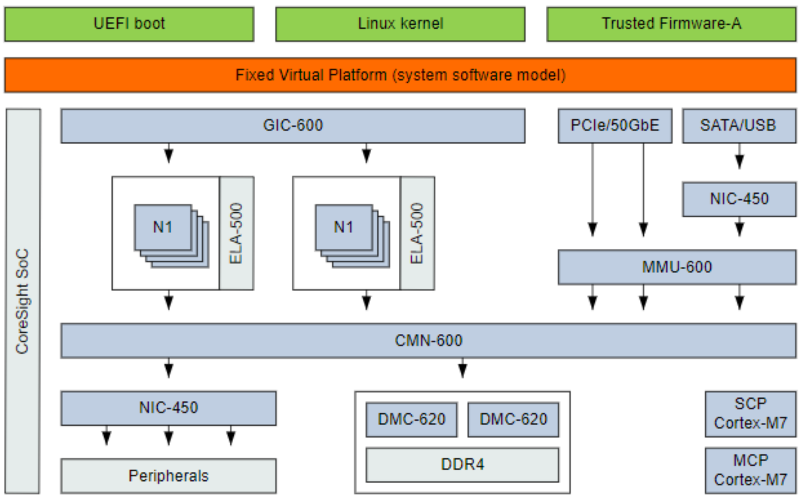


Figure 2.3: Block diagram of the Neoverse N1 edge reference design using ARM CMN-600 NoC [3].

2.2 NoCs

The need for CRTES manycores to connect their cores with other devices and components (e.g. main memory, shared caches), pushes for setting up efficient interconnects that are able to provide high-performance, low latency and time predictability. In this section, we provide the essential background and related work focusing on 2D meshes and their application in the software timing design for CRTES.

2.2.1 NoC fundamentals

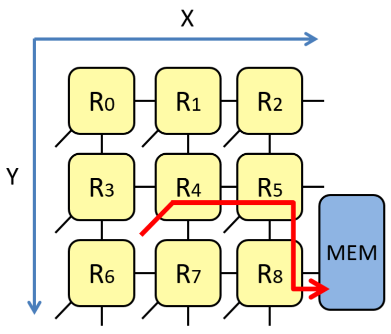
NoCs comprise point-to-point links, switches and routers to send packets across nodes. NoC characteristics impact the latency of requests to traverse them from source to destination. In this work, we consider a mesh NoC, as it has been shown to deliver high performance and has been implemented in several processors [5, 20]. Communication at the NoC level requires sending messages from source nodes to destination nodes (where Processor or Memory Elements (PMEs) units are attached). Messages can vary significantly depending on the information they bring (e.g. memory requests or replies, coherence messages). Network Interface Communications (NICs) with some Direct Memory Access (DMA) engines process messages and convert them into packets to be sent and processed inside interconnections. In general, packets are built of three parts: the Header part (includes NoC information such as source/destination ID, and packet information such as size and type), Payload (i.e. data) and Tail (contains the checksum field). In some networks, there is a fixed packet size so that network buffers can be sized in accordance with that. In most of cases, packets (payload and control information) will be split in Flow Control Units (flits) (logical unit information). However, flits may require higher BW than that allowed by NoC links. E.g., 500-bit packets cannot be sent atomically through a 128-bit wide link. Hence, packets will be split into Physical Units (phits) (physical unit information), which can be sent atomically as they are not wider than links. For simplicity, throughout this Thesis, we will talk about flits instead of phits. The group of flits that have the same source and destination node and usually share the same path (i.e. group of routers traversed) is known as flow.

NoCs are structured by nodes (i.e. PMEs) that are connected by links and routers that are in charge of routing and switching the packets by looking at their header information. Routing refers to the criteria followed to determine the paths that flits will follow when traversing the NoC from the source to the destination node. Conversely, switching refers to the criterion used to define the granularity at which flits can be sent or not across routers. To increase NoCs performance, buffers have been introduced into router input/output ports maximizing flits processing and ejection. However, buffers are limited, so it exists a mechanism that prevents buffers from overflowing by controlling nodes' and routers' flits arrival and departure. This mechanism is called flow control, and receives specific names when referring to each different implementation (e.g. Stop&Go, Credit-based, etc.). In some NoC topologies such as trees, meshes or torus, routers can have multiple input and output ports so that flits from different input ports can target the same output port at the same time. In these scenarios, an arbitration policy determines which flits can proceed to the output port and which others stall until the next arbitration round.

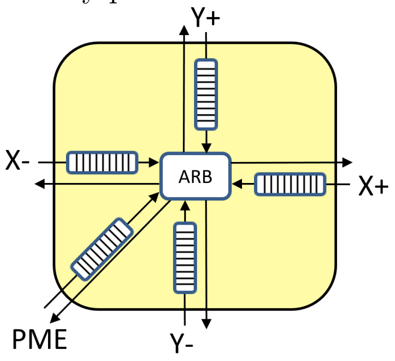
2.2.2 Real-Time wNoC Mesh

In this Thesis, we target NxM wormhole mesh NoCs, see Figure 2.4a and Figure 2.4b, as they are widely implemented in COTS manycores [6, 7, 20] and guarantee high performance. The main NoCs characteristics are summarized below:

Node. Each node, with an ID between 0 and $(N \times M) - 1$[1], see Figure 2.4a, comprises the router, serving as an interface to the mesh, and a PME. Each router comprises up to 5 bidirectional ports (i.e. input port and output port), see Figure 2.4b, and each input port comprises a queue to store flits. The main memory or memory controller is attached to one of the ports of one of the routers in a corner. Hence, such router has both a PME and a main memory port.



(a) 3x3 2Dmesh using XY routing.



(b) Router input/output ports.

Figure 2.4: Mesh concepts.

Routing. Deterministic routing policies like XY or YX routing are the preferred policies to allow the Worst Case Delay (WCD) estimation [51, 62] in CRTES. XY routing, for instance, forwards packets in the X direction first until reaching the column of the destination node and then forwards them in the Y direction. One of the best properties of these routing policies is that they take deterministic decisions which, in addition, result in minimal length routes (in terms of hops). Routing policies are defined over the set of routers $\mathcal{R}$ in the mesh and a given routing policy defines the traffic flows (F_i) per each source i as a set of packets that are sent over a predefined subset of routers in $\mathcal{R}$ (i.e., path). For example, in Figure 2.4a, flow F_4 identifies the set of packets that are sent over the path identified by routers $\{R_4, R_5, R_8\} \subseteq \mathcal{R}$. The term $\widehat{H_i}$ is used in this Thesis to denote the *ordered* set of routers traversed by F_i packets.

Switching and control flow. Wormhole switching is one of the most common approaches used in multicore processors [6, 20]. Indeed, recent works build upon wormhole switching and study contention effects such as, for instance, buffer backpressure [83], and how to take them into account in the late validation stages [55, 60]. In wormhole switching messages or if packets are split into several flits, the header flit that contains the destination information, the body flits contain the data, and the tail flit contains error detection codes' information. When the header flit of packet P_i arrives at R_n, the flit is stored in R_n's input port and the router allocates an entry queue in R_m (being R_m the next router in $\widehat{H_i}$). Once R_m can accept the header flit, the latter competes for an output port in R_n and, only when granted access, it traverses the router crossbar. Flits of a packet leave the router when the signals of the control flow mechanism from the next router inform

[1]Notice that nodes can also be identified by coordinates (x,y) being (0,0) R_0, (1,0) R_1 and so on so forth.

that there is an empty slot in the target queue. A new arbitration is performed once the entire packet has been sent. One of the main properties of wormhole switching is that switching is done at flit level, not at packet level, allowing flits to advance to the next router even if not all the flits of the packet fit in an input port, which maximizes buffer utilization and wNoC performance with reduced hardware resources.

Arbitration. Policies like round-robin are locally fair, which favors time predictability, and are easy to implement [84]. Round-robin policy fairly grants access to the input ports contending for the same output port, considering only those input ports that have a ready packet to be sent at arbitration time. When the header flit is stalled due to arbitration (contention), body flits are also stalled. Note that queues are typically sized with enough space to avoid bubbles in the packet transmission.

Packetization. As introduced in Section 2.2.1, packets consist of a group of flits whose number can vary depending on the type of data they bring inside. When computing WCD and WCET estimates, worst-case contention cases arise when contention occurs between long packets. Hence, to obtain a reliable WCET estimate, the longest packet size sent in the system is assumed as the size of all packets flowing in the interconnection. That allows to compute a reliable upper-bound, but brings pessimism to the WCET estimates that, in some cases, can end up not being useful at all (i.e. too pessimistic). To prevent that, packetization has been introduced as a way of having all packets of the same size in a way that longer packets are split into smaller packets. In some cases, packets size is set to the flit size, which simplifies the WCD and WCET estimation.

Virtual Channels. In canonical wormhole routers, Virtual-Channels (VCs) are used to multiplex physical channels: an input queue resource per port is assigned to a VC. Dynamic VC allocation increases the throughput of the wNoC, however, in the context of critical real-time, the dynamic allocation of VCs penalizes WCD [51]. Instead, static VC allocation can alleviate contention by providing isolation if the particular allocation of VCs allows for reducing the overlapping between the routes followed by the different flows. This ultimately reduces the WCD. With VCs, two different arbitration rounds take place in the router. A first arbitration determines the input port that is granted access to the output port. A second arbitration selects the VC in that input port that is granted access.

Weighted Mesh grants heterogeneous BW in the routers to the different flows to accommodate the different needs of different communication flows in the wNoC [85]. Weighted arbitration can be employed to achieve a globally-fair (homogeneous) BW allocation across cores [64]. This is achieved by using, for instance, a larger arbitration window in the case of round-robin, so that a larger number of slots is given to some ports so that the overall BW allocated to each core to the destination is homogenized. Conceptually, given a NoC with NxM nodes, globally-fair wmeshes reduce the BW for nodes whose allocated BW is above $\frac{1}{N \times M}$ for a given destination node and increase it for those whose BW is below. Such an approach has been shown doable and suitable for WCD estimation in the context of critical real-time systems [64]. However, weights leading to homogeneous BW allocation are not necessarily the best choice for any routing or workload. We will talk in more detail about wmeshes in Chapters 4 and 5.

Summary. WCET estimation is easier to carry out on deterministic routing policies. In particular, XY routing has been shown to be very suitable for meshes as WCD (and Worst-Case Traversal Time (WCTT)) bounds can easily be determined, and their implementation complexity is very low [51, 62]. XY routing builds upon forwarding packets in the X direction first until reaching the column of the destination node, and then forwarding them in the Y direction, thus making routes to have a minimal length (in terms of hops) and making routing decisions to be fully deterministic in any router (a single direction can be chosen for a given packet). In order to choose the packet to be granted access to an output port, we build upon round-robin arbitration, which has been shown to be a (locally) fair and easy to implement arbitration policy in NoCs [84]. However, for meshes, homogeneous round-robin at router level delivers highly unbalanced BW across nodes since nodes closer to the destination and those in routes with fewer flows receive higher overall BW than the other cores [85].

2.2.3 Related Work on Software Timing for NoC-based Systems

Timing-related aspects of multicore shared resources in CRTES are covered in different stages of the overall software engineering process (verification, validation and operation). In this section, we show a brief overview of the trends in the different stages.

2.2.3.1 Verification

As part of the system design and verification, the maximum time needed to execute each task (WCET) is estimated and assessed against its corresponding budget.

The estimation of time-composable WCET bounds [47] requires accounting for the worst potential congestion that can occur upon integration of other tasks in multicores. In the context of STA and NoCs, this requires computing the worst (theoretical) contention that requests can experience to traverse the NoC from their source to their destination [19, 86]. Then, contention bounds are added to the intrinsic latency of requests in a contention-free scenario.

Two main approaches exist to estimate such worst contention: WCTT [62] and WCD [51]. The former accounts for the worst overall contention that a request can experience, which may be caused (at least partially) by previous requests of the same task. Thus, an event causing contention on multiple requests of the same task would be accounted for multiple times. In particular, one request experiences such contention and the other ones get stalled due to backpressure of the other request. The latter, WCD, accounts only for the contention caused by requests of other tasks, thus not accounting for backpressure caused by requests of the same task. This avoids accounting multiple times for contention, thus delivering tighter (yet reliable) WCET estimates. In fact, measurement-based analysis of worst-case contention scenarios reveal that WCD tightly upper-bounds maximum contention [51]. Hence, we build on WCD for WCET estimation in this Thesis. In particular, we estimate the WCD, which can be added to requests latency in the context of STA, or can be factored in WCET estimates obtained with MBTA by adding WCD cycles per request to the WCET estimate obtained in a contention-free scenario.

For multicores and manycores, an extensive set of timing budgeting techniques are used to factor in the fact that multiple functionalities can be simultaneously executed. This includes hardware techniques to isolate tasks or bound the impact they can cause on each other [19, 49, 48, 87, 88] and software techniques that control the number of requests each task generates as a way to control the contention they can cause on others [31, 32, 33, 68, 78, 89].

In the case of wmesh NoCs, budgeting solutions result in a packet-level analysis of contention to accurately bound maximum contention experienced per packet and/or for the task as a whole [51, 55, 62, 90, 91, 92, 93]. Some of those solutions' breakdown contention across direct and indirect categories, where potential contention caused by flows sharing any resource with the one under analysis (e.g. an arbiter or a buffer in a router) is regarded as direct, and remaining contention as indirect. This concept is further reviewed in Section 6.2.

In the NoCs scope, budgeting works can be classified into three groups: (1) NoC designs for hard-real time systems, (2) optimization approaches to minimize NoC contention, and (3) analytic approaches to achieve tight budgeting.

Real time NoC designs. *Contention-free NoCs* have been traditionally considered the best fit to hard-real time applications. Contention free communication can be achieved by using time-division multiplexing [58, 94, 95], time-triggered architectures [96], and other ad-hoc wormhole-based designs like SurfNoC [97] or PhaseNoC [98]. However, as the number of cores included in the processor increases, time-division multiplexing approaches lose competitiveness since differences between worst-case performance and average performance increase with the number of cores [99]. Furthermore, customized NoCs will naturally find difficulties in being adopted by industry [100] since their implementation incurs high non-recurrent costs. *Priority-based flit-level preemption NoCs* based on customized wNoCs in which different priorities are assigned to the existing flows in the NoC have been proposed. The initial approaches [61, 70, 101] are based on assigning a different virtual-channel to each of the flows in the NoC such that the impact of NoC contention is accounted for in the schedulability analysis. Enhanced hardware designs have been proposed in this context to improve the performance of both, low-criticality and high-criticality traffic [71]. More recent approaches have shown that VCs can be reduced by ensuring a different priority (virtual channel) is assigned to the flows that actually share one or more links [102]. Virtual-channel requirements can be further reduced if the analysis includes the effect of contention of flows sharing priorities [103]. More recently, authors in [104] have shown that the model proposed in [61] and later enhanced [101] has some deficiencies. Based on the findings in [104], authors in [105] have proposed a tight alternative model for priority-preemptive real-time networks. *Best-effort wNoCs* can also be employed in the context of hard-real time systems by deriving latency upper-bounds as described in [62]. Using prioritization on a per-VC basis has proven to be an effective means to achieve tight latency bounds in wNoCs [61]. However, the use of per-VC prioritization becomes impractical when a significant number of flows exist in the network. To overcome this issue, the impact of VC sharing has been analyzed in [106] and [103]. However, while these approaches effectively reduce the number of VCs required, the timing guarantees obtained build upon a detailed knowledge of the characteristics of the software (applications and/or tasks) that will be executed in the deployed system and hence, do not meet incremental qualification requirements. The work in [107] has similar pros and cons, since the proposed solution guarantees specific BW allocation for guaranteed-service connections per port, by splitting the BW of output ports among best effort and guaranteed service connections.

However, as shown in [51] and [83], these bounds can be pessimistic when time-composability properties [47] have to be preserved. To mitigate this problem, authors in [64] proposed an alternative approach to meet CRTES requirements. In particular, they propose specific ways to derive time-composable WCD bounds without sacrificing average performance by allocating weights to arbiters so that fair BW allocation is achieved across cores, building upon wmeshes, wormhole networks [108].

wNoCs optimization approaches. In [35], Integer Linear Programming (ILP) models of heterogeneous multicores are used to find the optimal task layout that guarantees specific execution time bounds. In this approach, the internals of the communication infrastructure are not exposed to the solver. Since the adoption of NoCs as the primary interconnection architecture for multi/manycores, many works have targeted the problem of task mapping [109]. Authors in [110] proposed an ILP formulation for a contention-aware application mapping algorithm in tile-based NoCs using meshes with XY routing to minimize inter-tile network contention, being able to achieve a significant reduction of packet latency. In [111], the scheduling and mapping of tasks are combined to minimize packet latency and to increase predictability in the context of meshes with flexible routing decisions. To that end, communications have to follow a regular access pattern. A similar approach is the one in [112] that uses a constraint solver to find the mapping where contention is minimized. Once the mapping is fixed, communication bursts are mapped onto frames that are time-multiplexed in the context of 2D meshes with XY routing.

Analytic approaches. In the context of wNoCs, several works [113] have targeted analyzing NoC contention using queue models. Unfortunately, this analysis cannot be directly applied to precisely determine the WCD and/or the contention breakdown. To support wNoCs in hard real-time systems, using VC prioritization coupled with router architectures that support flit-level preemption have been proposed to limit inter-task interferences in the NoC [54, 61]. Nevertheless, COTS wNoCs do not implement such support. In processors with limited support for time predictability, approaches to analyze contention rely on the utilization of PMCs to track and enforce contention quotas [32, 33, 78, 89, 114]. These techniques require bounding the longest contention latency a request can suffer [21, 32, 115], and track the maximum number of accesses a task can perform to shared hardware resources [78]. Authors in [33, 78, 116] use PMCs to implement a software approach to enforce quotas on the maximum contention created by tasks during operation. To improve the quality of WCET/WCRT estimates for certain tasks, several works use PMCs to monitor tasks' activity and suspension mechanisms that can be implemented at the OS level [31, 33, 67, 68] or by hardware means [117].

Several works have shown that worst NoC interference can also be analytically or experimentally derived in regular wNoCs available in COTS manycores [51, 62, 118]. The impact of NoC interference in task execution time has been analyzed in [119] where a model to predict applications' slowdown is proposed. However, the proposed model does not provide a valid contention upper bound and therefore, is not valid in the context of CRTESs.

So that to fill this gap, different techniques focused on modeling wNoCs time and contention having complex effects when having multiple VCs, flows serialization or different buffer or packets sizes, have been proposed: Scheduling Theory, Compositional Timing Analysis, Recursive Calculus, Network Calculus and STA techniques applied to NoCs.

Scheduling Theory techniques [120] consider tasks contention managing flows priorities, multiple VCs and flit-level preemption so as to reduce contention suffered by high-priority flows. Network Calculus [52, 55, 90] techniques derive contention bounds by using mathematical and statistical tools and assuming per flow packets arrival and departure distribution curves. In this line, many recent works extending Network Calculus have been presented [55, 56, 60, 61]. Recursive Calculus techniques [93, 121] use branch and prune or branch, prune and collapse algorithms to compute end-to-end packets latency [91]. Compositional Timing Analysis techniques [83, 106] extend Network Calculus to obtain the worst-case contention delay and Statical Timing Analysis Techniques mathematically compute WCD and WCET by systematically considering the worst-possible contention case [51, 62].

Some of those solutions break down contention across direct and indirect categories [55, 60, 120], where potential contention caused by flows sharing any resource with the one under analysis (e.g. an arbiter or a buffer in a router) is regarded as direct, and remaining contention as indirect. This concept is further reviewed in Section 6.2.

Those techniques, in essence, intend to provide precise and tight upper-bounds to NoC contention during the system design and verification phase and the contention classification criteria that they use are directed to that end [61].

2.2.3.2 Validation

During late design stages, test campaigns are needed to validate that execution time budgets are not violated to provide evidence on the correctness of the selected time budgets (i.e. WCET or contention estimates). End-to-end execution times may easily hide effects across individual tasks or in specific NoC locations (e.g. routers) that detailed contention tracking/analysis would instead reveal. Otherwise, triggering those individual effects in a single test, so that they become visible in end-to-end execution times, may easily be beyond the reach of end-users, and it would not offer diagnosis information needed to identify the root causes of overruns. As we introduced in Section 1.2.2, focusing on total observed contention effects and end-to-end execution time only, is generally considered inadequate as observations can hide undesired contention behavior, decreasing the confidence that can be put on testing.

Specific domains such as avionics in CAST-32A [65] (directly) and other domains as automotive in the ISO26262 [10] (indirectly) call for proving freedom from interference. This is why interference-free designs were considered the default choice to implement multicore designs for CRTES in the past [48, 49, 58, 88]. Unfortunately, real-time specific designs (e.g. time-triggered ones and those based on Time Division Multiple Access (TDMA)) involve high non-recurrent costs that jeopardize their general adoption in the context of CRTES [100, 122]. Furthermore, other works have arisen in the same direction for mixed-criticality COTS systems processors [123].

2.2.3.3 Enforcement

Safety measures are deployed to prevent timing violations during system operation by monitoring the use of resources of the running tasks. In the former case, mechanisms are deployed to monitor the execution time of tasks to take corrective actions in case of a timing budget exhaustion (e.g. watchdogs [10]). In the latter case, the number of accesses to shared hardware resources (or any other usage measure) is monitored [33] to take corrective actions when specific usage thresholds are exceeded.

Hardware techniques: Several hardware designs providing predictability-aware resource sharing exist [19, 48, 49, 87, 88]. These designs combine the use of time-predictable arbitration schemes and provisioning each core/task with separate queues in each hardware shared resource to avoid a given core clogging that resource. Unfortunately, those changes require the re-verification of affected FUBs such as cache memories, buses and memory controllers among others that is the main reason for chip vendors not having adopted these solutions yet.

Software techniques: Contention models build on a timing estimate of the isolation, i.e. without contention, execution time of the task τ_a, C_a^{isol}, to derive a multicore estimate of τ_a's execution time (C_a^{muc}). To that end, the models bound the contention τ_a's requests can suffer in the access to hardware shared resources, Δ_a^{cont} so that $C_a^{muc} = C_i^{isol} + \Delta_a^{cont}$.

Δ_a^{cont} is computed combining (i) the longest contention each of its request can suffer, L_{max} and (ii) the number of requests, n_a, performed by τ_a and its contenders running in the other cores, referred to as $c(\tau_a)$. Let $cr(x \rightarrow a)$ be the number of τ_a requests that contend in the access to a shared resource with the requests of another concurrently running task t_x. It is defined as $cr_{b\rightarrow a} = min(n_a, n_b)$. Overall, the longest contention τ_a can suffer is defined as: $\Delta_a^{cont} = \sum_{\tau_x \in c(\tau_a)} cr_{x\rightarrow a} \times L_{max}$.

State-of-the-art works in this area can be classified into several groups. First, those that derive bounds to the number of requests performed by tasks to shared resources [33, 78] (n) which is affected, among others, by tasks' input data and the execution paths they traverse.

Another strand of works derive bounds to the maximum contention delay each request of a task can suffer [21, 32, 115] (L_{max}). These works build on time-predictable arbitration policies (e.g. round-robin) used in many shared hardware resources in high-performance embedded processors. For instance, for round-robin, the longest contention delay a request can suffer is $(Nc - 1) \times L$ when Nc is the number of requestors and L the duration of a request [34]. These works also build on documentation from processor manuals, when available, and carry strong validation through an extensive set of measurements to reduce the uncertainty on the validity of the bound [21, 115], which however cannot be removed.

And third, SQME works build on those bounds to derive contention models to bound Δ_a^{cont} in on-chip shared resources (e.g. bus and caches) [32, 33, 89] and enforce bounds (quotas) to task' access counts hence limiting Δ_a^{cont} [33, 78, 116]. SQME techniques reduce the impact of contention on WCET estimates of the monitored tasks' activity by stalling contender tasks if they go beyond their quota, preventing Δ_a^{cont} of the TuA from being violated [31, 33, 67, 68]. SQME techniques also allow defining *contention scenarios* and derive partially time-composable WCET estimates valid under those scenarios. The basic idea is to upper-bound the number of accesses of each type that potential contenders will put on the different shared resources and derive WCET estimates using those bounds. With this approach, WCET estimates are time-composable as long as the load put by contenders once the system is deployed is equal or lower than the one used to obtain these estimates.

Chapter 3

Experimental Setup

This chapter covers the experimental setup used and the research methodology followed to develop and evaluate the proposals made in this Thesis. The main aspects are grouped into: methodology, reference processor architectures, tools and simulators, and benchmarks and applications.

3.1 Methodology

We have followed the methodological workflow shown in Figure 3.1 that includes the simulation framework and describes how the different tools are interconnected in the workflow:

1. **Proposal:** After an analysis of the state of the art and its limitations, we define a proposal (roadmap) and implement it in an architectural simulator under specific setups. The proposal is made along with hypotheses on the results that we expect to obtain at the end of the workflow.
2. **Implementation:** The changes planned in the proposal are implemented in the simulators (i.e. SoClib, gNoCsim). Other elements of the workflow were implemented to analyze the results (scripts, parsers, statistical modules, etc.).
3. **Execute experiments:** Experiments are executed using well-known representative benchmarks (e.g. EEMBC automotive, Mediabench, self-made stressing kernels, etc.) and statistics about the benchmarks' execution are collected.
4. **Obtain results:** Results in form of PMC's counters, traces, feasible or optimal solutions, and WCET estimates are directly obtained from the benchmarks' execution or simulators.
5. **Results post-processing:** fine-grain and low-level results go through in-depth analysis. In the case of contention analysis, we developed a parser that creates multiple contention breakdowns and classifications. In the ILP solver case, we developed a parser that finds the optimal solutions from the feasible configurations found in the results step. We used an ILP solver for some of the works in this Thesis.

6. **Results evaluation:** results are assessed against related work and the hypotheses made in the proposal's initial step. Evaluation leads to a conclusion, depending on that conclusion a new proposal will be defined (modifying the architectural configurations or the hypothesis), additional experiments and results will be gathered, or the proposal will be completed finalizing the framework.

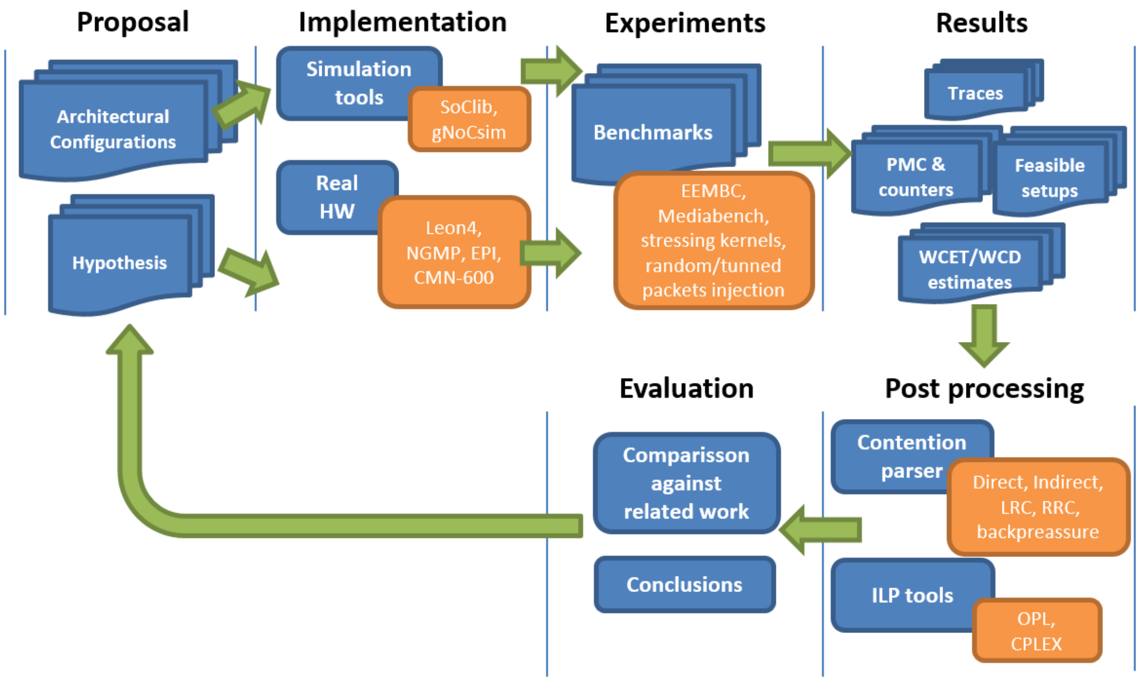


Figure 3.1: Methodology framework followed in the Thesis contributions.

3.2 Reference Processor Architectures

We model two multicore and manycore architectures representative of the state and coming trends of CRTES. We first model a Cobham Gaisler's Next Generation Microprocessor (NGMP), and second a 2D mesh-based variant from the NGMP processor modeling the EPI Processor. In this section, we provide a summary of the most relevant features of these architectures. Table 3.1 summarizes the modeled architectures features.

- **LEON 4.** LEON 4 [124] is a core based on its family predecessors LEON 3 and LEON 3 extension carried out as part of the PROXIMA project [125]. The LEON4 is a 32-bit in-order processor that has 7 pipeline stages (Fetch, Decode, Register File Access, Execute, Memory, Exception and Write-Back) with a branch predictor implementing an SPARC V8 Instruction Set Architecture (ISA). LEON4 is interfaced using the AMBA 2.0 AHB bus, has 1-cycle load latency, and can be efficiently implemented on Field-Programmable Gate Array (FPGA) and ASIC technologies. Its cache system consists of a separated instruction/data multi-set L1 cache (4-way) and an optional L2 cache.

- **NGMP.** The NGMP is a fault-tolerant quad-core LEON4 SPARC V8 processor that was first implemented as the radiation-hard System on Chip (SoC) GR740 and has eight port SpaceWire router, PCI initiator/target interface, CAN 2.0 interfaces and 10/100/1000 Mbit Ethernet interfaces. The GR740 was designed as the European Space Agency's NGMP and is part of the European Space Agency Roadmap. This device is targeted at high-performance general-purpose processing even though the architecture is suitable for symmetric, asymmetric multiprocessing and mixed-criticality applications (as their shared resources can be monitored). We use this reference architecture when later simulating our proposals in our cycle-accurate simulators in Chapter 8.

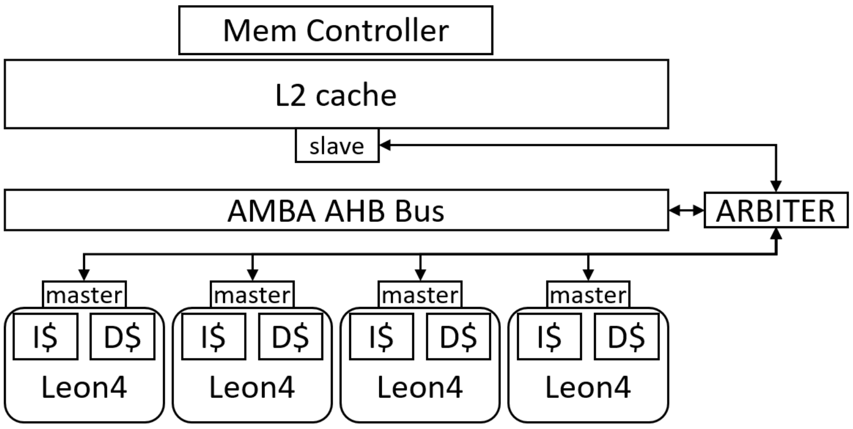


Figure 3.2: NGMP basic block diagram.

- **EPI processor + Tile based Manycore.** Part of this Thesis has been carried out under the scope of the EPI project. After the EPI General-Purpose Processor (GPP) was in progress in 2018 and only high-level architecture details were known, the Automotive Stream started thinking of a second version of the GPP processor adapted to the CRTESs field. As part of this Automotive Stream from the Barcelona Supercomputing Center Computing Architecture and Operating Systems group member (BSC-CAOS), we started analyzing how to apply timing analysis in an NoC-based system. As far as we know, EPI GPP will build on Zeus Cores (i.e. Neoverse N1 CPUs) and will be supporting a high-end memory subsystem, including HBM and DDR5 controllers.

  Since at the beginning of the Thesis there was neither a consolidated core nor NoC EPI simulators, we started working with the NGMP version modified to include a wormhole 2D Mesh interconnection with added features already known from the EPI processor. More in detail, we have modeled an XY-routing mesh network with 5 bidirectional input/output ports (X+, X-, Y+, Y- and PME) of 10 flits capacity. Routers implement round-robin arbitration, XY-routing, credit-based flow control, and wormhole switching (see Figure 2.4a and Figure 2.4b). Flit traversal latency in a no-contention scenario is 1 cycle to traverse the router and 1 cycle to traverse the link between routers. Cores are connected to each router and send requests to different memory modules attached to boundary routers, which serve one request per cycle.

We called this architecture Tile-based Manycore (TILED MC), and we mainly focused on working with a similar setup to the one provided by the ARM CoreLink CMN600 [25]. Since our purpose in the EPI Automotive Stream was more focused on the NoC contention analysis for real-time ES, NoC specification was enough for our research. We use this reference architecture when later simulating our proposals in our cycle-accurate simulators in Chapters 4 to 7.

3.3 Tools and Simulators

We used different tools to ease the implementation and testing of our proposals. While some of them have been implemented in the scope of some Thesis proposals as the contention breakdown parser (see Chapter 6), others have been obtained from the CAOS group at BSC and from the master studies in the Facultat d'Informàtica de Barcelona de la Universitat Politècnica de Catalunya (FIB-UPC).

3.3.1 Architectural Simulators

The architectural simulators used in this Thesis are: SoCLiB [126] and gNoCsim [127].

- SoCLiB [126] is an open virtual Cycle Accurate Bit-Accurate (CABA) timing simulator and functional emulator used to model multiprocessors. In our case, SoCLiB has been modified to model NGMP SoC and other complex NoC-based architectures. The version used of the tool in this Thesis is a modified version of the simulator that has been assessed to accurately simulate real hardware with discrepancies of 1% on average and 3% at most [128].
- gNoCsim [127] is an event cycle-accurate simulator developed in C and C++. This simulator models the flits flowing along the NoC links, router stages and collects statistics. gNoCsim has 2 working modes: In master mode, the simulator self-generates synthetic traffic that can be specified by the user at core level (e.g. IR per core, packets/flits size,...). In slave mode, the simulator works by simulating the NoC part of the system and is plugged into a core simulator such as SoCLiB.

In this Thesis, we have used SoCLiB to model the NGMP bus-based architecture in itself, but also to evaluate the impact of our proposals when using real traffic (e.i. real benchmark simulated by SoCLIB) attached to gNoCsim simulating a 2DMesh NoC interconnected system (e.g. gNoCsim in slave mode) when modeling the TILED MC described in the previous section. As there were neither low details of the EPI GPP nor a system emulator or simulator, we started working with LEON4 as cores (SoCLIB) but interconnected in a 2DMesh wNoC (gNoCsim). Moreover, we have also used gNoCsim in master mode to reach some NoC traffic cases (synthetic traffic) and also SoCLiB attached with gNoCsim simulating well-known benchmarks and self-created stressing benchmarks both described in Section 3.4.

Table 3.1: Summary of the modeled reference architectures.

	Parameters	LEON4	NGMP	TILED MC
Core	Stages	7	7	7
	Core Count	1	4	4 to 36
	Write Buffer	2	2	2
L1 I/D Cache	Line Size	32 Bytes		16 Bytes
	Set Number	128 Sets		512 Sets
	Associativity	4 Way		
	Total Size	16 KiB		32 KiB
	Write Policy	Write-Through		
	Placement Policy	Modulo		
	Replacement Policy	LRU		
L2 Cache	Line Size	32 Bytes		64 Bytes
	Set Number	2048 Sets		1024 Sets
	Associativity	4 Way		4 to 36 Way
	Total Size	256 KiB		256 KiB to 2,3 MB
	Write Policy	Copy-Back		
	Placement Policy	Modulo Partitioned		
	Replacement Policy	LRU		
Bus	Arbitration Policy	Round-Robin, Random		-
	Width	128 bits		-
2D Mesh	Sizes	-		2x2 to 6x6
	Router pipeline	-		Atomic
	Arbitration Policy	-		Round-Robin, Random, Weighted
	Routing Policy	-		XY, YX
	Link Width	-		144 bits
	Packet Size	-		128 bits
	Packetization	-		Yes 1 to 6 flits
	Memory Controllers	-		1 to 4

3.3.2 ILP Tools

As we described in the goals and contributions in Section 1.3, one of the targets of the Thesis has been the optimization of the WCET calculus obtaining tighter and more reliable estimates. After realizing in Chapter 4 that WCET estimates in NoC-based systems are a multidimensional problem, in Chapter 5 we have used the IBM

ILOG CPLEX Tool inside a more complex workflow described in Section 5.3. CPLEX is a tool that accelerates the development and deployment of decision optimization models using mathematical and constraint programming. In this Thesis, we have used the mixed ILP part of the tool to minimize the WCET estimates obtained when optimizing multiple parameters of the NoC setup. Despite IBM CPLEX being a licensed tool, it exists other alternatives and open source solvers (e.g. Coin-Or, GLPK or LP Solver).

3.3.3 Contention parser

With the goal of breaking down contention NoC cycles in Chapter 6 we created a tool that given an event trace from a simulator or processor it provides per cycle contention breakdown a task suffers from their co-runner tasks in a 2Dmesh with deterministic routing, router arbitration and task allocation. More in detail, the tool is a parser written in C and C++ that filters the execution trace and assigns the NoC contention suffered by a TuA chosen by the user to their co-runner tasks that causes this packet stall. We develop this tool in Chapter 6, and we used and extend it in Chapter 7.

3.3.4 Computational Intrastructure

Most of the experiments done during this Thesis have been launched in the Sert UPC Cluster with more than 130 nodes Intel Xeon E5-2630L, with 64 to 128 GB of RAM and 1TB HDD SATA-3 (see Research UPC Cluster [129] for more details). The less computationally demanding experiments have been directly obtained in a laptop with an Intel i7-8650U processor with 16GB of DRAM.

3.4 Benchmarks and Applications

We used different benchmarks and applications representative in the CRTES domain. On the one hand, we use the MediaBench [130] and the EEMBC Autobench [131] well-known suites but on the other hand, we have also created our own Benchmarks with more heavy memory-loaded scenarios. So as to recreate other scenarios, especially in the interconnection part without depending on a core architecture, we have also performed synthetic traffic injection experiments that are also detailed in this section. For the real core program execution experiments we have used:

- MediaBench [130] is a set of applications and algorithms that include communications and multimedia functions increasingly relevant for many CRTES domains as autonomous navigation and driving systems. MediaBench contains 20 applications (see Table 3.3) culled from available image processing, communications and DSP applications. Figure 3.3 shows a characterization of the memory load of the Mediabench suite (M benchmarks).

- EEMBC AutoBench [131] is a suite of benchmarks that allow measuring the performance of microprocessors and microcontrollers in automotive, industrial, and general-purpose applications. The benchmark suite is composed of 16 benchmark kernels (see Table 3.3) that include generic workload tests, basic automotive algorithms and signal processing algorithms. Each benchmark consists of a number of calls to different functions that are located in the main loop, and the user can configure the number of iterations that every benchmark loops (or use the default number provided by the suite). Figure 3.3 shows a characterization of the memory load of the EEMBC AutoBench suite (E benchmarks).

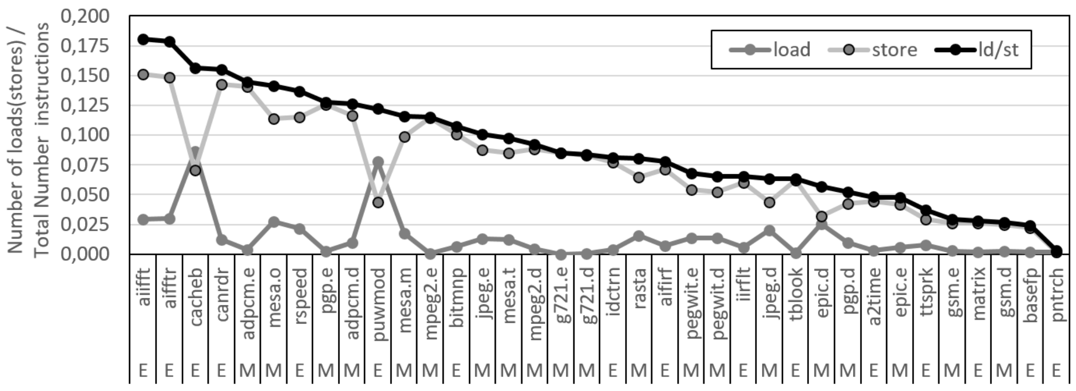


Figure 3.3: Percentage of loads/stores in each benchmark. *E* and *M* stands for EEMBC Auto and MediaBench, respectively.

- *A − H* Self-generated benchmarks: these 8 workloads (A, B, C, D, E, F, G, and H) generate different percentages of load (LD), store (ST) operations. Whereas EEMBC Automotive and Mediabench have between 0 and 18% of load/store memory operations in their benchmarks, our self-generated workloads have between 5 and 50% of load/store operations (see Table 3.2). Still, we have also created other synthetic benchmarks that are 100% load/store operations so as to evaluate saturation cases (i.e. higher packets injection than ejection in the NoC) or for validating the correct implementation of our router.

Table 3.2: Self-generated Benchmarks.

Benchmarks	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
% Local Op	80	50	50	60	95	87.5	87.5	90
% LD Op	10	10	40	20	2.5	2.5	10	5
% ST Op	10	40	10	20	2.5	10	2.5	5

Moreover, with the aim of recreating high contention scenarios in the NoC interconnection, we have also executed experiments using synthetic traffic (gNoCsim working in slave mode as explained in Section 3.3). This working mode has allowed us to tune the traffic injection at synthetic core level recreating worst-case scenarios and avoiding pathological packets' alignment.

Table 3.3: Summary of benchmarks used for evaluation.

Mediabench benchmarks	
adpcm.d	Adaptive differential pulse code modulation, decode
adpcm.e	Adaptive differential pulse code modulation, encode
epic.d	Experimental image compression utility, decode
epic.e	Experimental image compression utility, encode
g721.d	Voice compression G721, decode
g721.e	Voice compression G721, encode
gsm.d	GSM 6.10 standard for full-rate speech transcoding, decode
gsm.e	GSM 6.10 standard for full-rate speech transcoding, encode
jpeg.d	Image lossy compression JPEG, decode
jpeg.e	Image lossy compression JPEG, encode
mesa.m	3D graphics library clone of OpenGL, mipmap application
mesa.o	3D graphics library clone of OpenGL, osdemo application
mesa.t	3D graphics library clone of OpenGL, texgen application
mpeg2.d	Standard for high-quality digital video transmission, decode
mpeg2.e[1]	Standard for high-quality digital video transmission, encode
pegwit.d	Public key encryption and authentication, decode
pegwit.e	Public key encryption and authentication, encode
pgp.d	Security algorithm PGP, decode
pgp.e	Security algorithm PGP, encode
rasta	Speech recognition
EEMBC Autobench	
a2time	Angle to Time Conversion
aifftr	Fast Fourier Transform (FFT)
aifirf	Finite Impulse Response (FIR) Filter
aiifft	Inverse Fast Fourier Transform (iFFT)
basefp	Basic Integer and Floating-Point
bitmnp	Bit manipulation
cacheb	Cache buster
canrdr	CAN Remote Data Request
idctrn	Inverse Discrete Cosine Transform (iDCT)
iirflt	Infinite Impulse Response (IIR) Filter
matrix	Matrix Arithmetic
pntrch	Pointer Chasing
puwmod	Pulse Width Modulation (PWM)
rspeed	Road Speed Calculation
tblook	Table Lookup and Interpolation
ttsprk	Tooth to Spark

[1] mpeg2.e benchmark has been excluded in some contributions' evaluation due to issues executing it in our reference platform.

Part II

Contention Aware Optimizations in Verification stage

Chapter 4

Mesh NoC Flow Balancing Bandwidth Optimization

4.1 Introduction

The increasing performance needs in CRTES can only be satisfied with the use of high-performance manycore processors. NoC-connected manycores have already been deployed in high-performance (mainstream) processors [5, 20], and have been shown to allow, under some restrictions (e.g. deterministic routing), the derivation of tasks' WCET estimates as needed for CRTES [51, 62]. However, WCET estimates vary drastically across cores in mesh NoCs due to the different BW effectively allocated to each core and, to a lower extent, the diverse latencies caused by non-homogeneous distances from cores to their target node (e.g. the one where main memory is attached) [64]. Local homogeneous BW allocation (e.g. as provided by fair policies like round-robin) might cause unwanted heterogeneous global BW distribution across cores (i.e. cores further away from memory have lower BW than those closer to it). As a result, tasks running in cores with lower allocated BW can be severely penalized. Parallel applications, despite being less widespread than in the mainstream market, are now considered in CRTES for computing intensive functions related to AD and unmanned navigation. For those applications, unwanted BW imbalance may result in poor WCET estimates for the whole application since all threads may have to synchronize with the slowest one (i.e. normally the thread that runs in the farthest core from memory).

Wmeshes, widely used in high-performance routers for off-chip wormhole networks [108], have been proposed as a solution in CRTES to allow heterogeneous BW allocation across ports to homogenize overall BW across cores [64]. Unfortunately, wmeshes suffer from a key limitation: *efficiently sized resources in NoCs may create bubbles in some links, so that they are unable to send packets steadily. If the requirements on BW allocation impose highly unbalanced BW in some routers, bubbles may prevent from reaching the desired BW allocation in those routers, leading to globally unbalanced BW allocations.*

In this chapter, we tackle this challenge by proposing EOmesh, a new weighted mesh design that effectively achieves near-optimal (homogeneous) BW allocation across cores. EOmesh builds on the observation that BW allocation can only be practically achieved by combining the use of weights in the arbiters and balancing the number of flows served at each output port. In particular, our contributions are as follows:

1. We analyze the behavior of wmeshes, showing that, by construction, they cannot achieve homogeneous BW allocation across cores as long as one port needs to serve above 2/3 of the requests, which necessarily occurs for square meshes with 4x4 nodes and beyond.
2. We provide a smart combination of time-predictable routing policies (XY and YX) that allows reducing the imbalance across ports so that bubbles do not prevent from reaching the desired homogeneous BW allocation. The combination of multiple routing policies with statically allocated VCs makes the mesh NoC deadlock-free.
3. We use appropriate weight allocations that achieve homogeneous BW allocation across cores, in combination with the new routing scheme. Hence, fairness across cores in terms of BW is achieved despite bubbles in the NoC.

4.2 Modeling a Weighted Mesh

WCET estimation in manycores needs bounding access times to shared hardware resources [132]. In the case of NoCs, this translates into having a bounded WCD such that every request sent to the NoC has a bounded service (traversal) time at analysis time.

4.2.1 Baseline NoC

We consider a *NxM* mesh NoC as the one described in Section 2.2.2 in which each node can be identified using (x,y) coordinates, see Figure 4.1. The router located at coordinates (x,y) is referred to as $R(x, y)$, see Table 4.1 for the definitions used in this chapter. Each node comprises the router that communicates the node to the mesh and a *PME* (Processor/Memory element). The PME can be either a processor core, a cache memory, main memory, I/O, etc. In the network several traffic flows may exist. A traffic-flow (F_i) is a packet stream that traverses the same H-node route from a source to a destination node and requires the same grade of service along the path.

For the characterization we use deterministic XY routing but any other deterministic routing can be used too. Deterministic routing allows identifying routers in a given path as R^j where j is the hop number of the path (e.g. R^1 is the source node). With XY routing, packets are forced to use the X dimension first. In the X dimension the position of the target node with respect to the source node determines whether to go right (X+) or left (X-) direction. The same approach is used for the Y

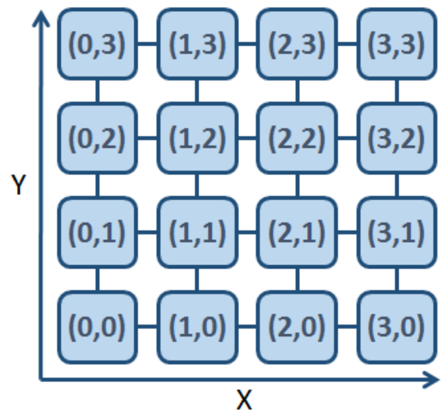


Figure 4.1: Router coordinates of a 4x4 2Dmesh.

Table 4.1: Definitions used in this chapter.

Acronym	Description
$R(x, y)$	Node with coordinates (x,y) in the NoC
F_i	Flow i
P_i^j	Number of requests that might contend for the same R^j output port as F_i under the worst-case contention scenario
ER_i^j	Rate at which flits of flow F_i can be ejected from router R^j
D_i^j	Maximum time that a packet of F_i requires to go from the input port of R^j to its destination node
$fx\{i\}$	Index of the flow causing the worst possible blocking on F_i
PER^{wc}	Propagated worst-case ejection rate
$PER_i(R^j)$	Propagated worst-case ejection rate for flow F_i at router R_j
L	Maximum packet size
I_{dir}	Number of flows traversing the dir input port
O_{dir}	Number of flows traversing the dir output port
N_{ports}	Number of ports per router

dimension. Once packets are routed using the Y dimension they cannot be forwarded to the X dimension. Note that the opposite port is represented as $\bar{Y}$ and $\bar{X}$. For instance, the opposite port of $Y+$ is $Y-$. Routing restrictions help determining the exact number of requests (P_i^j) that might contend at router R^j for the same output port as F_i in the worst-case situation. For instance, P_i^j values for a mesh with XY routing and assuming all-to-all communication are determined as follows:

$$P_i^j = \begin{cases} 2 & if\ destination\ is\ X+\ or\ X- \\ 4 & if\ destination\ is\ Y+,\ Y-\ or\ PME \end{cases}$$

4.2.2 Deriving Worst-Contention Delay

In this section, we provide expressions to compute WCD bounds that are also suitable for NoCs using weighted round-robin arbitration. Expressions given in this section are based on the concept of worst-case ejection rate (ER_i^j). We define ER_i^j as the rate at which flits of flow F_i can be ejected from router R^j to the corresponding port when the next router (R^{j+1}) is accepting incoming packets (i.e. it is not stalling R^j packet transmission). We also extend the concept of worst-case network ejection rate to model the rate at which flits can be ejected from a given router port when the network is fully congested. To do so, we define propagated worst-case ejection rate PER^{wc} as the minimum rate at which flits of F_i can be ejected from R^j in the worst-case situation. ER_i^j values can be computed by considering the maximum number of flows P_i^j contending at R_i^j for the same output port as F_i, see Equation 4.1.

$$ER_i^j = \frac{1}{P_i^j} \tag{4.1}$$

Then, $PER_i(R^j)$ is computed by multiplying ER_i^j values from the current router R_i^j to the target router R_i^H as presented next:

$$PER_i^j = \prod_{k=j}^{H} \frac{1}{P_i^k} \tag{4.2}$$

Let D_i^j be the time that a packet of flow F_i requires to go from the input port of R^j to its destination node. D_i^j can be computed recursively by considering the time required to reach R^{j+1} as $1/PER_{fx\{i\}}^j$ plus the time required to reach its destination once at R^{j+1}. $fx\{i\}$ represents the index of the flow that causes the worst possible blocking in F_i. Note that a $F_{fx\{i\}}$ packet stalled in a subsequent router of the path followed by F_i might cause F_i to suffer worst contention than one following exactly the same path. In the same way, $PER_{fx\{i\}}^j$ represents the worst ejection rate for F_i packets. To determine the flow causing the worst contention, PER values for all routers and all flows are computed in advance, and for any particular flow and router we choose the worst $PER_{fx\{i\}}^j$. Equation 4.3 shows the recursive definition of D_i^j.

$$D_i^j = \frac{1}{PER_{fx\{i\}}^j} + D_i^{j+1} \tag{4.3}$$

The WCD for flow F_i, given by D_i^1, is the time required to reach its destination ($j = H$) from the source node.

4.2.3 Computing WCD with Weighted arbitration

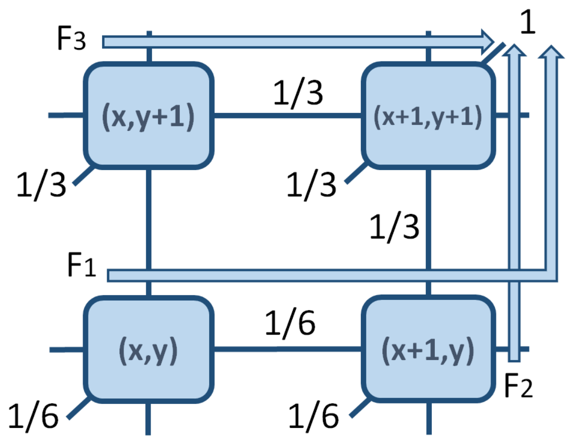


Figure 4.2: 2x2 2Dmesh unfair bandwidth allocation using round-robin arbitration.

We illustrate how to compute WCD using Equations 4.1, 4.2, 4.3 considering round-robin arbitration in the example presented in Figure 4.2. We aim at computing F_1 WCD, i.e. the WCD of packets with source node in (x,y) router and destination in (x+1,y+1). First, we compute PER_i^j as the product of the ER_i^j coefficients (shown in brackets in Figure 4.2) of all the routers that F_i $(i = 1)$ traverses. Later, we start from the last hop $(j = 3)$ and compute all D_i^j values;

$$D_1^3 = \frac{1}{1/3} = 3 \qquad D_1^2 = \frac{1}{1/6} + D_1^3 = 9 \qquad D_1^1 = \frac{1}{1/6} + D_1^2 = 15$$

Table 4.2: WCD values for L-flit packets.

	Round-Robin				Weighted			
	F_1	F_2	F_3	F_4	F_1	F_2	F_3	F_4
D_i^3	3L	-	-	-	2L	-	-	-
D_i^2	9L	3L	3L	-	6L	2L	4L	-
$D_i^1(WCD)$	15L	9L	6L	3L	10L	6L	8L	4L

Figure 4.2 shows WCD values for the 2x2 NoC for both round-robin and weighted round-robin arbitration. In particular, we compute the WCD of F_1, F_2, F_3 and F_4. WCD for F_3 is given by $D_3^1 = D(x, y+1)$. Table 4.2 shows the D_i^j and WCD values for F_1, F_2, F_3 and F_4 for both round-robin and weighted arbitrations. F_4 comes directly from one of the router (x+1,y+1) ports. As shown in the table, weighted arbitration makes the WCD of the packets to be reduced for those flows that are located in the farthest positions, and this comes at the expense of penalizing the WCD of the nodes closer to the destination. It is also important to mention that despite weighted arbitration has the potential to fairly share the BW, the WCD of each core is not fully equalized.

4.2.4 Limitations of Weighted Meshes

Wmeshes have been proposed jointly with XY routing[1]. Figure 4.3(a) shows an example of XY routing in which for a typical 4x4 mesh with the target node (e.g. a shared cache or main memory) connected to a port in a router in one corner (e.g. (3,0)) and all nodes attempting to send packets to that node.

As shown, under XY routing the router at the corner has to arbitrate among three input ports with highly unbalanced traffic: (1) the X+ port receives requests from 3 out of 16 cores, (2) the local port (not shown in the picture) receives requests from 1 out of 16 cores, and (3) the Y+ port receives requests from the remaining 12 cores. Overall, the Y+ port needs to absorb 75% of the traffic if BW across cores is homogeneously balanced.

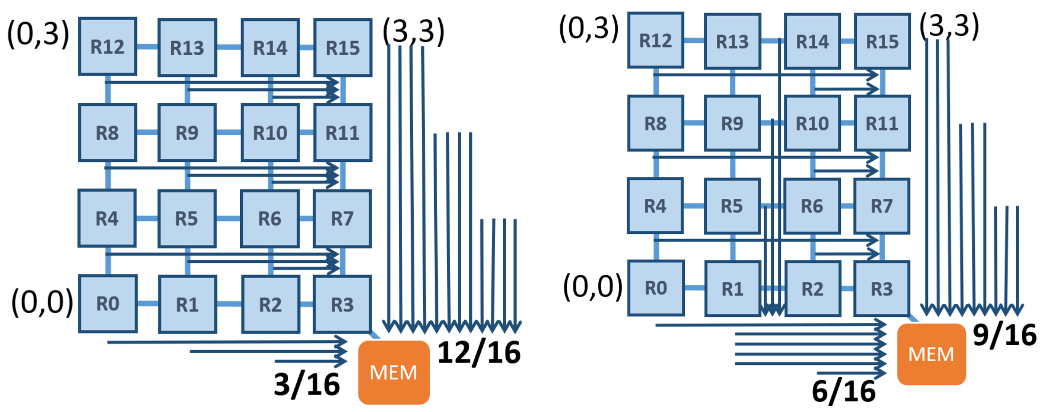
Let us consider an arbitration window implemented as an extended round-robin vector with 16 entries, so that BW can be weighted as needed for homogeneous BW allocation. The arbitration window for the router in the corner is composed of 16 elements (1 PME, 3 X+, 12 Y+). The only way to arrange those port grants in a window so that, inside the window and across windows, the maximum number of consecutive arbitrations granted to the same port is minimized is using the following pattern:
$PME, Y+, Y+, Y+, X+, Y+, Y+, Y+, X+, Y+, Y+, Y+, X+, Y+, Y+, Y+$.
Note that swapping PME and $X+$ grants or moving $Y+$ grants to the beginning would lead to equivalent patterns with $Y+, Y+, Y+$ sequences interleaved with other (individual) grants over time.

The effectiveness of the weighted arbitration is decreased in the presence of bubbles. In regular wormhole mesh NoCs designs, bubbles can occur due to local and global control-flow effects.

Canonical routers in a 2D mesh are pipelined into several stages. First, the incoming packet is stored in the corresponding input buffer. Then, routing and switching occur in one or more stages, and finally the packet is sent through the link. However, since no packet-loss is allowed in wormhole, before the packet is sent to the next router, the stall/go signal coming from the next router is checked to ensure there is enough buffer space to store the packet. This stall/go signal is used to ensure the link-level (or local) flow control and requires C_f cycles to travel from the destination router (R_{x+1}) to the current one (R_x), and the round trip time (RTT) is equal to $2 \times C_f$. The RTT determines the amount of buffering required at the input buffers to avoid having bubbles in the transmission.

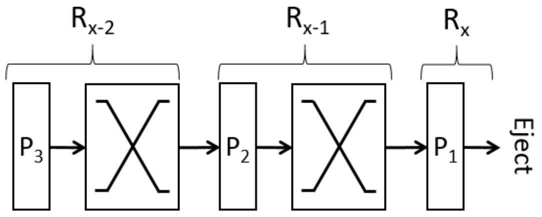
Let us consider the example in Figure 4.4, where we show the main stages of three consecutive routers (R_x, R_{x-1}, R_{x-2}), being packets ejected through router R_x. Let us also consider three packets (P_1, P_2, P_3), and an initial state where P_1 is in an input buffer in R_x, P_2 in an input buffer of R_{x-1}, and P_3 in an input buffer of R_{x-2}. If R_x did not allow these packets to make any progress in the previous cycle, but R_{x-1} and R_{x-2} allowed them to progress, P_1 could not be switched and ejected. Instead, P_2 and P_3 were switched but could not be transmitted to the following router due to

[1]Note that the reasoning that follows applies identically with YX routing.



(a) XY mesh (b) YX XY mesh

Figure 4.3: Routes to router (3,0).



	0	1	2	3	4
P_1	B(x)	Eject			
P_2	sw(x-1)	B(x)	Eject		
P_3	sw(x-2)	B(x-1)	sw(x-1)	B(x)	Eject

Figure 4.4: Example of bubbles when sending continuously packets from one port.

backpressure of the input buffers. This is illustrated in the chronogram in Figure 4.4 as the state in cycle 0. Eventually, in cycle 1 P_1 is ejected, P_2 reaches the input buffer of R_x, and P_3 the input buffer of R_{x-1}. In cycle 2 P_2 is ejected, and P_3 is switched in R_{x-1}. In cycle 3, P_3 reaches the input buffer of R_x, but cannot be ejected until cycle 4. Overall, R_x can eject up to 2 packets consecutively coming from the same input port. Hence, a weighted mesh where more than 2 packets need to be transmitted in consecutive cycles cannot serve those packets at a sufficient speed and causes some imbalance.

In this example, if buffers are large enough potentially all packets can reach R_x faster. However, buffers are the most expensive resource in routers, so their size is kept as low as reasonably possible.

Global Effects. In current CRTES, packets in the NoC usually correspond to memory transactions going from the cores to the memory devices and the other way around. The maximum speed at which cores issue requests to memory is determined by the number of cycles the requests going to memory need to be processed. In the context of a NoCs, this time is usually in the order tens of cycles and avoids having always requests to be served by the router during operation. However, WCD estimation cannot make any assumption on the actual load, so the worst congestion must be assumed for WCET estimation.

Thus, even allocating BW as in [64] (that theoretically allows achieving homogeneous arbitration using weights that consider the amount of flows using each input port), global fairness cannot be achieved in practice if the amount of traffic flows traversing each port is not balanced as well.

4.3 EOMESH: A Fair Weighted Mesh

As we have seen in the previous section, although regular weighted NoCs with XY routing are theoretically able to achieve a perfect balancing of the available BW, this balancing is not achieved in reality due to the presence of bubbles. To solve this problem, we propose a new mesh design intended to achieve near-optimal weighted arbitration. The idea behind Even/Odd mesh (EOmesh) is that the even allocation of BW cannot be practically achieved by simply playing with the arbitration, but it also requires balancing the amount of flows served at each output port. EOmesh combines both concepts, and it balances both (1) the BW each flow is assigned using a weighted arbitration and (2) the amount of flows each port has to serve. In the following sections, we describe how EOmesh implements these concepts.

4.3.1 Combined Flow Balancing & XY-YX routing

In order to balance the amount of flows each output port serves, we change the routing algorithm. With XY routing, the default routing policy in the wmesh proposed in [85], ports in the Y direction serve high number of flows that makes not possible to achieve a fair distribution of the BW due to the presence of bubbles. To avoid the imbalance in number of flows that traverse each given port when using XY routing, we propose a new routing algorithm that combines both XY and YX. More in detail, this algorithm uses XY for packets originated at the source nodes with an even identifier, and YX for the packets from source nodes with an odd identifier. The proposed mechanism is illustrated in Figure 4.3(b): combining XY and YX algorithms for even and odd sources allow improving the balancing in the number of flows each port has to serve. For the particular example in the plot, we see how the output port arbitration of the memory controller attached to node 3 serves 6, 9, and 1 flows through the X, Y, and PME input ports while in the case of the XY routing (Figure 4.3) the distribution of flows is 3, 12, 1, for the X, Y, and PME input ports.

VC allocation. The main reasons why XY routing is heavily utilized are its simplicity (low complexity) and its deadlock freedom properties. However, as we have analyzed previously, XY routing does not balance traffic flows efficiently. With EOmesh routing approach, the amount of flows traversing each port is more balanced. However, the combined use of XY and YX routings, allows the creation of deadlock situations for specific communication flows. In that respect, in order to avoid deadlock situations, EOmesh assigns a specific VC to each of the two routing policies employed (XY and YX). Performing a static VC allocation to isolate flows from the different routing policies XY and YX also allows reducing the worst contention that the different flows can have since the amount of contender packets that each flow finds in each hop is reduced.

Given that dynamic VC allocation policies have a very negative impact in WCD – since the number of potential contenders is increased in a router with VCs [51] – the static allocation of VCs performed by EOmesh does not degrade the best guaranteed performance achievable by the network.

4.3.2 Adapting arbitration weights

Weights in the weighted round-robin mesh design proposed in [85] can be computed using the following expression:

$$w(I_{dir}, O_{dir}) = I_{dir}/O_{dir} \tag{4.4}$$

where I_{dir} represents the number of communication flows traversing the dir_i input port of a given router being dir any of the possible mesh router port directions. Similarly, O_{dir} is the number of flows traversing the dir output port of the same router.

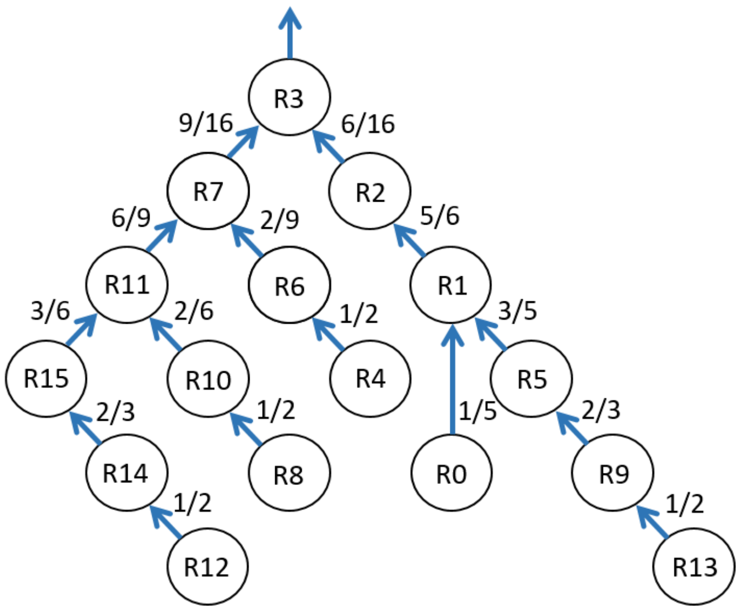


Figure 4.5: EOmesh arbitration weights to access a shared resource attached to router (3,0). Router internal port (PME) weights are not shown in the picture.

Given a fixed number of communication flows, the actual flows traversing the input/output ports of each router can be determined considering the particular route used by each flow. Figure 4.5 shows the EOmesh weights required to arbitrate the flows originated at each router that target the shared resource attached to Router (3,0).

4.4 Implementation

The baseline wmesh can be implemented in two main different ways: (1) as a programmable NoC or (2) as a hardwired NoC.

4.4.1 Programmable NoCs

In a programmable NoC, routing and arbitration decisions can be interfaced and modified by means of software commands, thus with no hardware modification.

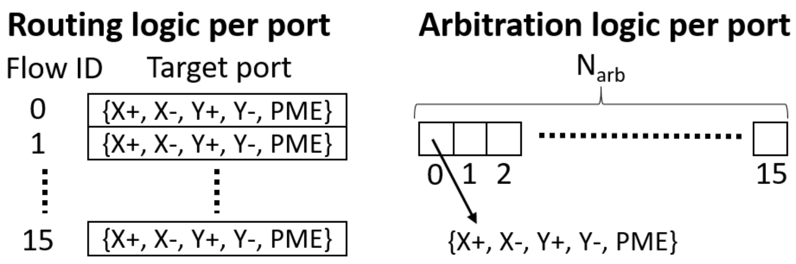


Figure 4.6: Routing and arbitration implementation of a programmable NoC.

Programmable routing. Making routing programmable requires, for instance, the use of routing tables in each port. Hence, for each port of each router, we need a table with as many entries as potential different flows (e.g. 16 entries for a 4x4 mesh) where each entry contains the identifier of a destination port, which can be $\{X+, X-, Y+, Y-, PME\}$. Although 5 different values are possible across all ports, for a given port only 4 of them are possible since a port cannot be its own target. For instance, valid values for the $X+$ port are $\{X-, Y+, Y-, PME\}$. Thus, this table needs 2-bit entries to encode the destination port for each flow in each input port. Overall, given a NxM mesh, the (distributed) storage required to make routing programmable is:

$$RoutCost = (NxM) \cdot N_{ports} \cdot (NxM) \cdot bits/entry \tag{4.5}$$

Where the first NxM factor is the number of routers, N_{ports} stands for the number of ports per router (up to 5 ignoring the fact that routers at the boundaries do not have all ports), the second NxM factor corresponds to the number of entries per routing table, and the last factor, $bits/entry$ is 2 as indicated before. For instance, in a 4x4 mesh, routing tables require less than $16 \cdot 5 \cdot 16 \cdot 2 = 2560$ bits, so 320 bytes, which is a rather small cost. This design is sketched in Figure 4.6 (left).

Programmable arbitration. Two main alternative implementations can be used to have programmable arbitration. One of them requires an arbitration window per port and per router, with an arbitrary number of entries N_{arb}, and 2 bits per entry indicating the input port that is granted access to the particular output port. N_{arb} must be sufficiently large so that $1/N_{arb}$ provides sufficient granularity to allocate weights as needed. Moreover, N_{arb} may change across ports and routers. For the sake of illustration, we assume that arbitration windows have NxM entries. Then, each arbiter also needs a $log_2 N_{arb}$ counter pointing to the next entry in the window along with an adder for that counter. Alternatively, one could use shift registers with wrap-up for the window and always use the value at a given position (e.g. first position) to determine what port is granted access next.

In this work, we build on the solution using the counter. Hence, the storage cost for a programmable arbitration is:

$$ArbCost = (NxM) \cdot N_{ports} \cdot ((NxM) \cdot bits/entry + log_2(NxM)) \quad (4.6)$$

For instance, in a 4x4 mesh, arbitration would require $16 \cdot 5 \cdot (16 \cdot 2 + 4) = 2880$ bits, so 360 bytes only. This design is sketched in Figure 4.6 (right).

A second alternative implementation of the arbitration can be built upon counters, where 4 counters are set per port, so that each one tracks how many times a given input port must be granted access to a given output port in each window. In general, this approach allows a finer-grain allocation of weights, but does not allow controlling with precision the order in which grants are given. For instance, it may grant access to the port with the highest count, or in a round-robin fashion across ports with non-zero counters.

4.4.2 Hardwired NoCs

Some NoC implementations favor efficiency in front of flexibility, and routing and arbitration choices are hardwired. Adapting such a NoC to implement the EOmesh would require, at most, duplicating (simple) routing logic in some routers to implement XY and YX policies for different flows. In practice, since a packet can be forwarded in any direction depending on its destination at a given router, the practical cost of an additional check will have a lower cost.

Regarding arbitration, using different weights for a wmesh would require hardwiring different choices in the arbitration windows, thus not increasing hardware costs.

Overall, hardware modifications would have limited impact on the overall cost of the NoC, which is mostly dominated by the buffering required at input ports.

4.5 Evaluation

4.5.1 Methodology

We evaluate EOmesh on a cycle-accurate simulator that is an enhanced version of SoCLib [126] that we integrated with gNoCsim [127] simulator to simulate a TILED MC using a 2D mesh NoC [6, 20] (described in detail in Sections 3.2 and 3.3).

We evaluate EOmesh on meshes with 16 cores (4x4) and 36 cores (6x6) to assess the scalability of our approach.

In our manycore, load (and write-miss) requests comprise four-flit messages from the core to memory. Given that cache line size is 64-bytes and we need 16-bits for control data (512+16 bits), memory answers with 4-flit messages over 132-bit wide links. Evicted line requests require a 4-flit message and a one-flit answer. In our proposed mesh design, packetization [51], shown to minimize WCD, adds control data to each of the flits, therefore requiring an extra flit, so 5 instead of 4 (512+5*16 bits over a 132-bit wide channel), leading to 25% overhead.

For each setup, usually matching one result chart, WCET estimates are shown normalized to the maximum WCET to those results obtained with the baseline wmesh. Hence, normalized WCET (nWCET) estimates below 1 are better than those provided by the default wmesh and vice versa.

$$nWCET = \frac{WCET_{EOmesh}}{max(WCET_{weighted})} \tag{4.7}$$

So as to evaluate EOmesh when speeding up the guaranteed execution time (i.e. reduce the WCET), we evaluate the WCD and WCET evolution in different scenarios: computing analytical maximum load, running independent apps, running parallel apps or heterogeneous apps. As the evaluated apps we will use the $A - H$ self-generated benchmark suit described in Section 3.4 (see LD/ST percentages in Table 3.2).

- Analytical maximum load. WCD bounds are computed for all cores. This is the most stressful scenario for the NoC, where requests are assumed to contend in the worst possible way with other cores' requests.
- Independent apps. Benchmarks A-D, correspond to high-demanding benchmarks in terms of BW, whereas E-H benchmarks are their low-demanding counterparts (25% requests w.r.t. A-D applications).
- Parallel apps. We create 2x2 homogeneous parallel applications where all threads correspond to the same reference benchmark to study their sensitivity to the particular mapping of the application in the mesh (e.g. AAAA or CCCC).
- Heterogeneous applications. We have generated heterogeneous parallel applications comprising several phases and varying the number of tasks in each of the phases. We have generated 8000 directed acyclic graphs (DAG) to simulate different coarse-grain parallel applications as the ones supported by the *Ada* programming language [133]. Figure 4.7 shows a schematic of the DAG template we have used to generate the different applications. For each application DAG tasks are chosen randomly from the workloads in Table 3.2 (e.g. A,CADB,BBB,A with 4 phases).

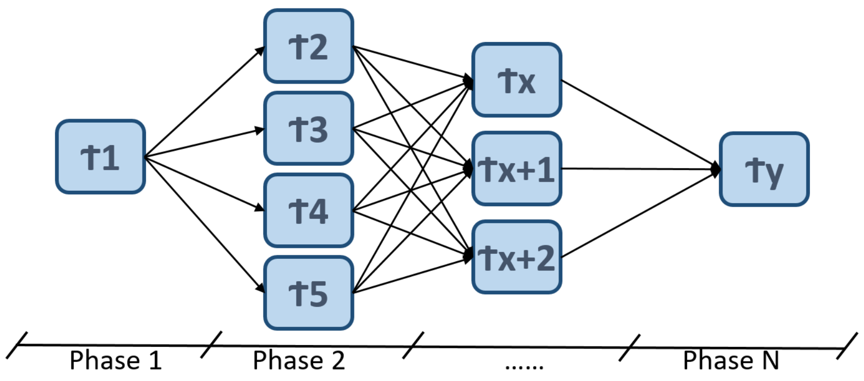


Figure 4.7: DAG schematic of the heterogeneous applications.

4.5.2 Analytical Contention Bounds

First, we compute for the flows allocation shown in Figure 4.3 (where all flows target memory controller in R3, R0 allocates F0, R1 allocates F1 and so one so forth) the WCD bounds across cores, whose maximum determines the worst contention that any core request could experience. As shown in Figure 4.8 for a 4x4 mesh, the EOmesh reduces significantly the WCD (cycles) for those flows (e.g. F4 to F15) with highest WCD values. This is achieved by a better organization of the traffic (combined XY and YX routing policy) and an appropriate weight allocation, which slightly increases the WCD for the fastest nodes (e.g. F0 to F3) to decrease it for the slowest ones (e.g. F4 to F15 and specially in F4, F5, F8, F12 and F13). In particular, the maximum WCD for the original wmesh (F12 and F13) decreases by 14.7% in the EOmesh (F13). Thus, the amount of contention that needs to be added to each request during WCET estimation is significantly reduced.

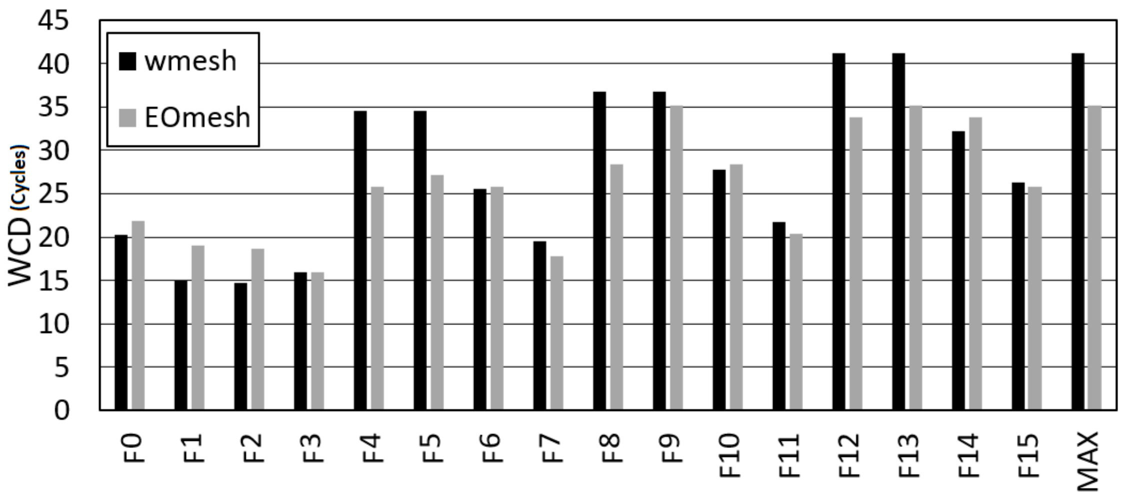


Figure 4.8: WCD bounds for all flows (4x4 mesh nodes).

EOmesh is specifically designed to reduce the WCET of the slowest thread in a parallel application (i.e. is the one that truly impacts the WCET of the entire application). This is achieved by allocating BW so that the maximum WCD (and so the imbalance) in the original wmesh is reduced. That is, the WCD (and hence the WCET) suffered by threads closer to main memory is on-purpose increased to reduce that of threads farther away from main memory, since those determine the WCET of the overall application. In this line, Figure 4.8 should not be read in terms of the number of threads (nodes) for which EOmesh reduces WCD, but instead, on whether EOmesh reduces the WCET of the slowest thread.

4.5.3 Independent (single-threaded) Applications

In order to assess the impact of WCD reduction of the EOmesh w.r.t. the baseline wmesh, we have evaluated it using all single-threaded applications. In particular, we obtain time-composable WCET bounds (thus valid regardless of the tasks running in the other cores) for each application on each of the cores. WCET estimates have been

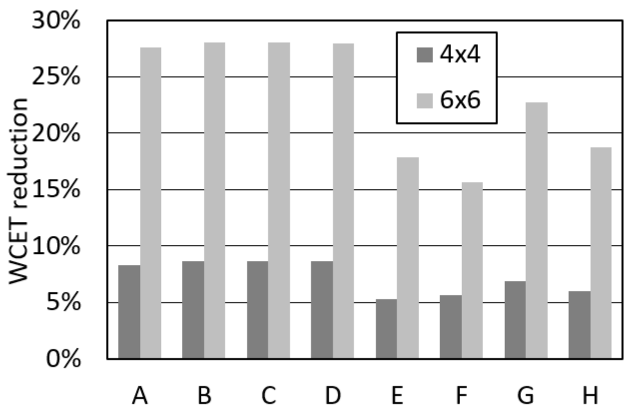


Figure 4.9: WCET estimation reduction.

obtained by considering the execution time in isolation (obtained with measurements in this particular evaluation) and adding the WCD to each request, where the WCD depends on the mesh policies (baseline wmesh or EOmesh) and the particular core where the application is run. Note that other WCET estimation practices applied during unit testing (aka when contender tasks are unknown) would provide different in-isolation execution times, but would account for NoC contention analogously to our work (i.e. considering the worst-case contention per request).

Figure 4.9 shows, for each application, the WCET reduction achieved with the EOmesh. It has been obtained by comparing the maximum WCET across all cores on the baseline wmesh and on the EOmesh . As shown, despite the fraction of operations causing NoC requests decreases from 50% (applications B and C) down to only 5% (application E), the relative WCET reduction in the range 5%-9% and 15%-28% for 4x4 and 6x6 meshes respectively due to the fact that NoC latency is the dominant factor in the WCET time. For completeness, we also show the results on a per-core basis for application A (see Figures 4.10 and 4.11 for 4x4 and 6x6 respectively), as illustrative example of how the WCET varies across cores for the same flow allocation described in Figure 4.3 but targeting $R3$ in 4x4 and $R5$ in 6x6. As shown, in line with WCD results, the EOmesh slightly penalizes the WCET for those cores with lowest in-isolation execution time to decrease appreciably the WCET in the cores with highest in-isolation execution time.

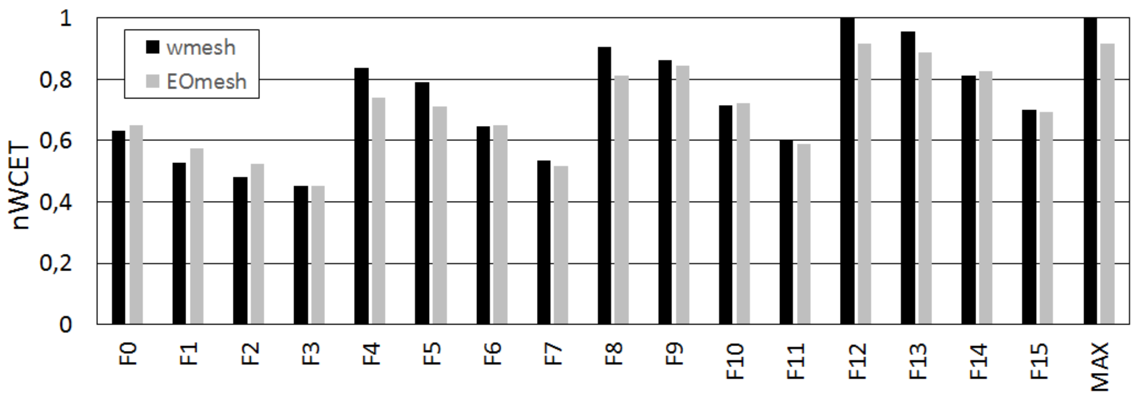


Figure 4.10: Normalized WCET for application A (4x4).

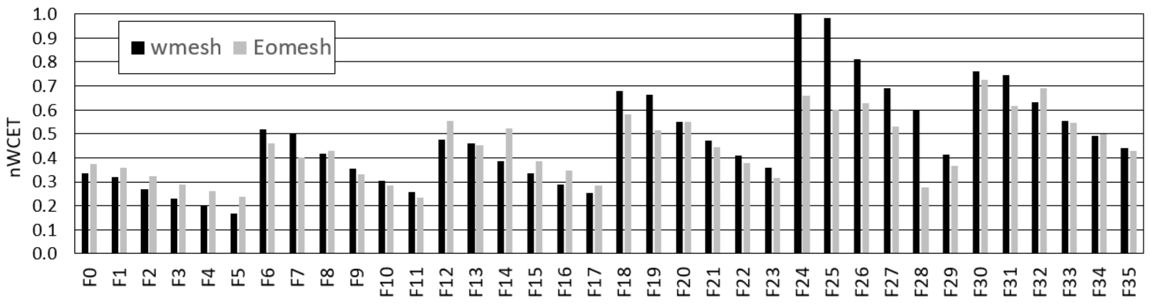


Figure 4.11: Normalized WCET for application A (6x6).

4.5.4 Parallel Applications

We have evaluated the relative WCET of a set of parallel applications building on our reference single-threaded applications on a 4x4 mesh. The first experiment consisted in building 16-thread parallel applications with all threads being of the same type (e.g. 16 applications A). As expected, the WCET was dictated by the worst thread, thus delivering the same results shown in Figure 4.9. Then, we have built 4-thread parallel applications mapping them in 4 different square regions (2x2 cores) in the 4x4 mesh. Each such square region corresponds to the partition of the mesh into 4 square regions, which we identify with *U* (upper), *B* (bottom), *L* (left) and *R* (right). For instance, UR corresponds to the upper-right square, which includes nodes (2,2), (2,3), (3,2) and (3,3).

Results for those 4-thread homogeneous parallel applications are shown in Figure 4.12, again, normalized (for each reference application) w.r.t. the maximum WCET across all parallel applications. As shown, the EOmesh provides significant gains for the UL mapping, which is the one with the highest WCET estimates. Gains for the second worst case (BL) are also noticeable. Results for the third worst case (UR) are slightly worse for the EOmesh. As explained before, the EOmesh decreases the WCD of the slowest nodes at the expense of increasing the WCD for some (fast) nodes. Finally, the best square (BR) experiences almost no change (up to 0.3% WCET variation).

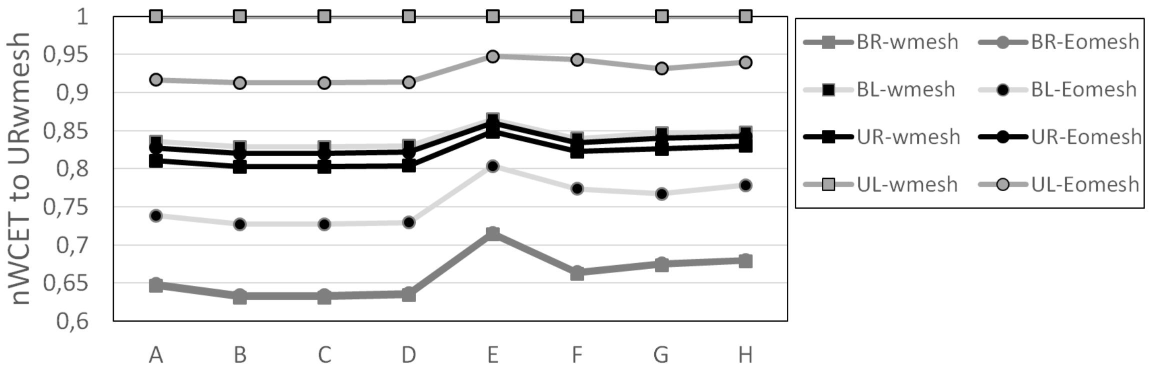


Figure 4.12: nWCET. 4-thread homogeneous apps (4x4 mesh).

Overall, the EOmesh proves to be also beneficial for parallel applications, decreasing the WCET in those regions that lead to highest WCET estimates.

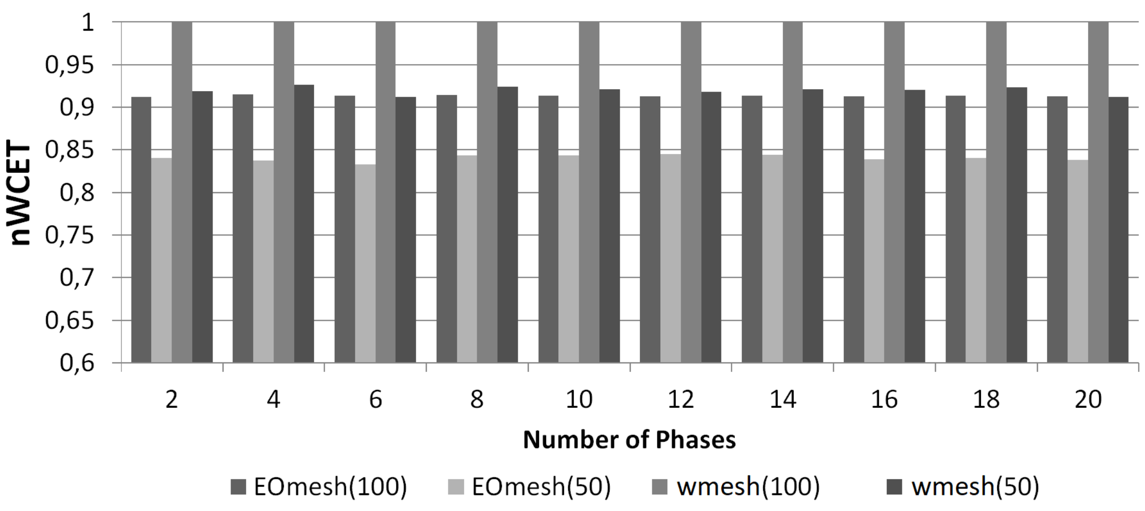


Figure 4.13: nWCET for heterogeneous parallel applications.

Figure 4.13 shows results for DAG-based heterogeneous parallel applications on a 4x4 mesh, whose threads are mapped randomly to cores. We show results for applications with 1 to 20 phases and processor utilization of 50% and 100%. A 100% utilization means that all the 16 cores in the manycore are utilized while with 50% utilization the amount of cores utilized in each of the phases is different but is 50% on average. As shown in the plot, WCET reductions around 8%-9% are obtained regardless of the degree of utilization and the number of phases of the parallel applications. EOmesh effectively homogenizes BW across cores, which ultimately decreases the imbalance across threads in parallel applications. While some threads run in cores with lower BW in the EOmesh, since some others run in cores with low BW in the baseline wmesh allows the EOmesh to speed them up. Hence, by speeding up the threads in the slowest cores (due to their lower BW), EOmesh improves the performance of parallel applications.

4.5.5 Average Performance

One of the main side effects when balancing the behavior of applications – QoS solutions in general – is the loss of throughput, since processor design is tuned to improve average performance. In order to show that our proposed approach does not negatively impact average performance, we have run our reference applications in all cores simultaneously and collected measurements with actual contention. Our results show that, in general, the discrepancies between the baseline wmesh and the EOmesh are very low. For illustration purposes, in Figure 4.14 we show the execution time reduction obtained with the EOmesh in the slowest core in each case. As shown, EOmesh does not cause any degradation in performance but results in some gains (around 1%-2% for 4x4 and 11%-15% for 6x6) due to avoiding bubbles in arbitration, thus removing unnecessary stalls and increasing also average performance.

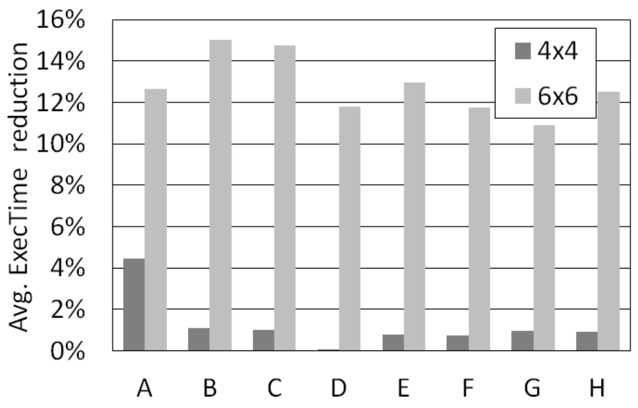


Figure 4.14: Average execution time reduction.

4.6 Conclusions

Wmeshes are an effective solution to homogenize BW allocation across cores, which is of prominent importance for tight WCET estimation in critical real-time systems. However, as shown in Section 4.5, wmeshes fail to provide homogeneous BW across cores. In this chapter, we have introduced the EOmesh that provides near-optimal homogeneous BW allocation across cores by (i) combining XY and YX routing policies and (ii) allocating weights accordingly. This allows limiting the local BW allocated to specific ports that cannot use all allocated slots, thus inducing some imbalance. Our results show large WCD reductions and WCET reductions in the range 5%-28%, while keeping hardware cost roughly unchanged. Moreover, we show that benefits hold in the context of parallel applications, whose WCET becomes less sensitive to the particular cores where they are allocated.

Chapter 5

Mesh NoC Multidimension Optimization

5.1 Introduction

Manycores bring several challenges for their adoption in the critical embedded market. One of those is deriving timing bounds to tasks' execution times as part of the overall timing V&V process [99]. In particular, the NoC has been shown to be the main resource in which contention arises, and hence hampers deriving tight bounds to the timing of tasks [132].

For widely-used wNoCs [6, 20], several proposals show how to compute latency upper-bounds to the different flows communicating on the manycore [51, 62] under some restrictions, e.g. deterministic routing. Unfortunately, WCET estimates computed with wNoCs are generally pessimistic when – as required to achieve composable estimates – no restrictions are imposed on when and where interference occurs in the wNoC. Interestingly, wNoCs offer several software-controllable parameters that allow optimizing (reduce) the WCD that packets crossing can suffer. These include mapping, routing, and allocation of weights (referred to as weighted allocation (walloc)) to arbitration policies in each router. NoC contention optimization solutions have been proposed for mapping [110, 112] and combining routing and mapping [111, 134]. Additionally, optimal allocation of weights to achieve fair BW balancing have been also proposed for TDMA [57] and wNoCs [64]. In general, those solutions do not tackle all parameters at once, which leads to globally suboptimal solutions. Overall, reducing WCD in NoCs is indeed a multidimensional problem and, to make things worse, strong dependencies exist between the different parameters. For instance, the impact of routing in WCD is heavily affected by the mapping of tasks to cores. Despite the interdependencies among these parameters, to our knowledge, no previous work proposes an integral solution to the problem of WCD reduction simultaneously optimizing mapping, routing and walloc.

In this chapter, we cover this gap by proposing *NoCo*, a wNoC ILP- and stochastic-based optimization framework that minimizes WCD estimates (and hence WCET estimates) of applications running in the wNoC-connected manycores.

In particular:

1. We analyze the main wNoC parameters that cause variability in WCD (mapping, routing, and walloc) and how they relate to each other. We propose a modeling approach that allows deriving the contribution of each wNoC parameter to WCD.
2. We show that reducing WCD is a multidimensional problem that we decompose into a stochastic-based optimization and an ILP formulation. The former covers the optimization of the routing, whereas the latter optimizes mapping and walloc.
3. We show the effectiveness of NoCo compared to hand-made setups and other approaches that optimize a subset of the parameters. Our results confirm that our multidimensional optimization approach achieves performance guarantees that outperform the other ones evaluated. We also show that optimizing VC allocation provides a subset of the configurations obtained with walloc, so that optimizing walloc makes VC not to provide any additional advantage.

5.2 Abstracting sources of jitter (WCD modeling)

We target a standard 2D *NxM* wormhole mesh NoC as the one described in Section 2.2.2. Input ports comprise a queue to store flits, or several if VCs are implemented.

5.2.1 Introduction to WCD modeling

The WCD of the packets of an application, and hence the WCET of the application, is heavily affected by the NoC parameters. In particular, the NoC has a twofold impact:

1. It affects the Zero-Load Latency (zll), i.e. the latency experienced by a packet to traverse the network from source to destination in the absence of contention.
2. More importantly, it plays a key role on the contention delay, bounded by WCD, that the application can suffer.

Finding an optimal network configuration to minimize WCD/WCET is a multidimensional problem encompassing several inter-dependent parameters. In order to make the optimization problem tractable, abstractions are needed to capture the impact of different parameters in a common modeling framework. This is challenging since existing abstractions are heterogeneous and not intended to fit ILP models.

In this chapter, we present a particular WCD-centric abstraction to address the main sources of jitter in a wNoC, namely: placement of tasks (threads) to cores, routing, and walloc.

Moreover, we also provide an abstraction for VCs that allows us to show that walloc provides a superset of the configurations obtained with VCs, so removing the need of having extra VCs to reduce contention if walloc is in place.

Table 5.1: Definitions used in this chapter.

Term	Definition
$R(x,y)$	Router with coordinates (x,y) in the NoC
F_i	Flow i: stream of flits traversing the same H-node route
R_i^j	For flow F_i is the router at hop j
$\widehat{FX_i^j}$	Set of flows targeting the same output port F_i targets at R_i^j
H_i	Number of hops in a flow F_i
P_i^j	No. of requests that may contend for the same R_i^j output port as F_i under the worst-case contention scenario
ER_i^j	Rate at which flits of flow F_i can be ejected from R_i^j in the absence of backpressure
D_i^j	Maximum time that a packet of F_i requires to go from the input port of R_i^j to its destination node
BW_i	Minimum bandwidth allocated to flow F_i at source node
$\underline{BW_i^j}$	Minimum bandwidth allocated to flow F_i at R_i^j
$\overline{BW_i^j}$	Progated worst-case ejection rate for F_i at R_i^j
NR_i^j	Number of queues that can potentially contend for an output port that F_i is targeting at R_i^j
$\omega(i,j)$	Function returning the index x of the worst possible destination flow F_x that starts at R_i^{j+1} and reaches the worst destination in terms of indirect blocking of packets of F_i

In order to derive the WCD for a flow F_i we build on the formulation in [51], see Equation 5.1.

Table 5.1 describes the terms used in this chapter. The first multiplicand provides an upper-bound to the number of rounds each packet in F_i is stalled until all contending requests in router R_i^j are able to progress, which is $NR_i^j - 1$ for round-robin.

The second multiplicand is the *indirect contention delay* each packet of F_i can suffer in each hop due to the worst possible destination flow $F_{\omega(i,j)}$. The worst destination of flow $F_{\omega(i,j)}$ is the one causing the highest contention to F_i in router R_i^j. It can be computed iterating all flows until they target their destination [51]. We also define $w(i,j)$ as a function that returns the index of the worst-destination flow.

$$\overline{WCD_i} = \sum_{j=1}^{H_i} \left(\underbrace{(NR_i^j - 1)}_{ContendingRequests} \times \underbrace{\prod_{m=1}^{H_{\omega(i,j)}} NR_{\omega(i,j)}^m}_{1/Bandwidth} \right) \tag{5.1}$$

In this chapter, we refactor this expression to capture the impact of walloc in the WCD of a flow F_i. In an arbitrary network the time required to process a packet, i.e. its network traversal time, can be computed based on the network BW. Let BW_i be the bandwidth assigned to F_i, then its traversal time can be computed as $1/BW_i$. Note WCD is the result of adding the time the network requires to process the $NR_i^j - 1$ corresponding requests at each of the H_i hops traversed. Then, from Equation 5.1 we can derive the minimum bandwidth BW_i^j allocated to any packet at R_i^j targeting the same output port F_i targets as follows:

$$BW_i^j = \frac{1}{\prod_{m=1}^{H_{\omega(i,j)}} NR_{\omega(i,j)}^m} \tag{5.2}$$

Alternatively, and with the aim of simplifying ILP formulation, the minimum allocated BW can be derived building on the concept of worst-case ejection rate (ER_i^j). ER_i^j is the worst ejection rate for flits of flow F_i in router R_i^j through the output port determined by the routing policy whenever the next router in the path (R_i^{j+1}) can accept an incoming packet. ER_i^j can be computed as $ER_i^j = 1/P_i^j$ where P_i^j equals NR_i^j when packets of F_i can be blocked due to contention and 0 otherwise[1]. The effect of backpressure is covered by the propagated worst-case ejection rate that represents the minimum BW allocated to F_i in router R_i^j. Let $\widehat{FX_i^j}$ be the set of flows that contend for the output port targeted by F_i in R_i^j the worst-case propagated ejection rate $\overline{BW_i^j}$ can be formulated as follows:

$$\overline{BW_i^j} = \min_{\forall F_x \in \widehat{FX}} \left(\prod_{k=j}^{H_w(x,j)} \frac{1}{P_x^k} \right) \tag{5.3}$$

In the formulation above when P_x^k is 0 the ejection rate and the corresponding allocated BW are defined to be ∞, representing there is no contention due to backpressure and the packets of F_i can progress without contention to the next router. Note also that once in a given router R_i^j packets of F_i suffer from contention, P_x^k values cannot be equal to 0 in subsequent routers due to backpressure effects.

Building on $\overline{BW_i^j}$, we can derive the WCD that a packet of flow F_i takes to reach its destination node from the input port of router R_i^j. To that end, we define D_i^j as the latency that a packet requires going from R_i^j to destination. Overall, the recursive definition of D_i^j is as follows:

$$D_i^j = \frac{1}{\overline{BW_i^j}} + D_i^{j+1} \tag{5.4}$$

Note that WCD_i is then equivalent to D_i^1 and can be obtained recursively by adding the time to move from each router to the next one, where the time to move from one particular router R_i^j to R_i^{j+1} is upper-bounded by $1/\overline{BW_i^j}$ and equal to 0 when $\overline{BW_i^j}$ is ∞.

[1]Note that there are cases in which in spite of the routing does not prevent flows to contend with F_i, no contention with other flows can occur due to the way the flows are mapped to the network.

We have thoroughly validated WCD values computed using the expressions above by observing WCD values obtained for XY routing and round-robin arbitration match the ones obtained with formulations in [51] and [62] for the analyzed flows. Furthermore, we have assessed the accuracy of our model in upper-bounding the actual contention caused in the wNoC by empirically reproducing a worst-congestion scenario using gNoCsim [127]. To that end, we simulate the traffic generated by a memory-intensive micro-kernel in which all tasks in the NoC send packets to memory steadily. We have performed this experiment for 2 different network setups based on NoC sizes from real systems (e.g. 6x4 in Intel SCC [5] and 6x6 in Tilera-Gx36 [20]). To ensure a steady congestion state is reached, we have taken measurements once at least 1,000 packets per node have been injected. We also repeat the measurement process until all nodes have sent at least 2 million requests. Figure 5.1 shows the comparison of the analytically computed values with respect the ones obtained with simulations (e.i. $nWCD = computedWCD/obtainedWCD$). The comparison is performed for the flows with the minimum and maximum WCD (referred to as *Fmin* and *Fmax*, respectively), and the average across all flows, referred to as *meanF*. As it can be seen in the figure our expressions provide a tight upper-bound of the NoC contention.

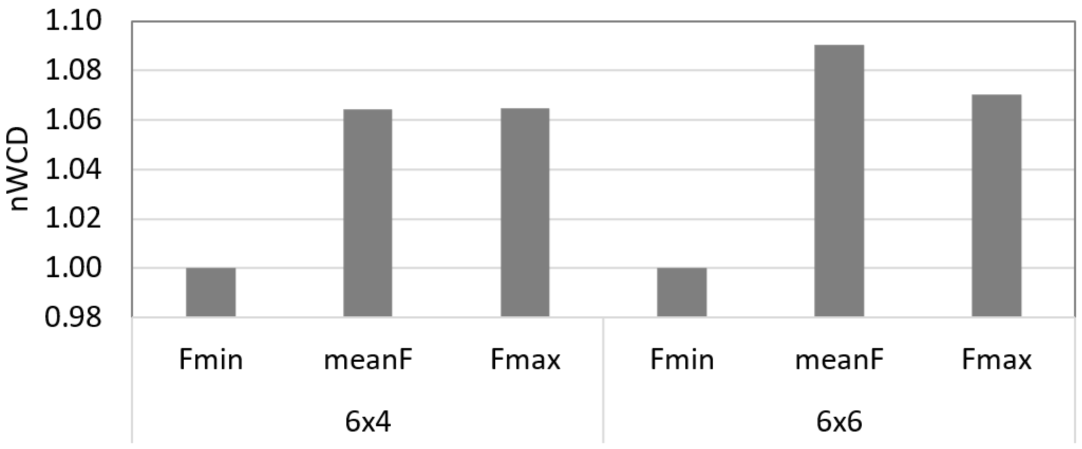


Figure 5.1: Tightness of WCD bounds for a 6x4 and 6x6 wNoC.

5.2.2 Routing

In this work we consider only routing policies with minimal distance routes, that is those policies forward packets in each router in any direction that guarantees that the distance to the target decreases by one hop. For instance, XY routing meets this constraint.

Moreover, due to their suitability for critical real-time systems and their efficiency in terms of implementation, we further assume deterministic routing policies, so that a given packet can only be sent to a specific output port in a router, as dictated by the routing policy. For instance, recalling example in Figure 5.2(a), a packet sent from $R(2,2)$ to $R(0,0)$ with XY routing can only move in the X direction until $R(0,2)$, and then in the Y direction until $R(0,0)$. Note that with deterministic routing, the

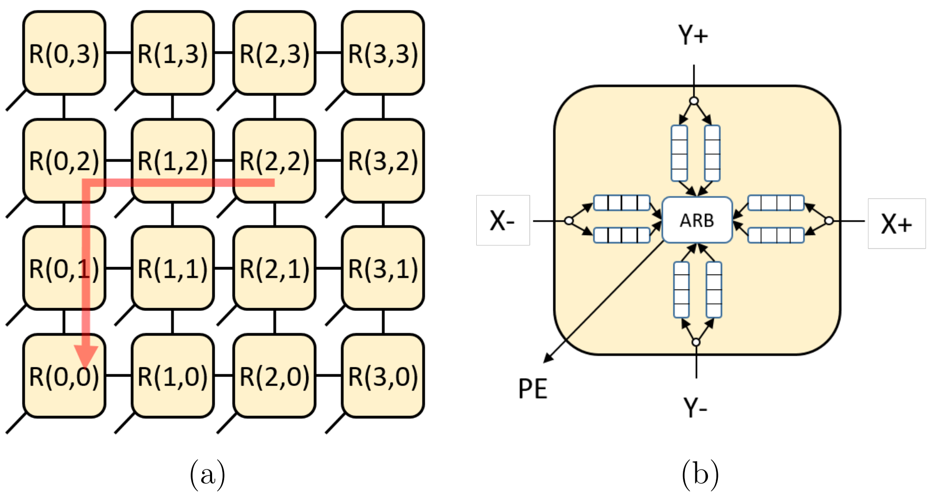


Figure 5.2: Mesh basics. (a) Router coordinates in a 4x4 part of a mesh. (b) Ports and VCs competing for output PE port in a canonical 2D-mesh router.

route to follow by a packet is independent of whether there is contention in the path. Such contention may delay progress, but not alter the route.

Routing determines for a packet stored in an input port queue of the router the other input ports can contend for a given output port. To simplify the routing algorithm implementation, in the routing policies considered, all flows in the wNoC share the same routing decisions/restrictions. This approach is also followed by the majority of on-chip routing algorithms including XY, dimension-order-routing, segment-based routing [135], and derivatives.

Routing restrictions help to determine the exact number of requests (P_i^j) that might contend at router R_i^j for the same output port as F_i in the worst-case situation. For instance, P_i^j values for a mesh with XY routing and assuming all-to-all communication are determined as: $P_i^j = 2$ if the destination is X or $X-$ and $P_i^j = 4$ if the destination is Y or $Y-$ or the PE. For a particular routing policy P_i^j is fixed, with different values across routers. Hence, routing can be abstracted by replacing P_x^k in Equation 5.3 by the actual number of flows that can contend for the output port used by the routing algorithm at each router R_i^j. Therefore, by setting specific routing directions in each router, P_i^j changes for the output port of each router and new D_i^j values (per-core WCD) are obtained for each core.

Deadlock Avoidance. Routing algorithms in wormhole have to ensure deadlock freedom. Deadlock situations in wNoCs occur when packets are waiting on each other in a cycle. For instance, XY algorithm avoids deadlock situations by prohibiting certain turns. We prevent deadlocks by ensuring that there are no prohibited cycles in the generated routing. This can be alternatively achieved by imposing further restrictions in the routing inputs of the model or using specific VCs to this end. However, this is orthogonal to our overall formulation just removing some routing options.

5.2.3 Weight Allocation (Arbitration)

Wmeshes allow allocating heterogeneous BW in the routers to the different flows to accommodate the different needs of different communication flows in the weighted NoC [85]. Weighted arbitration can be employed to achieve a globally-fair (homogeneous) BW allocation across cores [64]. Conceptually, given a NoC with NxM nodes, globally-fair wmeshes reduce the BW for nodes whose allocated BW is above $\frac{1}{N \times M}$ for a given destination node and increases it for those whose BW is below. This is achieved by using, for instance, a larger arbitration window in the case of round-robin, so that a larger number of slots is given to some ports so that the overall BW allocated to each core to the destination can be arbitrarily chosen.

So far we have assumed round-robin with homogeneous weights across flows, so that given P_i^j flows contending for an output port in a router, each one is allowed to eject a packet at a rate $ER_i^j = \frac{1}{P_i^j}$ whenever the next router in the path accepts incoming packets. Hence, the total ejection rate of the output port is $\frac{1}{P_i^j}$ for each of the P_i^j flows.

For instance, given $P_i^j = 3$ contending flows, the default round-robin arbitration policy uses arbitration windows with 3 slots, one of which is given to each flow. Hence, each flow has 1/3 ejection rate. With weighted arbitration, we can set a window with an arbitrary number of slots (e.g. 5) and allocate them to flows as wanted. For instance, we could allocate 3 slots to flow 1 and 1 slot to each other flow, so that their respective ejection rates would be 3/5, 1/5 and 1/5. Appropriate weights can be set up to modify the WCD of each core. For instance, authors in [64] have shown that homogeneous BW can be achieved by properly allocating weights in routers, which homogenizes to some extent WCD values[2]. In general, weight allocation can be set as needed – with a granularity limited by the size of the arbitration window size – and abstracted by using appropriate P_i^j values for each flow in each router for the computation of the D_i^j values.

5.2.4 Virtual Channels

As we stated in the VC part in Section 2.2.2, VCs are used to multiplex physical channels. In the context of critical real-time, static VC allocation alleviates contention reducing WCD by providing isolation and reducing the overlapping between the routes followed by the different flows.

With VCs, two different arbitration rounds take place in the router, see Figure 5.2(b). A first arbitration determines the input port that is granted access to the output port. A second arbitration selects the VC that is granted access. For instance, given a router with 2 VCs that are statically allocated, with 1 flow in the former (VC1) and 3 in the latter (VC2), the arbiter grants access alternatively to each VC, and within the second VC in a round-robin fashion to each flow. Hence, the flow in VC1 has an ejection rate of 1/2, whereas each of the other flows has an

[2]While weights can homogenize BW, they cannot mitigate the communication cost caused by physical distance to destination (zll), so in general, WCD cannot be made fully homogeneous.

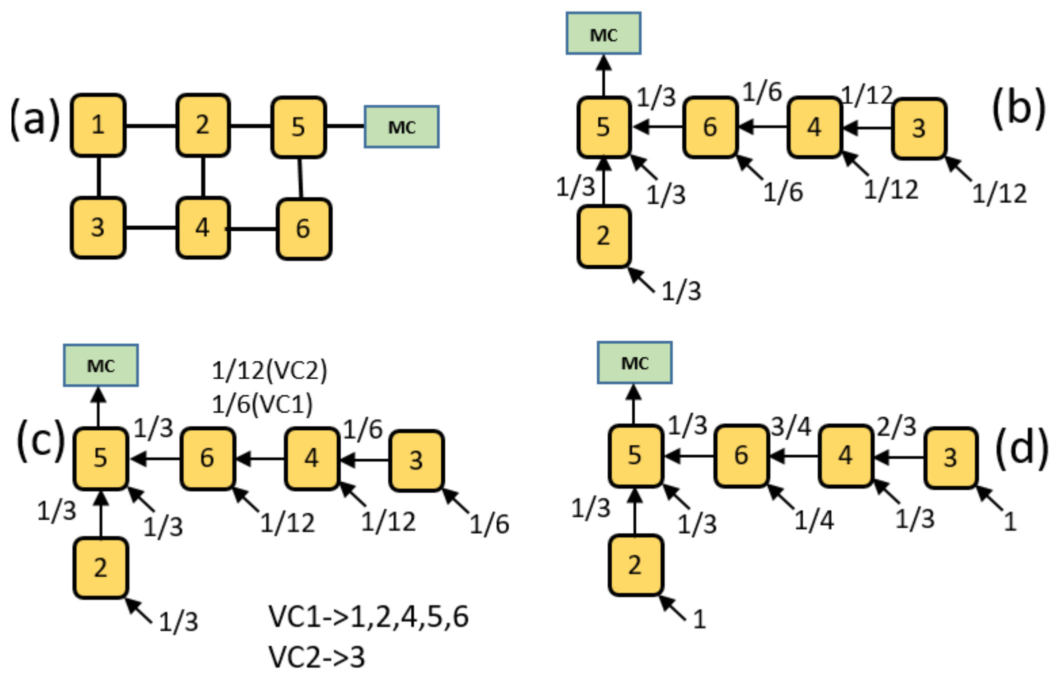


Figure 5.3: Example of VCs and its equivalent walloc design.

ejection rate of 1/6. As shown, this formulation is identical to that of the weights for wmeshes, with the difference that weights can be allocated as needed at a much finer granularity.

We illustrate how VCs are subsumed by walloc with an example for a $XY - RR$ 3x2 mesh NoC, see Figure 5.3(a). Its representation with no VCs in the form of a tree to access the memory controller (MC) is shown in Figure 5.3(b), where values at the edges indicate the ejection rates for each port. Figure 5.3(c) shows the tree with 2 VCs where node 3 is allocated to VC2 and the rest of nodes are allocated to VC1. In the mesh without VCs, the BW is evenly shared across input ports, which makes node 3 to have only 1/12 of the BW (see Figure 5.3(b)). Instead, when we use the particular VC allocation in Figure 5.3(c), BW is allocated in a different manner making node 3 enjoy 1/6 of the total BW. This occurs because BW allocation is modified in the links from 6 to 5 (6to5) and from 4 to 6 (4to6) since two VCs are multiplexed over the same physical link. In particular, 6to5 BW (1/3) and 4to6 (1/6) are shared between VC1 and VC2. In 6to5, 1/6 of the BW is allocated to VC1 and the other 1/6 is allocated to VC2. Thus, the node attached to router 6 gets 1/12 of the BW and the remaining 1/12 is allocated to the node attached to router 4. On the contrary, since node 3 uses VC2, its allocated BW is still the one coming from 6to5 (1/6). Finally, once BW per core is determined, it is trivial to set the weights that lead to that particular BW allocation starting from the destination node and splitting it as needed. In this particular example, Figure 5.3(d) shows the walloc that leads to identical behavior to the case of 2 VCs on $XY - RR$.

5.2.5 Mapping

The WCET of parallel and single thread applications can be obtained using the formulation below. For parallel applications WCET is determined by the thread finishing the latest.

The WCET of an individual thread (application) T_i, $WCET_i$, is computed as shown in Equation 5.5.

$$WCET_i = ETI_i^j + Nreq_i \cdot D_i^j \tag{5.5}$$

The ETI_i^j figure corresponds to the classical notion of WCET, defining an upper-bound for the execution of T_i *in isolation*, under any possible execution scenario. In a NoC, however, the ETI_i^j bound depends on the node where T_i executes, and must be defined for all nodes j in the system (i.e. all different Cartesian distance values). Since ETI_i^j is independent of contention and we assume routing policies with minimal distance routes, some nodes will share the same ETI_i^j. Hence, deriving the worst-case execution time bounds in isolation of T_i just on a subset of the cores will suffice to represent all potential distances to the memory node. For instance, recalling the example in Figure 5.2(a), and assuming that memory is located in $R(0,0)$, we could run T_i in the cores at routers $R(0,0)$, $R(0,1)$, $R(0,2)$, $R(0,3)$, $R(1,3)$, $R(2,3)$ and $R(3,3)$, since, for instance, ETI_i^j will be identical at $R(0,2)$, $R(1,1)$ and $R(2,0)$ since all them have the same Cartesian distance to the destination.

Similarly to ETI_i^j, $Nreq_i$ captures the worst-case number of requests triggered by T_i under any possible execution conditions, with the difference that it does not depend on the execution node (i.e., it is constant across cores). The worst-case number of requests is an essential dimension to consider when bounding contention effects: $Nreq_i$ bounds can be derived either statically [33] or based on measurements [78]. Note that it is fundamental to conservatively consider WCET and number of requests separately as the corresponding scenarios (e.g., input data or execution path) do not necessarily match [78].

Finally, D_i^j is derived according to Equation 5.4. Ultimately, for a given thread-to-core mapping, D_i^j is constant for each node j (i.e., it is constant once T_i is mapped to a core).

Therefore, the effect of thread-to-core mapping on $WCET_i$ can be abstracted by using the corresponding ETI_i^j and D_i^j values precomputed for the core where the thread is mapped.

5.3 Formulation

We propose a hybrid NoC Optimization (NoCo) approach to solve the multidimensional problem of NoC parameter optimization. Our approach combines optimization algorithms and ILP formulation to reach its goals. The approach also comprises a less important post-processing module. In particular, building on the analysis Section 5.2, the optimization algorithms of NoCo cover the variability of routing while mapping and walloc are optimized via an ILP formulation.This is

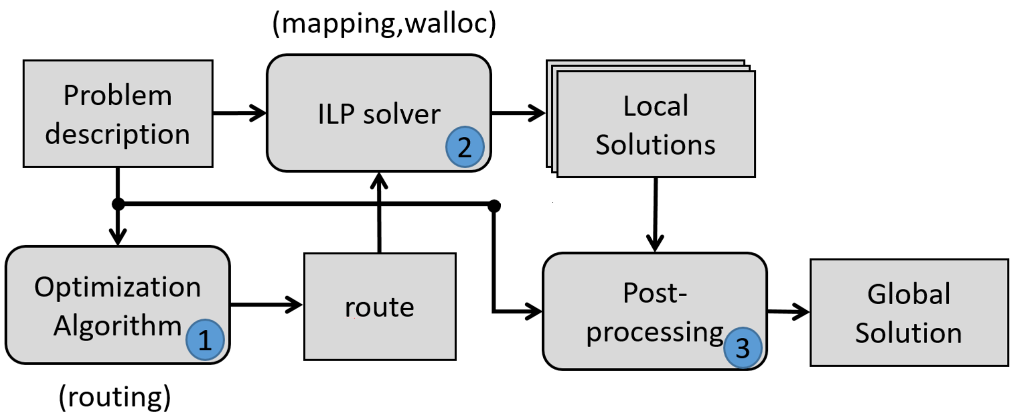


Figure 5.4: Main stages of NoCo.

sketched in Figure 5.4. The overall goal of NoCo is to derive the WCET of the application factoring in NoC contention.

Under a given routing, the zll for a given thread is constant given a specific core c while the WCD for the packets sent by that core (D^c) depends on the particular routing (r) followed by the packets of the other cores, walloc (w), and VCalloc (v) used. Formally stated, $D^c = f(r, w, v)$. In fact, as we show later, given a routing r, the WCD imposed by a given WCD w and VCalloc v can be obtained without using VC, since there exists a walloc w' that delivers the same WCD across cores. Hence, $D^c = f(r, w, v) = f'(r, w')$.

It is noted that, a holistic approach modeling also routing as an ILP variable, would cause the optimization problem to become quadratic as the correlation between BW quotas assignment and routing cannot be modeled with linear constraints. In contrast to ILP or MIP (Mixed-Integer Programming) problems, quadratic optimization problems are generally NP-hard and state-of-the-art solvers are normally incapable of proving the optimality of any solution possibly found.

For the routing optimization, we opted for a stochastic approach instead of a heuristic-based one for a two-fold reason. First, heuristic-based solutions present the problem that in general it is not possible to assess their quality, since they might be subject to local maxima. Furthermore, the use of routing-only heuristics to find a (local) solution can result in negative overall results when walloc and mapping optimization are applied. And second, as detailed later in this section, a stochastic-based solution allows exploring a restricted number of routes so with limited exploration time, while allowing to argue that the best evaluated route belongs to the top $X\%$ best routes.

5.3.1 Routing

For time predictability reasons, we stick to static (predictable) routing that must further avoid deadlocks. In particular, we explore XY, referred to as '0' in the following figures, and YX ('1') routing for each core. Combining both allows for achieving good malleability in limiting the number of contenting flows P_i^j in the

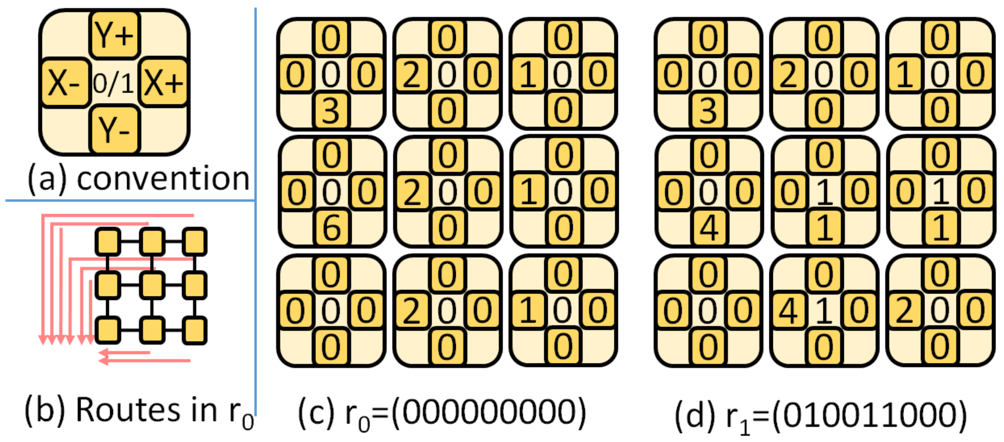


Figure 5.5: Examples of different routings and pressure on different output ports.

routers of a particular flow F_i. Figure 5.5 illustrates with an example how the proposed routing determination algorithm works. Figure 5.5(a) sketches the graphical convention we use to represent the routing selected by each node and the number of flows contending in each output port. If we apply XY for all nodes, $r_0 = (000000000)$, we obtain the routes in Figure 5.5(b) and the contention per output port as shown in Figure 5.5(c). We can see that some output ports suffer high contention (up to 6 flows). A different arbitrary routing $r_1 = (000010010)$ presents lower maximum contention per output port, see Figure 5.5(d).

It follows that good routes are those that limit the number of contenting flows P_i^j in the routers used by of a particular flow F_i. However, this is a local (i.e. routing-only) optimization. When combined with mapping and walloc optimization, routes that create more contention – and hence are less optimal from the routing point of view – can result in reduced maxWCD/sumWCD. For instance, let us assume that under a particular routing one route suffer high contention while the rest suffers low contention. Further, assume a second routing that much better balance contention. For an application in which one thread (task) is insensitive to NoC WCD, while the rest of the threads are, the former (less balanced) routing results in reduced WCD and hence WCET.

Hence, since a priori we cannot determine what a good route is, it would be hard to define a standard heuristic approach to address the problem (e.g. genetic algorithms, simulated annealing). Besides, those heuristics would not allow assessing how far a solution (routing) is from the optimal one.

In order to cover both issues, we use a stochastic approach based on a Monte-Carlo experiment. Basically, we produce static routings schemes by selecting randomly whether each node uses XY or YX routing. Hence, from the finite – but huge – population of all routings, a sample is selected using random sampling with replacement. This can be done by following the simple approach shown in algorithm 1: for every node in the NxM mesh we generate a random integer: if it is odd we assume XY routing (0) and YX (1) otherwise. We ensure resulting configurations are deadlock-free by filtering out the samples in which routing cycles are created [135].

Algorithm 1 Algorithm to generate random routings.

```
1 procedure GEN_ROUTE_RANDOM($sd$,$iter$,$type$, $ncount$)
2     for ($i = 0; i < NxM; i++;$) do
3         $mapping[i] = random()\ mod\ 2$;
4     end for
5 end procedure
```

With this approach, we can probabilistically reason on the quality of a given routing. Let C be the probability that a sample of random routings contains at least one of the top X ($X \in [0..1]$) routings, i.e. fraction of routing from the entire population providing the best results for a given target metric (e.g. maximum WCD across cores). Let $\bar{C}$ the complementary of C, i.e. $1-C$. The probability that a single random routing is not in the top X of the population is $(1 - X)$. The probability that all k mappings do not belong to the top X is, therefore, $(1 - X)^k$. Hence, its complementary, C, is the probability that the best routing choice in the random sample belongs to the top X routings. Hence, $C = 1 - (1 - X)^k$.

Table 5.2: $\bar{C}$ for different random sample sizes.

	C̄				
X	$\mathbf{10^{-2}}$	$\mathbf{10^{-3}}$	$\mathbf{10^{-4}}$	$\mathbf{10^{-5}}$	$\mathbf{10^{-6}}$
$\mathbf{10^2}$	0.37	0.90	0.99	0.999	0.9999
$\mathbf{10^3}$	$4.3 \cdot 10^{-5}$	0.37	0.90	0.99	0.999
$\mathbf{10^4}$	$< 10^{-43}$	$4.5 \cdot 10^{-5}$	0.37	0.90	0.99
$\mathbf{10^5}$	$< 10^{-300}$	$< 10^{-43}$	$4.5 \cdot 10^{-5}$	0.37	0.90
$\mathbf{10^6}$	$< 10^{-300}$	$< 10^{-300}$	$< 10^{-43}$	$4.5 \cdot 10^{-5}$	0.37

As illustrated in Table 5.2, the probability of *not* having, for instance, any routing within the top best 0.01% routings ($\bar{C} = 10^{-4}$) with a sample size of 100,000 is of around $4.5 \cdot 10^{-5}$ (so 0.0045%). Thus, the probability of having at least one of those top $X = 0.0001$ routings is $C = 0.999955$ (above 99.99%). In general, we observe that a sample size of around 1,000 random routings allows guaranteeing with very high confidence that at least one of the top 1% best routings is observed.

Once a routing is fixed, we can derive the WCD for every packet going from a given node to the memory. The route information is encoded in a *route*, see Figure 5.4, passed to the ILP model to optimize mapping and walloc.

5.3.2 Mapping and walloc

The WCD potentially suffered by a task τ_i upon each performed memory access, when executing on a wmesh NoC, is determined by the interrelation of several factors: the router τ_i is mapped to, the adopted routing configuration (with its inherent flow constraints) and the BW distribution along the mesh [3]. The WCD has to be accounted for in the definition of the WCET of all tasks. We present an ILP

[3]In this subsection we refer to the walloc optimization problem as BW distribution problem.

Table 5.3: ILP model notation.

$\mathcal{R} = \{R_0, \ldots, R_s\}$	Set of routers in the mesh
$\mathcal{N} = \{N_0, \ldots, N_s\}$	Set of computational nodes in the mesh
$\mathcal{T} = \{\tau_0, \ldots, \tau_n\}$	Task set to be executed on the mesh
a_i	Number of memory accesses triggered by τ_i
c_i^k	Execution time in isolation of τ_i when executed on node N_k
$WCET_i^k$	Worst-case execution time of τ_i when executed on node N_k
WCD^k	Worst-case delay suffered by a task mapped to node N_k
$\mathcal{M} : \mathcal{T} \mapsto \mathbb{Z}$	Mapping of τ_i to indexes in the node set $\mathcal{N}$
$\mathcal{B} : \{\mathcal{R} \cup \mathcal{N}\} \mapsto \mathbb{R}$	Mapping of R_k and N_k to a bandwidth assignment
$WCET_i^{\mathcal{M},\mathcal{B}}$	WCET for task τ_i under mapping $\mathcal{M}$ and bandwidth allocation $\mathcal{B}$
$WCD^{\mathcal{M}(i),\mathcal{B}}$	WCD suffered by τ_i, under mapping $\mathcal{M}$ and bandwidth allocation $\mathcal{B}$.
$\mathcal{H} : \mathcal{R} \mapsto \mathcal{P}(\mathcal{R})$	List of routers (hops) a packet needs to traverse to reach a destination from a given source router
$\mathcal{L} : \mathcal{R} \mapsto \mathcal{P}(\mathcal{R})$	Map of active links in the mesh
$BWR_k^{\mathcal{B}}$	Bandwidth assigned to the output port of router R_k
$BWN_k^{\mathcal{B}}$	Bandwidth assigned to the output port of the computational node attached to router R_k

formulation for optimizing the WCET of a task by finding an optimal task mapping and BW assignment, under a given routing configuration.

We consider a 2D mesh NoC comprising a set of routers $\mathcal{R} = \{R_0, \ldots, R_s\}$, with associated computational nodes $\mathcal{N} = \{N_0, \ldots, N_s\}$ (such that N_k is attached to R_k) and a set of tasks $\mathcal{T} = \{\tau_0, \ldots, \tau_n\}$, that need to be executed on the mesh NoC. Each task τ_i is characterized by the number of performed memory accesses a_i, and an execution time bound computed in isolation, dependent on the node it executes on: we use the notation c_i^k to represent the timing bound of τ_i, when executed on node N_k (implicitly accounts for the routing policy). The WCET of a task τ_i when executed on router R_k, with a given BW quota, is obtained by inflating the execution time in isolation with a worst-case delay penalty for each memory access a_i:

$$WCET_i^k = WCD^k * a_i + c_i^k \tag{5.6}$$

In turn, the worst-case delay WCD^k potentially suffered by τ_i when executed on node N_k is determined in particular by the routing policy in use, and the specific BW assignment among nodes.

Given a fixed routing policy (configuration), our ILP model leverages on task mapping and bandwidth assignment to optimize the WCET of tasks.

B.1. Objective Function

The ILP formulation supports two objective functions representative of two typical metrics for assessing the performance of a wNoC. We are interested in finding the optimal task-to-node mapping and bandwidth assignment that minimizes:

- *SumWCD.* For independent workloads, reducing the addition of the WCD of all tasks/threads, helps optimizing resource usage and improving overall guaranteed performance. Reducing *SumWCD* can be done under some constraints on the maximum WCD allowed per task. Overall this metric, which minimizes the WCD experienced by all requests of all threads in the workload, is particularly relevant to assess the global efficiency of our approach.
- *MaxWCD* is derived by computing the total WCD experienced by all requests for each thread, and minimizing the maximum WCD value across threads. For parallel applications, this metric provides information about the thread experiencing the highest contention and hence potentially delaying the completion of the application.

We introduce $\mathcal{M}(i) : \mathcal{T} \mapsto \mathbb{Z}$ to define a mapping from tasks to computational nodes in the mesh, where $\mathcal{M}(i)$ returns the index k that identifies the node $N_k \in \mathcal{N}$ such that τ_i is mapped to N_k. Note that N_k is by definition attached to router R_k. Similarly, we define $\mathcal{B} : \{\mathcal{R} \cup \mathcal{N}\} \mapsto \mathbb{R}$, mapping from routers and nodes to a bandwidth quota, where $\mathcal{B}(R_k)$ returns router R_k bandwidth assignment and $\mathcal{B}(N_k)$ returns the same for node N_k. It is worth noting that $\mathcal{B}$ depends on the specific routing configuration.

A formulation for the WCET parametric on task mapping and bandwidth assignment would be as follows, where $WCD^{\mathcal{M}(i),\mathcal{B}}$ identifies the per-access WCD potentially suffered by τ_i, under mapping $\mathcal{M}$ and bandwidth allocation $\mathcal{B}$.

$$WCET_i^{\mathcal{M},\mathcal{B}} = WCD^{\mathcal{M}(i),\mathcal{B}} * a_i + c_i^{\mathcal{M}(i)} \tag{5.7}$$

For *maxWCD*, the ILP is meant to optimize the time required to execute the longest activity in the mesh.

$$\min_{\mathcal{M},\mathcal{B}} \max_{\tau_i \in \mathcal{T}} WCET_i^{\mathcal{M},\mathcal{B}} \tag{5.8}$$

For *sumWCD*, instead, ILP aims at minimizing the resources required to execute whole workload.

$$\min_{\mathcal{M},\mathcal{B}} \sum_{\tau_i \in \mathcal{T}} WCET_i^{\mathcal{M},\mathcal{B}} \tag{5.9}$$

B.2. Modeling WCD

Similarly to the execution time in isolation, WCD is affected by the task location in the mesh and the routing policy in use. However, while the set of C_i^k is an input variable to the ILP model, the WCD is indirectly a decision variable, as a function of BW allocation. The part taken by the routing policy in the computation of the WCD has to be made explicit, in the same way as the mesh BW assignment.

The BW allocated to each element in the mesh is modeled by an explicit decision variable: $BWR_k^{\mathcal{B}}$ defines the bandwidth allocated to the output port of router R_k under bandwidth assignment $\mathcal{B}$, whereas $BWN_k^{\mathcal{B}}$ stands for the bandwidth assigned to the computational node N_k, attached to R_k.

Routing is indeed relevant for determining both the packet route and the feasible bandwidth split rules. We defined two abstractions, $\mathcal{H}$ and $\mathcal{L}$ to capture these relevant aspects.

$\mathcal{H} : \mathcal{R} \mapsto \mathcal{P}(\mathcal{R})$ models the list of routers (hops) a packet needs to traverse to reach a destination from a given source router. This information is necessary to accumulate the WCD along the end-to-end flow.

$\mathcal{L} : \mathcal{R} \mapsto \mathcal{P}(\mathcal{R})$, instead, is a map of the active links in the mesh, in accordance with the routing rules. This information is fundamental to encode the rules for BW allocation.

Given a routing policy defined by the pair $< \mathcal{H}, \mathcal{L} >$, we model BW allocation rules and WCD bound as follows. First Eq. 5.10 models the fact that the BW assigned to the output port of a router R_k is determined by the BW in the node-local computational node BWN_k, and the cumulative BW propagated by other routers connected to R_k through an active link.

$$BWR_k^{\mathcal{B}} = BWN_k^{\mathcal{B}} + \sum_{R_t \in \mathcal{L}(R_k)} BWR_t^{\mathcal{B}} \tag{5.10}$$

Having defined the bandwidth per router, it is possible to model the WCD for a router R_k as follows:

$$WCD_i^{\mathcal{M},\mathcal{B}} = \frac{1}{BWN_{\mathcal{M}(i)}^{\mathcal{B}}} + \sum_{R_t \in \mathcal{H}(\mathcal{M}(i))} \frac{1}{BWR_t^{\mathcal{B}}} \tag{5.11}$$

As defined in Eq. 5.11, the WCD per-access for a task mapped to router R_k is determined by the cumulative inverse BW across all hops in the path from source to destination. The computed WCD can be used for the WCET computation in Eq. 5.7.

B.3. Modeling Constraints

The mesh topology and routing policy allow to derive a set of constraints to guide BW allocation across routers and task mapping. Some simple constraints can be put on $\mathcal{B}$ by defining the domain space for the ILP variable:

- The WCD is always greater than zero.

$$WCD_i^{\mathcal{M},\mathcal{B}} > 0.0 \quad \forall i, \mathcal{M}, \mathcal{B}$$

- Allocated bandwidth per router must be larger than zero.

$$BWR_k^{\mathcal{B}} > 0.0 \quad \forall k, \mathcal{B}$$

- The cumulative amount of bandwidth allocated to all computational nodes must be exactly one

$$\sum_{R_k \in \mathcal{R}} BWN_k^{\mathcal{B}} = 1$$

Similarly, several constraints can be defined on $\mathcal{M}$.

- Each router cannot be assigned to more than one thread

$$\mathcal{M}(i) \neq \mathcal{M}(j) \quad \forall i, j$$

- Each thread can only be assigned to one router

$$|\mathcal{M}(i)| = 1$$

B.4. Putting it all Together

NoCo optimizes routing with a stochastic approach that allows assessing the quality of the explored routes. The alternative approach of expressing the routing as a decision variable in the problem formulation has the notable drawback of breaking the linearity of the model. Modeling the routing as the combination of two new variables, representing the BW distribution and the flow configuration, would turn the computation of both the router BW quotas and, ultimately, the WCD into a quadratic optimization problem. For each route, an ILP model optimizes mapping and walloc (bandwidth allocation) to minimize either $maxWCD$ or $sumWCD$. In the next section, we empirically assess the benefits of NoCo over other existing approaches.

5.4 Experimental Evaluation

We conduct the evaluation of NoCo on SoCLib [126] and gNoCSim [127] described in Section 3.3. In this case, we model the TILED MC where each tile comprises L1 cache memory and a core that communicates with the rest of tiles and memory using a NoC router (see Section 3.2 for more details.).

The manycore also includes a unified distributed shared L2 cache memory, so that each core has a local partition of the L2 cache. To increase the predictability of the cache hierarchy the L2 cache is partitioned and each core is provided with a 64KB region with 64B per cache line. L2 partitioning does not only avoid inter-task interferences in the L2 but allows both isolating cache coherence between different critical and non-critical tasks [136] and/or to disable coherence support to avoid further NoC interferences.

Memory requests access main memory through the NoC. Each memory request operates at a granularity equal or smaller to a L2 cache line, thus transferring up to 16B (128 bits) per NoC packet, which also has 16 control bits. Overall, packets with up to 144 bits are sent in a single flit through a 144-bit link width, which allows all NoC packets affecting our tasks to have one flit. Despite one-flit packets are preferred to reduce contention, our model is also able to deal with other packet lengths by computing worst-case ejection rates considering packets of the maximum length.

Note that routers are connected through 2 links, each one sending data in one direction. Hence, whereas memory accesses may experience contention, memory responses cannot since, at every router only one input port (the one used for responses) contends for the corresponding output port (the one to move the response to destination).

To evaluate NoCo when reducing the WCET of parallel applications supported by the ADA programming language [133] but also when consolidating single-task applications, we use the $A-H$ self-generated benchmark suit described in Section 3.4.

From this set of benchmarks, we have generated workloads with NxN benchmarks for each mesh size analyzed. Some workloads are homogeneous, thus having all benchmarks of the same type (so 8 workloads in total), whereas others (4 workloads) are heterogeneous by choosing the NxN benchmarks randomly (with replacement) out of our set of 8.

5.4.1 Reference techniques

As reference approaches to compare NoCo against, we use:

- $XY - RR$ deploys predictable XY routing with standard predictable round-robin arbitration in each router.
- $XY - WRR$ is analogous to $XY - RR$ except that we modify weights to balance the BW across all nodes [64]. This makes the WCD of all nodes accessing memory more homogeneous than in $XY - RR$, effectively mitigating mapping as a source of variability.

In the rest of this section we refer to our technique as $rILP(m, w)$ since NoCo optimizes routing, r, first using the stochastic approach in Section 5.2.2 with samples of size 10 for 3x3 experiments and 100 for 4x4 experiments, and mapping and walloc using the ILP approach, $ILP(m, w)$, presented in Section 5.3. Results are shown in the form of WCET reduction w.r.t. the $XY - RR$ case. For instance, the maxWCET improvement of $ILP(m, w)$ is obtained as:

$$1 - \frac{maxWCET_{ILP(m,w)}}{maxWCET_{XY-RR}} \tag{5.12}$$

5.4.2 Incremental evaluation

In order to assess the benefits of optimizing each individual parameter of a NoC (mapping, routing, and walloc), we present the results obtained with our approach as it incrementally optimizes them. In particular, we compare these setups:

(a) $XY - WRR - ILP(m)$. Both routing (XY) and weights (WRR) are fixed. Therefore, NoCo only optimizes mapping, i.e. it explores the different mappings of tasks to cores ($ILP(m)$).

(b) $XY-ILP(m,w)$. Only routing is fixed (XY) and NoCo explores (i.e. optimizes) both mapping and weight allocation at the arbitration level ($ILP(m,w)$).

(c) $rILP(m,w)$. All three NoC parameters are optimized by NoCo: routing, mapping and walloc.

With this incremental approach we can derive the benefits of optimizing each parameter: (a) gives the benefits of optimizing mapping; (b)-(a) the benefits of optimizing walloc; and finally (c)-(b) the benefits of optimizing routing.

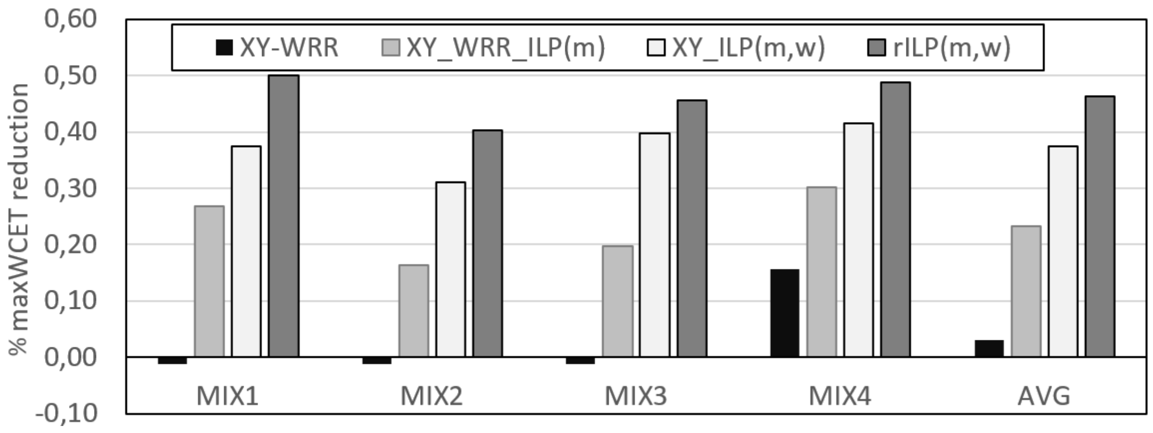


Figure 5.6: Effect of incremental optimizations: mapping, walloc, and routing.

Figure 5.6 shows the impact of applying the optimization of the different parameters incrementally. In particular, we evaluate the 4 heterogeneous 9-task workloads, which we refer to as MIX1 to MIX4 on a 3x3 mesh NoC. Note that the chart shows the maxWCET reduction that each technique obtains over the baseline $XY-RR$.

Compared to the baseline $XY-RR$, we can see that $XY-WRR$ [64] obtains similar results (-1%) for three of the workloads, while for one workload it is 16% better.

Effect of mapping. Compared to the baseline $XY-RR$, $XY-WRR-ILP(m)$ obtains maxWCD reductions of 23% on average, showing the benefits of optimizing task mapping.

Effect of mapping and walloc. The combined effect of mapping and walloc $XY-ILP(m,w)$ results in further reductions of maxWCD across all workloads. On average improvements 37%, so that the benefit of optimizing only *walloc* is 14 percentage points.

Full optimization. When NoCo optimizes all three NoC parameters it produces the tightest WCET estimates, with maxWCET reductions of 46% on average, so that the benefit of routing optimization is 9 percentage points.

Overall can see that results are consistent across all workloads ranging from 40% to 50%. These results show the benefits of simultaneously optimizing routing, mapping, and walloc and how this provides the best WCET reduction results.

We evaluate $rILP(m,w)$ in the rest of this section without further breaking down experiments incrementally.

5.4.3 Optimal routing

In this section, we compare the random routing selection to exploring all possible routings. While the latter is not feasible in general due to its huge execution time overheads, for the small set of experiments we evaluate it provides evidence on the benefits of the stochastic approach to optimize routing.

Figure 5.7 shows the maxWCET obtained with random routing selection normalized so that 100% corresponds to the lowest maxWCET and 0% to the highest maxWCET. Thus, we show how close to the optimal is the best solution so far.

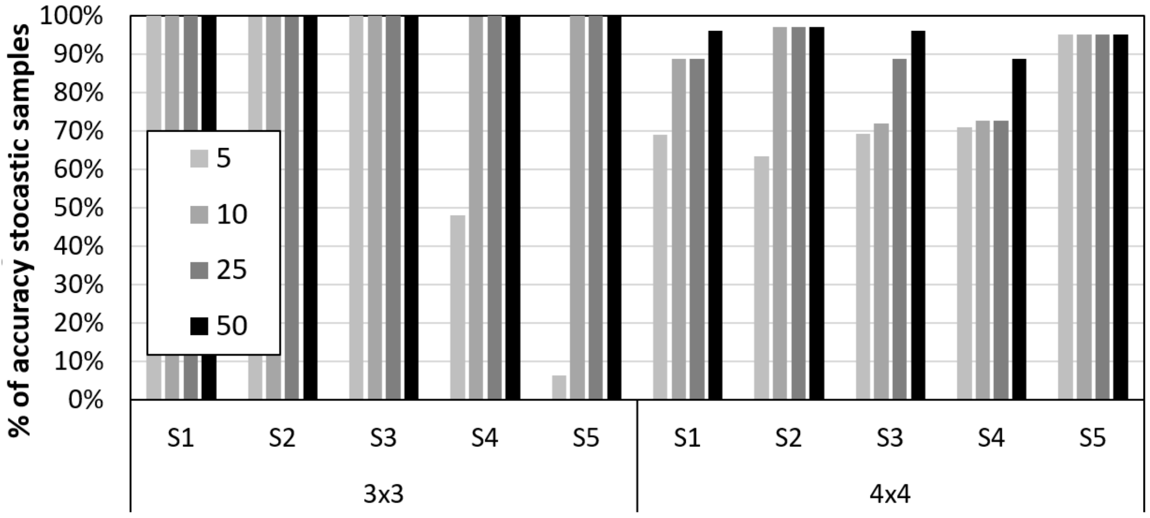


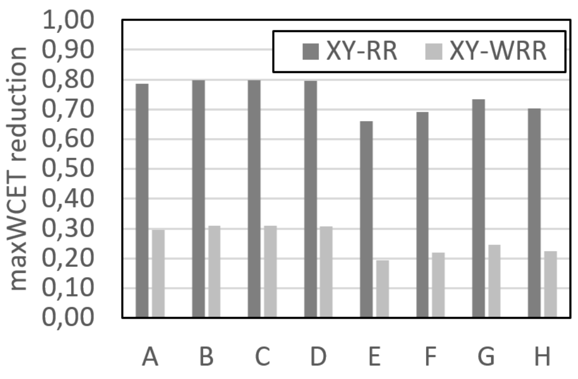
Figure 5.7: Normalized maxWCET of random routing w.r.t the best routing.

We have performed two experiments for 3x3 and 4x4 respectively. In both cases, we take 5 samples (S1 to S5), each of which with just 5, 10, 25 and 50 of all possible routings, i.e. 512 for 3x3 and 65,536 for 4x4. Across all five samples, we can see that the maxWCET results obtained with random routing are very close to those obtained with the best routing with samples of 50 observations. In the case of 3x3, sample sizes of 10 already find optimal results. For 4x4, samples for size 10 get on average a solution 85.2% optimal. With samples of 50, the average is 94.6% and the worst case 88.7%. With samples of 100 (not shown in the plot), the average is 98.6% and with samples of up to 330 (0.5% of the population), the optimal solution is found in the 5 samples.

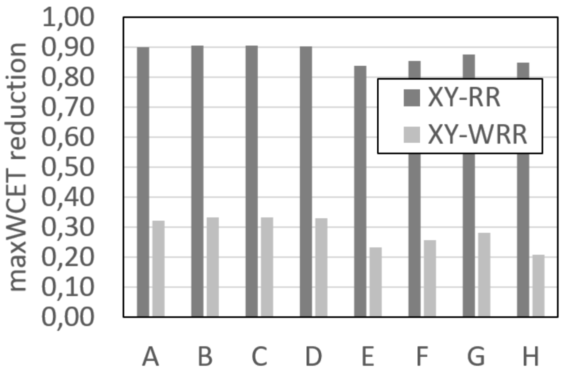
5.4.4 Homogeneous Workloads

For homogeneous workloads, the optimization goal we set for NoCo is reducing the WCET of the thread with the longest WCET (maxWCET). For parallel applications using coarse-grain parallelism, the thread with the lowest performance is usually the one determining the WCET of the application and thus, optimizing maxWCET minimizes the overall WCET of the parallel applications.

It is also worth mentioning that for homogeneous workloads thread mapping plays no role since all threads are identical. Hence, gains come from walloc and routing optimization only.



(a) 9 threads.



(b) 16 threads.

Figure 5.8: rILP(m,w) maxWCET results for 9- and 16-thread applications.

Figure 5.8(a) and Figure 5.8(b) compare the WCET reduction of the full NoCo optimization, i.e. $rILP(m,w)$, against the reference $XY-RR$ and $XY-WRR$ designs for 9-thread and 16-thread applications, respectively. For those applications we map to a 3x3 and 4x4 mesh NoCs respectively.

For 9-thread workloads, $rILP(m,w)$ achieves average 74% WCET reductions w.r.t. $XY-RR$ and 26% w.r.t. $XY-WRR$. The rationale behind these results is as follows:

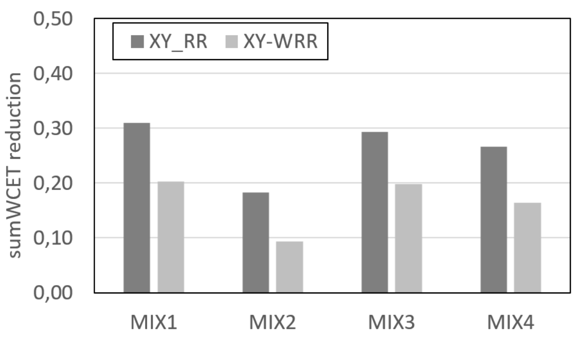
- WCD values for $XY-RR$ are highly heterogeneous and hence, the WCD experienced by the thread at the core with highest contention is far higher than that of most of the other threads.
- By using WRR, WCD values become more homogeneous, thus significantly decreasing the opportunities for optimization in the context of homogeneous workloads. Yet, even in this case, $rILP(m,w)$ decreases maxWCET significantly (26% on average).

Results across workloads show that those with higher access frequencies obtain higher benefits with NoCo, since the impact of WCD on their WCET is higher. Still, we observe that decreasing NoC requirements by 4x only decreases gains from 26% to 22%, thus showing the importance of optimizing NoC configuration to improve performance.

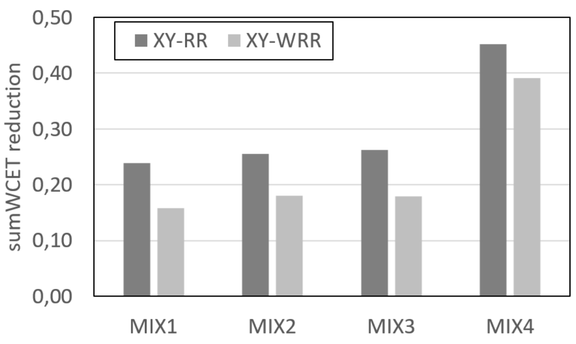
For 16-thread applications maxWCET reduction follows the same trend, though improvements are more noticeable. In particular, $rILP(m,w)$ decreases maxWCET by 88% and 29% on average w.r.t. $XY-RR$ and $XY-WRR$ respectively, with increased gains occurring consistently across all workloads.

5.4.5 Heterogeneous Workloads

For heterogeneous workloads, NoCo focuses on reducing sumWCET, hence actually reducing the overall impact – and wasted resources due to contention. While in the experiments in this section, NoCo optimizes all three NoC parameters to reduce sumWCET with unrestricted per-task WCET, our formulation supports setting specific bounds to the WCET for some tasks. Unlike homogeneous workloads, for heterogeneous ones, (task) mapping plays a role in optimizing performance – as it was analyzed in Section 5.4.2.



(a) 9 threads.



(b) 16 threads.

Figure 5.9: rILP(m,w) sumWCET results for 9- and 16-task workloads.

Figure 5.9 shows the sumWCET reduction of NoCo with respect to $XY - RR$ and $XY - WRR$ designs for 9-task workloads (Figure 5.9(a)) and 16-task workloads (Figure 5.9(b)).

For the 9-task workloads, $rILP(m,w)$ we observe pretty consistent improvements. In particular, with respect to $XY - WRR$ improvements are similar to those obtained for homogeneous workloads ranging from 17% to 22% (19% on average). We also see that for the heterogeneous workloads, the results of RR are not as bad as for the homogeneous. Yet NoCo improves $XY - RR$ by 30% on average. Results for the 16-task workloads support that $rILP(m,w)$ achieve consistent reduction w.r.t. the other two techniques.

5.4.6 Other Metrics

In previous sections, we have used two optimization criteria sumWCET and maxWCET. NoCo also supports other criteria such as, for example, limiting the maximum WCET/WCD of tasks. This is better illustrated with an example that uses the homogenous 9-thread workload A that we run in a 3x3 network under a fixed mapping. In a first experiment we run NoCo minimizing sumWCET. This produces the WCET shown in the upper part (1) of Table 5.4. In a second experiment we run NoCo, still minimizing sumWCET, but also enforcing that τ_8 cannot exceed 11.56 (million cycles). The result of this experiment is shown in the lower part (2) of Table 5.4. As it can be seen, the WCET of τ_8 is indeed 11.55, which is achieved by maintaining its WCD below 3.3. As a side effect, we observe a small increase in sumWCET, from 87.9 to 88.2.

Table 5.4: Effect of limiting the WCET of one thread. *ET* stands for execution time in isolation. Time show in million cycles.

		τ_0	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
1	ET	4.99	5.81	6.63	5.81	6.63	7.44	6.63	7.44	8.26
	WCD	2.27	2.89	3.57	2.56	2.97	3.83	2.89	3.28	3.98
	WCET	7.26	8.70	10.20	8.37	9.59	11.28	9.51	10.72	12.24
2	ET	4.99	5.81	6.63	5.81	6.63	7.44	6.63	7.44	8.26
	WCD	2.55	2.93	3.80	2.56	3.18	3.86	3.07	3.29	3.29
	WCET	7.54	8.74	10.43	8.36	9.80	11.30	9.69	10.73	**11.55**

5.5 Conclusions

In this chapter, we have analyzed the impact that wNoC configuration parameters' have on the WCD and thus to the WCET estimates in multicore and manycore systems. While several proposals focus on minimizing wNoC parameters impact in WCET individually or in pairs, in this chapter we have shown that the optimal setup only can be achieved when optimizing mapping, routing and walloc with arbitration at the same time. To that end, we have presented NoCo, a framework that leverages the configuration potentials of wNoCs via a hybrid stochastic/ILP approach. NoCo finds the best routing configurations with a stochastic approach and applies ILP to the generated routing policies to find the optimal mapping and weights allocation for each of them. We show that NoCo achieves significant improvements over other optimization strategies that focus on just a subset of the NoC parameters. NoCo outperforms reference XY-RR and XY-WRR wNoC designs for both heterogeneous and homogeneous workloads, and also in the context of parallel and single-thread workloads.

Part III

Contention Aware Optimizations in Validation stage

Chapter 6

Accurately Measuring Contention in Mesh NoCs

6.1 Introduction

The trend towards more autonomous software-centric functionalities is on the rise in CRTES in relevant industrial domains such as automotive. Major CRTES industry players have adopted (or are on the way to doing so) small multicores, such as the Infineon AURIX family [137] in automotive and the Xilinx Zynq UltraScale+ [138] in avionics, which include few cores (e.g. 3 to 6). For the interconnection, manycores build on NoCs as they provide good scalability and high flexibility to set appropriate topologies, routing algorithms, arbitration policies, etc. Multiple processors targeting CRTES already deploy NoCs (e.g. to connect 10 to 20 nodes), such as the Kalray MPPA 256 SoC [7] (e.g. a 4x4 mesh) and the Xilinx VERSAL [26].

The other side of the coin is that hardware-shared resources in general, and NoCs in particular, cause the timing of an application (and the bounds derived to it) to depend on the activity of its co-runner tasks, i.e. their usage of shared resources. When considering NoCs, several solutions bounds to the worst-case contention in NoCs [52, 55, 60] classify contention into direct or indirect depending on whether contending flows share resources over their paths in the NoC or not [61]. However, these methods aim at deriving upper bounds to contention, needed during verification, rather than tracking the actual contention during the testing phase, needed for validation, and hence, they cannot be used for validation/testing purposes.

In this work, we contend that the ability to accurately track the contention a task suffers in each node of a NoC-connected manycore by each other CT is instrumental to validate the timing behavior of CRTES to a sufficient extent and properly diagnosing software timing overruns. Overruns can otherwise go unnoticed if we just track the cumulative contention a task suffers by all its CTs. More in detail, accurately tracking per-node and per-task contention brings the following advantages:

- *Detection.* It allows detecting situations in which a given task incurs longer contention than expected by a contender task, even if this effect is hidden or compensated by another contender task causing less contention than expected.

This anomalous behavior, which does not arise just by analyzing end-to-end timings [69], must be detected during testing.

- *Correction.* It allows determining the point of the mesh where the contention occurs and the tasks' contribution to this contention, which is also fundamental to propose and apply corrective measures in case of detected misbehavior.

In this line, the goal of this work is to set the foundations of a fine-grained *contention tracking* approach that aims at capturing the actual contention tasks generate to each other in a wmesh, as a building block for the timing validation for wmesh NoCs. Our contribution develops in the following three axes:

(a) We define a golden metric called *PairWise Contention (PWC)* that captures the slowdown the packets generated by the TuA suffer when accessing the wmesh due to packets from the Contender Tasks (CTs). The main challenge in deriving PWC emanates from the distributed nature of the wmesh NoCs that causes contention to happen in different nodes (locations), as opposed to centralized interconnects where all contention occurs at a single location. For wmesh, simple ways exist to trace contention information locally in each router. That information only reflects the packet (including its source core) that stalls another packet. However, it does not reflect whether the blocking packet is effectively causing such contention or is, in turn, blocked by another packet. Ascribing all contention suffered by the TuA only to packets arbitrated in that same router leads to incorrectly ascribing contention effects to the contenders. Instead, in order to effectively capture the source of the contention, PWC defines local and remote contention that is to be applied to each pair of tasks running.

(b) We define and formulate for the first time a *Golden Reference Value (GRV)* for PWC. GRV is a criterion to derive the local and remote PWC components for tasks running in a wmesh-centric processor. GRV fairly ascribes the contention the TuA suffers in the wmesh to its CTs. GRV builds around the idea of ascribing contention experienced by an analyzed packet to the actual contending packet causing the contention, whether it shares (local) or not (remote) nodes with the packet causing contention. GRV is complete, meaning that it is able to classify all types of contention packets may suffer, distinguishing the source and location of each contention case.

(c) We propose a particular implementation of GRV via an offline method for timing validation. For each test, our method processes execution traces of a set of tasks executed on a wmesh-centric multicore to break down the contention each task suffers in each router. We assess the effectiveness of GRV in controlled scenarios (including a variety of wmesh sizes and setups) in which contention can be ascribed to the cores issuing packets. Our method, for every single cycle of contention experienced by a packet, identifies the core that issued the packet ultimately causing such contention, the router where contention occurred, and hence, whether such contention is local or remote. Finally, we also assess the scalability of the proposed approach to large NoCs (e.g. 5x5 and 6x6 meshes), which are already larger than those NoCs in current and evaluated COTS manycores for CRTES [7, 26].

6.2 Defining Pairwise Contention (PWC)

Previous works in the literature have focused on providing bounds to NoC contention, but they provide no information about the actual NoC contention tasks suffer. This is better illustrated with an example. We model a 2x2 2Dmesh NoC with XY routing, see Figure 6.3, in which all cores target the same memory controller located in the right output port of R_3. For simplicity, we assume that all routers have one virtual channel, ports have a 2-entry queue that can store two packets and traversing a NoC link and a router takes 1 cycle.

The left bar in Figure 6.1 shows the WCET estimate for τ_A that sends 1000 packets to the NoC derived assuming that the other three tasks are larger than τ_A so that they send packets at their maximum rate during τ_A execution time. The WCET estimate is derived as the addition of the execution time in isolation of τ_A (i.e. with no contention), $Tiso$, and the WCD derived for each packet [61, 51]. In this case, $Tiso = 7000$ cycles and WCD varies for each contending task based on the core in which they run and the path followed to reach the memory controller $WCD_B = 2000$ cycles, $WCD_C = 5000$ cycles, $WCD_D = 5000$ cycles. So that WCD and WCET are derived analytically and built on information about the contention τ_A packets suffer in each link, buffer and routing information, and the number of requests each contenting task sends.

For the purpose of illustration, we then run τ_A in three different scenarios: S1, S2, and S3. In all of them, the number of requests each task generates is the same 1000, but the tasks take different times to execute in each experiment. Bars labeled S1, S2, and S3 in Figure 6.1 show the actual contention suffered by τ_A from the other tasks in each scenario. In S1 requests from τ_A and τ_D do not overlap in time so the latter does not generate any contention on τ_A, while τ_B and τ_C only partially overlap. In S2 the situation is similar with the exception that τ_B and τ_A request do not collide and the request from τ_C and τ_D only partially collide with τ_A. Finally, in S3, tasks send requests to the NoC so that they do not collide with each other.

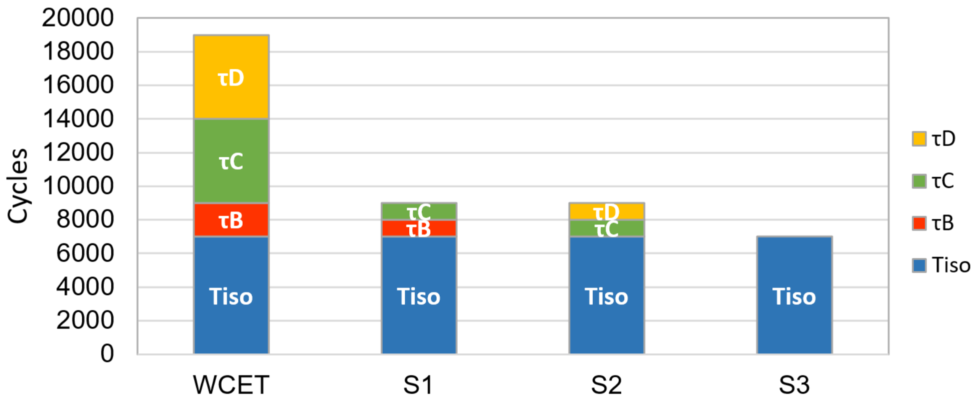


Figure 6.1: Comparison between worst-case and observed contention CTs generate on τ_A under scenarios S1, S2 and S3 as introduced early in this Section.

Hence, τ_A's WCET analytical computation that builds on WCD (bounds) caused by τ_A worst potential arbitration packets alignment does not reflect the actual contention suffered by the packets of τ_A in scenarios S1 to S3 nor correctly determines which tasks are creating more contention to τ_A. In fact, the WCD and WCET derived for τ_A are only valid as a global contention upperbounds for all three scenarios but not as a method to bound the maximum contention contribution each task can create to τ_A. The actual contention τ_A suffers depends on how requests align with the requests of other tasks, which varies across experiments. Hence, techniques deriving bounds to contention (WCD) cannot be used to measure the actual contention tasks suffer.

That is so because bound-based approaches, in general, assume certain traffic conditions (e.g. worst-case packets alignment with a certain packets arrival distribution) that may not reflect the real traffic behavior in execution. However, as they do not analyze fine-grain real traffic from a run, these techniques cannot provide any feedback other than whether the final WCET is accomplished or not (not reliable in validation). Similarly, bound-based approaches that rely on static arbitration policies (e.g. round-robin) to compute the shares between tasks (i.e. bandwidth distribution), cannot capture bandwidth distribution variation over time in tasks' runs. This can occur, for instance, when one or more tasks of the system do not use the entire bandwidth assigned by the arbitration policy used. In that case, the remaining unused bandwidth is distributed among the NoC and can be used by the other CTs. That can lead to a scenario where a task generates more contention to other tasks than the expected one assuming a computed bandwidth distribution independently if it exceeds or not the global WCET. In these kinds of scenarios, only fine-grain packet analysis methods such as PWC and GRV can detect and provide precise information useful for validation and verification purposes.

Detailed information about actual contention suffered by a task in the mesh allows detecting any unexpected timing behavior during the validation phase, even if no deadline violation occurs but tasks individually exceed their quota (i.e. the contention they are expected to cause on the TuA). This may happen, for example, when contention caused by τ_B on τ_A exceeds its estimated bound, but the total contention caused by all contenders on τ_A happens to stay within the admitted threshold. Such diagnosis information also helps to identify the root cause of a timing misbehavior during operation and promptly react by applying the appropriate safety measure like switching to a different precomputed task-to-core mapping or adjusting the interconnect configuration [139]. Safety standards and reference documents like CAST-32A and A(M)C 20-193 in avionics advocate identifying each *interference channel* and provide evidence that it has been removed or mitigated. This cannot be easily achieved with end-to-end measurements and requires per-resource (interference channel) contention tracking.

Actual contention bounds can also be used as additional evidence that the derived contention bounds are correct since small changes in the configuration in the NoC can invalidate the derived bounds. To that end, benchmarks generating high load on the network are executed against reference applications to compute the observed contention and the theoretical bound.

In both cases, detailed information about actual contention can provide accurate diagnostics in specific (and relevant due to causing overruns) scenarios that are unlikely to be easily reproducible due to the difficulties to control application execution at a sufficiently fine grain. Hence, overruns may easily occur sporadically and mechanisms to diagnose the causes without needing re-execution, which may not reproduce the overrun, become of prominent importance. However, the challenge lies in determining the information that is required and how to combine it to produce a metric that captures the actual contention that tasks generate on each other.

6.2.1 Centralized interconnects

For centralized interconnects, like buses, PWC can be defined as the contention a request from a core (master) C_B causes on a request from another core C_A. It can be derived as the time interval in which C_B is granted access to the interconnect and C_A has its request signal active. Hence, in centralized interconnects, contention-related information is available in a single location and, since contention occurs locally in the centralized interconnect, reasoning about the cause and effects of contention ('who' causes it and 'who' experiences it) is relatively simple.

Intuitively, PWC for wmesh could be defined as for centralized interconnections by applying it locally in each router. As an input port can be shared by multiple packets belonging to different flows from different cores, the contention is tracked per packet and classified per flow. The rule to account for contention is relatively straightforward: every cycle a given packet P_j from F_j (terms are defined in Table 6.1) in a given input port in R_n is granted access to the output port during the header flit arbitration, the other packets (e.g. P_i and P_k from F_i and F_k) that lose the arbitration are accounted for an additional cycle of contention, see Figure 6.2 left router (R_n). This contention is also accounted for while P_j is using the output port during the transmission of body flits, and the flits in the other input ports, e.g. P_i and P_k, or the PME are waiting to get access to the same output port. In both scenarios, every cycle $cont_{j \triangleright i}^{R_n}$ and $cont_{j \triangleright k}^{R_n}$ are incremented[1].

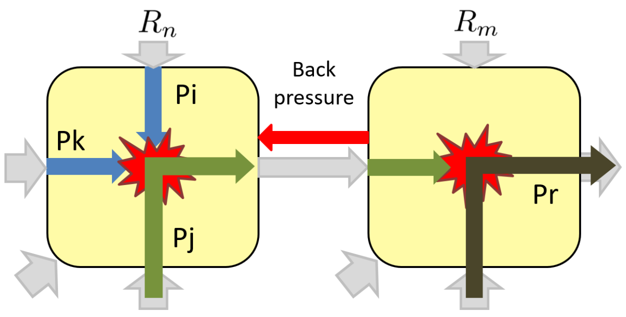


Figure 6.2: Backpressure example.

[1]Note that, a flow is composed of packets, and each packet is composed of flits. However, in several discussions in this chapter, some of these terms can be used indistinctly. For instance, a (header) flit may suffer contention, but we can also state that the packet suffers contention.

Table 6.1: Definitions used in this contribution.

Term	Definition
$\mathcal{R}$	Set of $N \times M$ routers in the mesh
R_n	A router in $\mathcal{R}$
F_i	Stream of packets (Flow) traversing the same route
P_i	A generic packet belonging to flow F_i
PTT_{P_i}	Packet Traversal Time of a packet P_i
$PTT_{P_i}^{R_n}$	Packet Traversal Time of a packet P_i in router R^n
FTT_{F_j}	Flow Traversal Time of all packets in Flow F_j
$\widehat{H}_i$	Ordered set of R^n routers that define F_i's path
$zll_{P_i}^{R_n}$	Traversal time of packet P when traversing router R_n without contention
$cont_{P_i}^{R_n}$	Contention a packet P_i suffers due to all other contending packets in router R_n
$cont_{j \triangleright i}^{R_n}$	Contention a packet P_i in suffers due to a packet P_j in router R_n
$cont_{\tau_x}^{R_n}$	Contention task τ_x suffers from all other tasks in R_n
$cont_{\tau_x}^{lrc}$	Local Router contention task τ_x suffers due to all other tasks
$cont_{\tau_x}^{rrc}$	Remote Router contention task τ_x suffers due to all other tasks
$cont_{\tau_x}^{lrc,Rn}$	Local Router contention task τ_x suffers due to all other tasks in R_n
$cont_{\tau_x}^{rrc,Rn}$	Remote Router contention task τ_x suffers due to all other tasks in R_n
$lrc_{P_i}^{R_n}$	Local Router Contention a packet P_i suffers due to all other contending packets in router R_n
$rrc_{P_i}^{R_n}$	Remote Router Contention a packet P_i suffers due to all other contending packets in router R_n
$lrc_{j \triangleright i}^{R_n}$	Local Router Contention F_i suffers due to F_j in router R_n
$rrc_{j \triangleright i}^{R_n}$	Remote Router Contention F_i suffers due to F_j in router R_n
$cont_{\tau_i}$	Total contention suffered by τ_i
$cont_{\tau_j \triangleright \tau_i}$	Total contention τ_i suffers due to τ_j

However, this approach that considers local router information only fails to capture the contention caused due to propagated backpressure, which is common in wmeshes. It arises, for instance, when a packet P_j in the output port under analysis in R_n cannot access R_m because it is busy. For instance, in Figure 6.2, P_i and P_k are delayed by P_j who wins the arbitration for X+ in R_n but awaits to win X+ arbitration in R_m. Propagated backpressure can happen when the output port in R_m that P_j is willing to use is occupied by another packet P_r. In this scenario, if we only use local router information, $cont_{j \triangleright i}^{R_n}$ and $cont_{j \triangleright k}^{R_n}$ in R_n would be incremented every cycle P_i and P_k are stalled because P_j cannot be transmitted due to backpressure from P_r in the following router. However, in reality, P_j is not ascribable for stalling P_k and P_i, which are instead stalled in R_n because of P_r's backpressure in R_m. Accordingly, if we consider the state beyond the local router, $cont_{r \triangleright i}^{R_n}$, $cont_{r \triangleright k}^{R_n}$ and $cont_{r \triangleright j}^{R_n}$ should be updated instead as P_r is the 'guilty' packet that prevents P_i, P_k and P_j to traverse R_n.

6.2.2 PWC for NoCs

We illustrate PWC for NoCs via the scenario depicted in Figure 6.3 in which four tasks (τ_A-τ_D) run in a 2x2 wmesh-connected multicore, with τ_A being the TuA and τ_B, τ_C and τ_D the CTs. τ_A runs in (the core at) R_0, τ_B in R_1, τ_D in R_2 and τ_D in R_3.

For this example, let us assume that the upper gray area in the left bar of Figure 6.4 represents the cumulative contention experienced by the packets of τ_A (the bottom stripped area is the time in isolation of τ_A). The contention part of the τ_A execution time is incremented every cycle a packet of τ_A is stalled by a contending packet of a different task, whether the other contending packet is either in the same router as the stalled packet of τ_A (local contention) or in another router propagating contention through backpressure (remote contention).

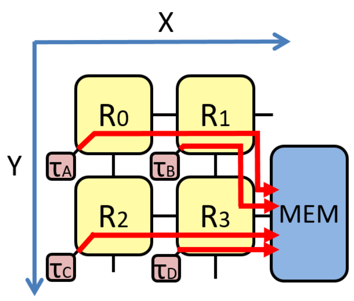


Figure 6.3: 2x2 2Dmesh XY-routing setup.

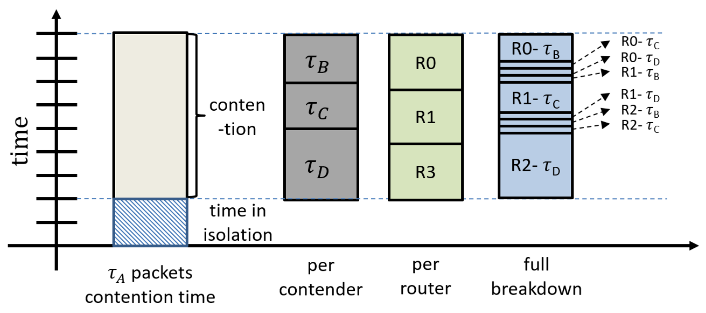


Figure 6.4: Synthetic example breaking down of contention into its PWC components.

The contention time that τ_A's packets experience can be broken down following different criteria, as shown in the different bars in Figure 6.4. From left to right, (i) per contender task delaying τ_A in the wmesh; (ii) per router where τ_A suffers contention; and (iii) a combination of both, i.e. per router and contending task. Section 6.3 details how the information gathered by PWC and GRV allows producing those and many other breakdowns. The latter breakdowns let us understand that τ_A is suffering contention mainly in R0, R1, and R3 by τ_B, τ_C, and τ_D, respectively, capturing the requirements for validation described in previous sections.

Overall, to properly capture propagated backpressure our proposed PWC differentiates between local contention and remote contention.

- *Local router contention* (lrc) is experienced by a packet P_i in one of the routers R_n to its destination, i.e. $R_n \in \widehat{H_i}$ where its header flit is and the contention (guiltiness) can be ascribed to a packet P_j that gains the arbitration or is traversing one of the output ports in R_n.
- *Remote router contention* (rrc) is experienced by a packet P_i in one of its routers R_n and can be ascribed to propagated backpressure in another router in the mesh. Hence, the contention is not ascribable to any other packet in the same router (which would instead fall into the lrc category). As a distinguishing factor, in this scenario, the contention guiltiness cannot be ascribed within the same router R_n where the contention takes place, but is to be assigned to another packet P_j in one of its routers $R_m \in \widehat{H_j}$.

Both lrc and rrc are identified as part of PWC and defined per pair of tasks (i.e. a task causes lrc and/or rrc on another task). Also, the rrc suffered by F_i can be further broken down per router, allowing to identify where contention originates.

It is worth noting that the same breakdown principle we apply to local/remote contention can be applied to direct/indirect contention classification [61, 55]. That is, our proposal is transversal to both. Direct/indirect contention has been proposed to classify the contention that packets suffer from other flows depending on whether those other flows share or not physical resources in their paths with the flow of the TuA. Beyond the fact that direct/indirect classification has been used so far only for contention estimates or upper bounds (i.e. expected maximum contention), it can also be applied to measured (observed) contention. However, in our view, the local/remote classification is more naturally applied as it provides insightful information about where and who delayed the packets of a given task's flow, rather than capturing whether the flow delaying the TuA's flow shares or not routers with it. Hence, and without loss of generality, during the rest of this work, we apply our approach to local/remote contention.

It is also worth mentioning that contention tracking approaches can be applied to any wmesh NoC configuration that uses deterministic routing setups whereas techniques to derive NoC contention bounds typically require more specific configurations so as to contain NoC contention estimates. That is, under an unsatisfactory NoC configuration for which no bounds can be derived, contention tracking allows determining, for a specific run, how tasks affect each other in the NoC. For a NoC configuration for which bounds can be derived, contention tracking allows validating the bounds derived for each flow/task and assessing how far the real case is from the worst case. In that line, while our solution is not restricted to wmesh NoCs, they offer a favorable application scenario because it has been shown that tight bounds can be produced for specific configurations thereof.

6.3 Defining a Golden Reference Value (GRV)

Building on PWC, we define a GRV that correctly captures the sources of contention in a mesh NoC. GRV aims at enabling effective diagnosis of the root causes of potential timing task violations, as well as identification of individual contention bounds for tasks. Moreover, tailoring PWC metrics to specific COTS multicores where monitoring support is limited requires a reference value to assess their accuracy. GRV fills this gap by allowing the comparison of the specific PWC metric for such a COTS processor against GRV (e.g. in a timing simulator), thus allowing to tune of the PWC metric. For the definition of GRV, we identify several properties that a reliable GRV to breakdown contention in wmesh NoCs must exhibit.

(a) *Completeness.* The criterion must classify as contention all the additional time, w.r.t. to isolation time, that each packet of τ_A needs to traverse the NoC due to interaction with its CTs.

(b) *Source and accuracy of the contention.* The criterion must be capable of identifying the packets causing contention on any given packet of task τ_A, thus allowing to know where contention occurred (router), what task caused it, and whether it was lrc or rrc. This per-packet information can be aggregated to have per-task figures.

In order to show that our proposed GRV achieves these goals, we perform an analysis at packet level, where the source of each contention cycle can be singled out unequivocally. We focus on the NoC contention. Thus, the contention that occurred in other shared resources is not considered in the analysis. Note that, in this chapter, we implicitly use the term *time* to refer to a discrete number of cycles, putting aside any further consideration about the duration of each cycle (i.e. as if all the analysis was performed under constant operating frequency). Considering cycles of different duration would require expanding each term in each formula into as many terms as potential cycle durations were possible, which would be against the clarity of our already complex formulation.

We define the Packet Traversal Time of a packet P_i, denoted as PTT_{P_i}, as the cumulative time spent by P_i in all routers $R_n \in \widehat{H_i}$ it traverses. Accordingly, we define the Flow Traversal Time of a flow F_x, denoted as FTT_x, as the addition of the traversal time of all packets $P_i \in F_x$.

$$FTT_x = \sum_{P_i \in F_x} PTT_{P_i} = \sum_{P_i \in F_x} \sum_{R_n \in \widehat{H_i}} PTT_{P_i}^{R_n} \tag{6.1}$$

More in detail, $PTT_{P_i}^{R_n}$ is the result of the packet P_i traversal time when traversing router R_n without contention, also called Zero Load Latency ($zll_{P_i}^{R_n}$), plus the increased traversal time due to other packets contention ($cont_{P_i}^{R_n}$).

$$PTT_{P_i}^{R_n} = zll_{P_i}^{R_n} + cont_{P_i}^{R_n} \tag{6.2}$$

On the one hand, $zll_{P_i}^{R_n}$ depends on the router architecture implementation (e.g. number of pipeline stages). It can be obtained measuring the time a packet P_i takes to traverse router R_n in isolation[2]. On the other hand, $cont_{P_i}^{R_n}$ is determined by the interference caused by other packets on packet P_i in router R_n. As introduced in Section 6.2.2, we classify the contention a packet P_i suffers in router R_n into two different types of contention:

- local router contention ($lrc_{P_i}^{R_n}$) when the packet causing a delay to P_i is in router R_n.
- remote router contention ($rrc_{P_i}^{R_n}$) when the packet delaying P_i is in another router when P_i suffers contention in router R_n.

[2]In many NoCs, $zll_{P_i}^{R_n} = zll_{P_j}^{R_m}$ for all packets P_i and P_j of any flow, and all routers R_n and R_m in the NoC, so we could refer simply to zll, but we keep $zll_{P_i}^{R_n}$ for the sake of generality.

Accordingly, we can define $cont_{P_i}^{R_n}$ as follows:

$$cont_{P_i}^{R_n} = lrc_{P_i}^{R_n} + rrc_{P_i}^{R_n} \tag{6.3}$$

In order to fulfill the identified mandatory properties on the PWC metric (completeness, source, and accuracy of the contention), we bound the classification of lrc and rrc for a given packet $P_i \in F_x$ in a router R_n to a specific time (cycle) t, where $t \in stalled(P_i, R_n)$. Note that $stalled(P_i, R_n)$ is the set of cycles when P_i is stalled in R_n and its cardinality is $|stalled(P_i, R_n)| = PTT_{P_i}^{R_n} - zll_{P_i}^{R_n}$. That is, $stalled(P_i, R_n)$ is the P_i traversal time of router R_n minus the zero load latency. It is worth noting that $cont_{P_i}^{R_n}$ and $stalled(P_i, R_n)$ terms are closely related. The former one identifies the cumulative effect of contention that P_i suffers in R_n whereas the second one is used to identify, in the formulations, the specific set of cycles where the contention happens. For example, $stalled(P_i, R_n) = \{3, 4, 5\}$ contains the set of cycles where the contention occurs and $cont_{P_i}^{R_n} = 3$ contains the count of these cycles.

For each $t \in stalled(P_i, R_n)$, there exists exactly one guilty packet $P_j \in F_y$ that causes such cycle of contention on P_i.

We can define $cont_{P_i}^{R_n}(t)$ with the lrc and rrc parameters, i.e. the contention a packet P_i suffers in a router R_n at time t (Eq. 6.4) with respect to a guilty packet P_j (Eq. 6.5), as follows:

$$cont_{P_i}^{R_n}(t) = \sum_{P_j} cont_{j \rhd i}^{R_n}(t) \tag{6.4}$$

$$cont_{j \rhd i}^{R_n}(t) = lrc_{j \rhd i}^{R_n}(t) + rrc_{j \rhd i}^{R_n}(t) \tag{6.5}$$

where $lrc_{j \rhd i}^{R_n}(t)$ and $rrc_{j \rhd i}^{R_n}(t)$ represent the PWC unfolding of the local and remote contention terms in Eq. 6.3. Note that, for a given $t \in stalled(P_i, R_n)$, exactly one of the two terms in Eq. 6.5 is one and the other is zero for packet P_j. Those terms are both zero for any other packet different from P_j. We can model the cumulative contention suffered by a task τ_x building on FTT_x as follows:

$$cont_{\tau_x} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} cont_{P_i}^{R_n}(t) \tag{6.6}$$

We can build on Eq. 6.4 and 6.5 to narrow the scope of Eq. 6.6 to model the PWC suffered from τ_x because of τ_y (per contender breakdown in Figure 6.4):

$$cont_{\tau_y \rhd \tau_x} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} \sum_{P_j \in F_y} \left(lrc_{j \rhd i}^{R_n}(t) + rrc_{j \rhd i}^{R_n}(t)\right) \tag{6.7}$$

By restricting Eq. 6.6 and 6.7, we can also obtain different contention breakdowns, such as those shown in Figure 6.4 or combinations thereof, for instance considering only lrc or rrc, considering only a given router R_n, or considering only a given contender τ_y. In particular, we can model the cumulative contention suffered by a task τ_x per router R_n, denoted as $cont_{\tau_x}^{R_n}$, building on Eq. 6.6 as follows (per router breakdown in Figure 6.4):

$$cont_{\tau_x} = \sum_{R_n \in \widehat{H_i}} cont_{\tau_x}^{R_n} \tag{6.8}$$

$$cont_{\tau_x}^{R_n} = \sum_{P_i \in F_x} \sum_{t \in stalled(P_i, R_n)} cont_{P_i}^{R_n}(t) \tag{6.9}$$

where $cont_{P_i}^{R_n}(t)$ is the PWC suffered by each packet P_i of flow F_x from all other tasks unfolded per router R_n.

Similarly, we can model the cumulative contention suffered by a task τ_x per contention type, denoted respectively as $cont_{\tau_x}^{lrc}$ and $cont_{\tau_x}^{rrc}$, as follows:

$$cont_{\tau_x} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} \left(lrc_{P_i}^{R_n}(t) + rrc_{P_i}^{R_n}(t) \right) \tag{6.10}$$

$$cont_{\tau_x}^{lrc} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} \left(lrc_{P_i}^{R_n}(t) \right) \tag{6.11}$$

$$cont_{\tau_x}^{rrc} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} \left(rrc_{P_i}^{R_n}(t) \right) \tag{6.12}$$

where $lrc_{P_i}^{R_n}(t)$ and $rrc_{P_i}^{R_n}(t)$ are respectively the amount of local and remote router contention each packet P_i from a flow F_x suffers from all other packets in router R_n in cycle t.

The finest grain cumulative contention analysis breakdown using the 3 parameters, namely task, type of contention and router, can be obtained building on Eq. 6.10 in the following manner:

$$cont_{\tau_x}^{lrc,R_n} = \sum_{\substack{P_i \in \widehat{F_x} \\ t \in stalled(P_i, R_n)}} \left(lrc_{P_i}^{R_n}(t) \right) \tag{6.13}$$

$$cont_{\tau_x}^{rrc,R_n} = \sum_{\substack{P_i \in \widehat{F_x} \\ t \in stalled(P_i, R_n)}} \left(rrc_{P_i}^{R_n}(t) \right) \tag{6.14}$$

where $lrc_{P_i}^{R_n}(t)$ and $rrc_{P_i}^{R_n}(t)$ are respectively the lrc and rrc contention suffered by each packet P_i of F_x in R_n in cycle t.

6.3.1 Defining lrc

For the sake of clarity, in this section, we use P_i to refer to the packet under analysis, and P_j and P_k to other packets in the same router.

Table 6.2: Functions used in this chapter.

Term	Function
$stalled(P_i, R_n)$	Given a packet P_i in a router R_n it returns the cycles where P_i is stalled suffering contention in R_n.
$PH(P_i, R_n, t)$	Given a packet P_i in a router R_n at time t, it returns the packet P_j that is the packet at the head of P_i input port.
$PSO(P_i, R_n, t)$	Given a packet P_i in a router R_n at time t, it returns the packet P_j that is targeting the same output port P_i targets, and granted permission to move forward (e.g. to the next router), if any.
$HN(P_i, R_{n+1}, t)$	Given a packet P_i at router R_n in time t, it returns the packet P_j that is the head packet of the targeted input port of P_i in the next router R_{n+1}.
$SP(P_i, R_n, t)$	Given a packet P_i at router R_n in time t, it returns the packet P_j that causes contention on P_i in a router R_m different to R_n due to backpressure. Such packet P_j is found following the procedure described in steps (S3a), (S3b) and (S3c) detailed in Section 6.3.2.

Lrc is defined over the set of packets that are ready to be arbitrated, and it identifies the contention a given packet $P_i \in F_i$ suffers due to the arbitration of another packet $P_k \in F_k$ in the same router R_n. A packet is considered 'ready to be arbitrated' whenever it could leave the router in a no-contention scenario.

We identify two scenarios. S1 captures lrc when P_i is ready to be arbitrated and is waiting for another packet P_k to traverse its targeted port (i.e. P_i loses the arbitration). S2 considers lrc when P_i is not the first packet of an input port and there is at least one packet P_j in that input port that is currently suffering lrc contention due to another packet P_k in another input port. As in this section we deal with lrc, all scenarios take place in the same router R_n. Note that we build on the definitions of $PH()$ and $PSO()$ in Table 6.2.

(S1) At a given time t in router $R_n \in \widehat{H_i}$, a packet P_i is queued and ready to be arbitrated at the *head* of its corresponding input port ($P_i = PH(P_i, R_n, t)$). P_i suffers contention from another packet P_k that is currently traversing the same target output port as P_i, i.e. $\exists P_k | P_k = PSO(P_i, R_n, t)$. Therefore, $lrc^{R_n}_{k \triangleright i}(t) = 1$.

(S2) At a given time t, a packet P_i is not the *head* of its corresponding input port in $R_n \in \widehat{H_i}$ such that $\exists P_j = PH(P_i, R_n, t)$, with $P_j \neq P_i$. If P_j is granted access to traverse the output port, denoted $P_j = PSO(P_j, R_n, t)$, then $lrc^{R_n}_{j \triangleright i}(t) = 1$. Alternatively, P_j could be in turn delayed by another packet P_k from another input port that is granted access to traverse P_j's output port, denoted $P_k = PSO(P_j, R_n, t)$. In this case, P_k is the one causing contention, i.e. $lrc^{R_n}_{k \triangleright i}(t) = 1$.

Note that when $P_i = PH(P_i, R_n, t)$ and $P_k = PSO(P_i, Rn, t) = \emptyset$ but the target of P_i is not a router (i.e. it is a PME) then no packet is causing NoC contention as contention may arise from the PME.

6.3.2 Defining rrc

As for Section 6.3.1, in this section, we use P_i to refer to the packet under analysis, P_j and P_k to other packets in the same router. Besides we use P_f and P_g to refer to other packets in a remote router.

As introduced before, rrc captures the contention a packet P_i suffers in router R_n due to the arbitration of another packet P_g in a router $R_m \neq R_n$. That is, those scenarios in which a packet is ready to be arbitrated to an output port but suffers contention delay without this being ascribable to any other packet traversing any output port of the same router (which would fall into the lrc category instead). With this rrc definition, we address the same corresponding scenarios already explained in lrc but with the relevant difference that contention actually arises in a remote router.

(S3) At a given time t in router $R_n \in \widehat{H_i}$, a packet P_i is queued and ready to be arbitrated at the *head* of an input port, i.e. $P_i = PH(P_i, R_n, t)$, and no packet is granted to traverse the output port because of backpressure from the destination input port ($\emptyset = PSO(P_i, R_n, t)$). The packet causing such contention needs to be looked up with the following recursive procedure:

  (S3a) If the packet P_g at the head of the target input port in the following router R_m is granted access to traverse its output port, then P_g causes contention on P_i.

  (S3b) If the packet P_g at the head of the target input port in the following router R_m is stalled because another packet P_f in another input port in R_m is granted access to traverse its output port, then P_f causes contention on P_i.

  (S3c) If the packet P_g at the head of the target input port in the following router R_m is stalled and no packet in R_m is granted access to traverse P_g's output port, then the packet causing contention on P_i needs to be looked up recursively in router R_o repeating the process from (S3a), where R_o is the next router for P_g in $\widehat{H_g}$. If the next target of P_g is not a router (i.e. it is a PME), then no contention guiltiness is ascribable to the NoC. Hence, no NoC contention is suffered by P_i [3].

Note that, with this procedure, we find a packet P_g in the NoC causing rrc on P_i, if it exists. We define such packet as $P_g = SP(P_i, R_n, t)$, where $SP(P_i, R_n, t)$ performs the recursive search described in steps (S3a), (S3b) and (S3c). Overall, $rrc^{R_n}_{g \triangleright i}(t) = 1$.

(S4) The previous scenario can be extended to also cover the cases in which P_i is not ready to be arbitrated. First, we identify the packet at the head of P_i's input port $P_j = PH(P_i, R_n, t)$, as in (S2). Then, we find out the packet causing remote contention on P_j, namely $P_g = SP(P_j, R_n, t)$, which in turn causes contention on P_i. Therefore, $rrc^{R_n}_{g \triangleright i}(t) = 1$.

[3]Notice that even though P_i does not suffer contention ascribable to the NoC, it can be suffering contention in other shared resources as the PME

6.3.3 Multiple VC impact on lrc and rrc

Generally, VCs are implemented to minimize the contention caused by packets that are in the same router's input port but that go to different output ports. That allows, for instance, to avoid head of line (HoL) blocking effect when possible maximizing routers' packets ejection. To do so, an input queue per port is assigned to each VC. VCs can be dynamically or statically allocated. With static VC allocation, the idea is to isolate certain communication flows which improve time predictability. With dynamic VC allocation, HoL blocking is reduced and average performance is improved but time predictability is more difficult to achieve which usually leads to pessimistic WCET estimates. Multiple VC's implementations in routers exist depending on the domain where they are applied.

In the high-performance domain, input port arbitration and VC arbitration are done together or in multiple rounds so as to maximize input ports and VCs requests matching with the available output ports. However, in CRTES with different area and energy constraints, simpler solutions [140] are usually implemented favoring also the systems' time predictability. For instance, VCs are usually implemented in on-chip routers by performing two arbitration rounds. In the first round, the input port that is granted access to the output port is selected and, in the second one, the VC for the already selected input port is chosen[4].

The extension of PWC and GRV to include VCs with static or dynamic allocation assuming a hierarchical two-round arbitration can be done by slightly modifying the functions already defined in Table 6.2. Functions that initially refer to the head packet of the input port, now need to provide the head packet of the VC in the same input port that has the turn for granting the VC arbitration. More in detail:

- $PH(P_i, R_n, t)$: PH function that initially was returning the P_j packet head of P_i packet input port for R_n in time t, now returns the packet P_j that is the packet that has won or has the turn to win the VC arbitration inside packet's P_i input port.
- $PSO(P_i, R_n, t)$: PSO function as PH, needs to return the packet P_j that is the head packet that has won the VC arbitration inside packet's P_j input port, target the same output port of P_i and granted permission to move forward to the next router, if any.
- $HN(P_i, R_{n+1}, t)$: HN function also should return the packet P_j that is the head packet of the targeted input port and VC of P_i in the next router R_{n+1}.
- $SP(P_i, R_n, t)$: SP function definition, in contrast, keeps being correct as it recursively refers to other scenarios (S3a, S3b and S3c in Section 6.3.2) that are now compatible with the hierarchical static-VC allocation adoption.

PWC formulation and scenarios can also be extended to more complex VC router implementations by modifying functions according to the VC and input port arbitration criterion used in systems' routers (e.g. each function returns a set of packets in the head of each VC of an input port).

[4]Notice that in the lrc and rrc formulation presented, we only consider the first arbitration round mentioned.

6.4 Deriving GRV

The use of GRV for PWC can be leveraged in two different stages of the software development cycle. First, during the timing validation phase for each test performed the breakdown provided by GRV complements the raw execution time measurements to determine whether the contention tasks generate each other stay within the allocated budget. GRV helps to distill the root causes for those cases with a test failing, i.e. resulting in a timing violation. It also captures *hidden* contention effects that compensate each other in the test but can potentially arise during operation. And second, GRV can be exploited during operation in case of overruns so that appropriate safety measures are applied, based on the root cause of the timing violation.

While the definition of GRV remains unchanged, regardless of the application scenario, the same cannot be said about its implementation. During the validation phase, execution time information of the packets/flows is collected whilst tests are executed. This information is analyzed off-line, reporting back any contention-related issue. Instead, during the operation phase, the analysis shall be performed on-line, which is more challenging since information is distributed in the wmesh, and a computation node or hardware, would be required to derive GRV. In this work, we target timing validation and hence, we focus on the former use of GRV (off-line analysis).

6.4.1 Off-line Analysis

In this case, GRV works with an execution trace containing information about packet ingress and egress in each router. Each generated trace contains: packet time information including arrival and leaving time to/from a router in the 2Dmesh, its source packet (i.e. the flow to which it belongs), destination, identifier, the router identifier in which the packet is stored at the time of the recording. We generate traces with the gNoCsim simulator [126], as shown in Section 6.5. We added regular monitors capturing traffic information in the NoC similar to how they can be implemented in any COTS multi/many-core based system. This allows us to compare GRV against PWC to assess its accuracy.

The information in the traces is used to determine the (contending) packet delaying the progress of any other (stalled) packet in the NoC. To that end, GRV checks for every flow, i.e. core, the packets that enter and leave each router and compares their timing with the ideal case (i.e. the packet is never stalled inside the router). When a packet is stalled, GRV identifies the source of the stall and classifies the contention delay as caused by the contending packet/flow. GRV also records the router where the stalled packet is so that every single cycle of contention experienced by any packet can be tagged with the contending flow, the router where it is experienced, and whether it fits the lrc or rrc case.

The outermost level of the GRV implementation is shown in Algorithm 2. The algorithm operates on the set of routers in the mesh ($\mathcal{R}$) and accumulates the information on contention in specific data structures (*lrc* and *rrc*).

Algorithm 2 GRV offline implementation.

```
procedure COMPUTE_GRV( lrc, rrc)
    for each time $t$ in $Time$ do
        for each router $R_n$ in $\mathcal{R}$ do
            GRV($R_n$, $t$, $lrc$, $rrc$)
        end for
    end for
end procedure
```

Algorithm 3 GRV routine.

```
procedure GRV($R_n$, $t$, $lrc$, $rrc$)                    ▷ Call for Router $R_n$ at time $t$
    $P_{ready}(R_n, t) \leftarrow$ packets ready to be arbitrated in $R_n$ at time t
    for $p_i \in P_{ready}(R_n, t)$ do
        $< CONT, R_{guilty}, p_{guilty} > =$ GRVREC$(p_i, R_n, t)$
        if $CONT$ then
            if $R_{guilty} == R_n$ then
                $lrc[R_n][p_i.src][p_{guilty}.src] + +$
            else
                $rrc[R_n][p_i.src][p_{guilty}.src] + +$
            end if
        end if
    end for
end procedure
```

For every time instant t, which corresponds to cycles in a discrete approximation, the algorithm considers the packets' status at time t (e.g. packets position in the buffers, packets traversing routers,..) and calls *GRV routine* (see Algorithm 3) on all routers in $\mathcal{R}$ to populate the *lrc* and *rrc* data structures with contention information.

GRV routine builds on a recursive approach to compute lrc and rrc calling *GRVrec*. As a base step, the algorithm iterates over all packets ready to be arbitrated and queued in all R_n router input ports (they were not granted access to their output port): the recursive step *GRVrec* is invoked on those packets to determine the source of contention, if any. *GRVrec* eventually returns the guilty packet p_{guilty} causing contention to the packet under analysis (p_i) and the router R_{guilty} where p_{guilty} packet has been found as the actual source of contention. Note that $CONT$ is a boolean indicating whether any packet effectively causes NoC contention on p_i.

- If R_{guilty} corresponds to R_n, then p_{guilty} causes the contention in the same router where p suffers the contention (R_n). Hence, contention is classified as local router contention (lrc).
- Otherwise, if R_{guilty} identifies a different router than R_n, the contention suffered by p must be classified as remote router contention (rrc).

In both cases (lrc and rrc) contention is accounted in the router where p_i suffers the contention (R_n). Per-flow information on what flow causes contention and what flow suffers it is directly obtained from the tasks owners of p and p_{guilty}.

Algorithm 4 GRV Recursive function.

```
function GRVREC($p$, $R_n$, $t$)                    ▷ Call for packet $p$ in Router $R_n$ at time $t$
    $p_h \leftarrow PH(p, R_n, t)$
    $p_g \leftarrow PSO(p_h, R_n, t)$
    if $p_g \neq \emptyset$ then                     ▷ $p_g$ traverses the output port
        return $< TRUE, R_n, p_g >$
    else
        if $target(p_h)$ is a router then
            $p_{next} \leftarrow HN(p_h, R_{n+1}, t)$
            return GRVREC($p_{next}, R_{n+1}, t$)
        else
            return $< FALSE, R_n, p_h >$             ▷ Target is a PME
        end if
    end if
end function
```

Function *GRVrec* (see Algorithm 4) implements the core search for the p_{guilty} in router R_{guilty} that is causing contention to p at time t. The function implementation exhaustively captures the lrc and rrc scenarios detailed in sections 6.3.1 and 6.3.2 respectively.

6.4.1.1 *lrc*

As a first step, $GRVrec$ retrieves the head packet at the input port targeted by p (i.e. the packet passed to $GRVrec$ as input), which could be p itself. In line 3, the algorithm gets the packet granted access to the output port targeted by the input port head p_h. Packet p_g (line 4), if it exists, is the one that causes contention to the others. In the very first invocation to $GRVrec$ from GRV, p_g in line 4 is in the same router R_n as p (i.e. p_i in GRV) and hence, generates lrc on p. By definition, if $p_g = p_h$, then p_h would be granted access to the output port, and hence, would experience no contention but cause contention on p. A special scenario is where $p = p_h = p_g$: in that case p experiences no contention at all.

The lrc scenarios described in Section 6.3.2 are exhaustively modeled by Algorithm 4 as follows:

- (S1): if $PH(p, R_n, t) = p$ and $p_g \neq p$, the packet under analysis is stalled in the head of the input port suffering lrc contention because of another packet, from another input port in R_n, being arbitrated in the same output port.
- (S2): if $PH(p, R_n, t) \neq p$ and $p_g \neq p$, the packet under analysis is stalled suffering contention because the packet at the head of the same input port is currently being arbitrated, or the latter is itself stalled by another packet, from another input port in R_n, being arbitrated in its output port.

Conversely, if p_g is empty in the first call (line 6), but the target of p is not a router (line 10), then no other packet in the NoC is blocking p. Hence, no NoC contention needs to be accounted for.

6.4.1.2 *rrc*

If p_g is empty in the first call (line 6), then the packet causing contention is in a different router and we fall into the rrc scenario. Hence, we obtain p_{next}, which corresponds to the head packet in the input port of p's next router (R_{n+1}). This is shown with function $HN(p_h, R_{n+1}, t)$ in line 8. Then, we trigger the recursive call to $GRVrec$ over p_{next} (line 9) to find the blocking packet in R_{n+1} or beyond. The recursion ends as soon as the PSO returns a non-null value or the next target of the packet under analysis is a PME. The rrc scenarios described in Section 6.3.2 are exhaustively modeled by Algorithm 4 as follows:

- (S3a-c): if in the first iteration $PH(p, R_n, t) = p$ but $PSO(p_h, R_n, t) = \emptyset$ then the packet under analysis is suffering contention because of a packet in another router. The packet causing contention could be either: p_{next} that is the head of the target input port for p in the following router R_{n+1} (S3a), or the packet blocking p_{next}, which can be either the packet being granted access to p_{next} output port (S3b) or a packet in another router down the chain of routers (S3c).
- (S4): if $PH(p, R_n, t) \neq p$ and $PSO(p_h, R_n, t) = \emptyset$, then we fit exactly in the same scenarios as in the previous point, with the only difference that p is not suffering contention directly but through the packet p_h at the head of p's input port.

Note that in a deadlock-free NoC, we will eventually find a packet making progress or a PME. So, the algorithm eventually finds the blocking packet causing NoC contention for p_i in GRV.

6.4.2 GRV for different wmesh setups and NoCs

Besides the wmesh setup we have used in this work, several other setups can be adopted. In this section, we cover the most relevant ones along with how PWC-GRV covers them.

Several wmesh features can cause predictability (budgeting) problems, including non-predictable arbitration policies, virtual channel allocation, and maximum packet length, which depending on whether they are allocated dynamically or statically can result in huge contention bounds [64]. However, this does not have any effect on the functioning of our PWC-GRV contention measuring approach. Hence, hard-to-predict features only affect the number of cycles accounted as PWC among each pair of tasks. In fact, even in a NoC setup in which starvation can occur our PWC-GRV would work and help to identify this issue.

Our PWC-GRV proposal targets measuring contention for timing validation and optimization in wNoCs systems using deterministic routing algorithms, as these are the preferred policies to allow the WCD estimation CRTES [141]. We do not target the applicability of PWC-GRV in systems using non-deterministic routing algorithms as adaptive or dynamic routing. These kinds of non-deterministic routings, even though they increase NoC performance, bring unpredictability to the NoC and hamper the time V&V required for CRTES, as NoC WCD can become too pessimistic to be useful. Hence, the presented PWC-GRV does not directly support adaptive or non-deterministic routings which are out of the scope of this work.

Regarding other NoCs, PWC-GRV can also be applied to other types of distributed NoCs and network configurations such as the ones using virtual cut-through switching [142]. Indeed, we present PWC reference in a way that eases the adaptation of our proposal to other NoC topologies that for instance have routers with more or fewer ports, routing setups, and multiple or single destination flows.

6.5 Results

We evaluate GRV on 2DMesh wNoCs of different dimensions, ranging from 3x3 to 6x6, hence including the dimensions of COTS manycores (e.g. 4x4 in the case of the Kalray MPPA 256 [7]) and beyond. We use gNoCsim [126] cycle-accurate NoC simulator that injects both synthetic and real traffic in the NoC. We model an XY-routing mesh network with 5 bidirectional input/output ports (X+, X-, Y+, Y- and PME) of 10 flits capacity (i.e. we use 2 buffers per router port: one used as input and the other as an output, each of them with a capacity of 10 flits). Routers implement round-robin arbitration, XY-routing, and wormhole switching. Flit traversal latency in the no-contention scenario is 1 cycle to traverse the router and 1 cycle to traverse the link between routers. Cores are connected to each router and send requests to different memory modules attached to boundary routers, which serve one request per cycle.

(a) Synthetic traffic: we use gNoCSim as a standalone simulator that injects self-generate synthetic traffic in the NoC. We inject packets in the PME input ports that we have named as synthetic cores C_x with a given IR, in some cases limiting the number of in-flight requests to mimic the impact of contention experienced by a task τ_x executing in C_x.

(b) Hybrid traffic: we use SoCLib [126] SoC simulator, which we integrate with gNoCsim so that the latter works in slave mode. In this experimental setup, SoCLib simulates real code being executed in an NGMP Sparc-based core [50], whose memory petitions traverse the NoC (implemented with gNoCsim) to reach memory. We model a TILED MC with each tile comprising L1 cache memories and a core that communicates with the rest of tiles and memory using a NoC router. Processor cores implement an in-order pipeline with 32KB 4-way 16B/line IL1 and DL1 caches, where DL1 is write-through, in line with NGMP multicore for the space domain. The manycore architecture also includes a unified memory, so that each core targets a shared memory. For the sake of controllability to assess high-contention scenarios, in this setup, we have some of the cores running real benchmarks and hence, injecting the corresponding petitions in the NoC, whereas the remaining cores inject synthetic traffic generated by gNoCsim according to given specifications (e.g. sustained write traffic to memory).

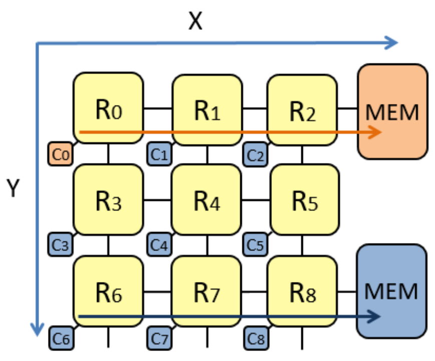
In this section, we mainly focus on packets of 1 flit size (i.e. all flits are header flits and switching and arbitration can take place every cycle), in line with recommendations in [51] to minimize maximum contention. We discuss packets with multiple sizes specifically in Section 6.5.2.

From gNoCsim, either standalone or integrated with SoCLib, we generate an execution (timing) trace with the information presented in Section 6.4.1. In order to compute the GRV of the experiments, we have implemented a C++ trace parser based on the pseudocode described in Section 6.4. The results reported in this section are obtained by applying GRV, as formalized in Section 6.3, on simulator traces to obtain contention breakdowns. It is worth noting that the same methodology can also be applied to PMCs or event traces collected from real NoC-based systems operation.

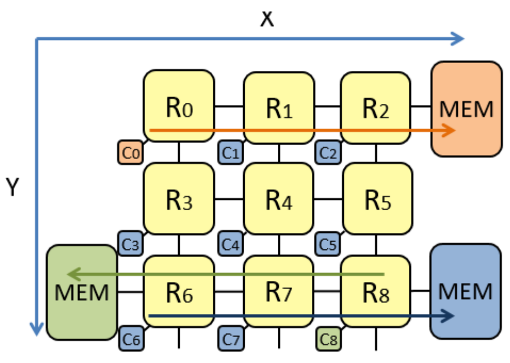
6.5.1 Synthetic Traffic

We have performed several experiments with different mesh architectural setups and different IRs intended to create high contention in meshes using synthetic traffic. The particular evaluation choices taken allow for determining a priori where contention should occur and what core causes it, thus allowing to validate GRV. The setups chosen intend to be representative of different traffic patterns with varying sources of contention, thus challenging GRV capabilities.

For the synthetic traffic and for the sake of simplicity, the task under analysis, TuA or (τ_0), is placed in C_0 and has exactly one packet in-flight. Hence, whenever the packet reaches memory, a new packet is inserted. The remaining cores, instead, inject packets at a high injection rate (IR = 1) with no packets in-flight restriction.



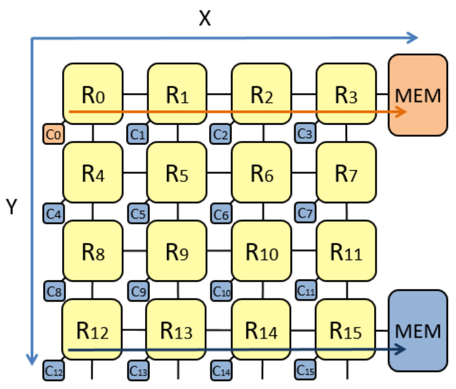
(a) Setup 1.



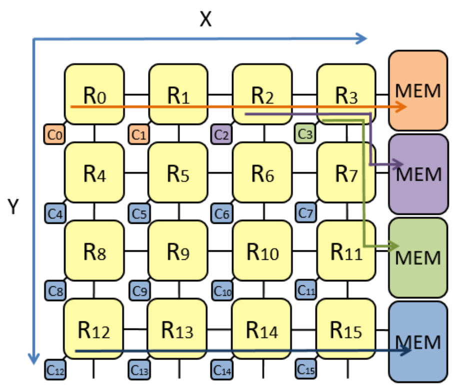
(b) Setup 2.

Figure 6.5: Illustrative 3x3 mesh setups evaluated.

3x3 wmesh: the first setup (see Setup 1, Figure 6.5a), has 2 memory modules, one attached to R_2 targeted by packets from C_0, and the other to R_8 targeted by packets from cores C_1 to C_8. Setup 2 (see Figure 6.5b) is like Setup 1 but with C_8 targeting the 3rd memory module attached in R_6.



(a) Setup 3.



(b) Setup 4.

Figure 6.6: Illustrative 4x4 mesh setups evaluated.

4x4 wmesh. In Figure 6.6a (Setup 3) we define a 4x4 2DMesh with two memory controllers, analogous to Setup 1. Setup 4 (see Figure 6.6b) corresponds to a more complex scenario where the NoC has 4 memory modules. The first memory module is attached to R_3, and is targeted by packets from cores C_0 and C_1. The second and third memory modules, attached to R_7 and R_{11}, are targeted respectively by packets from C_2 and C_3. Finally, the fourth memory module is attached to R_{15} and is targeted by packets from the rest of the cores in the mesh (C_4 to C_{15}).

5x5 mesh and **6x6 mesh**. In order to analyze the scalability of our approach, we have defined setups for bigger meshes analogous to Setups 3 and 4 for 4x4, i.e. with two memory modules (one for C_0 and one for the rest of cores), and with one memory module per row where cores C_0 and C_1 target the memory module in the first row, and each other core in the first row one memory module in another row. In this latter setup, cores not in the first row target the memory module in the last row.

For each experiment we analyze the contention the TuA suffers ($cont_{\tau_0}$) due to the other tasks:

(a) We can break down $cont_{\tau_0}$ per each router where the contention takes place. This information can be obtained with both, the baseline contention breakdown metric and GRV.

(b) Baseline contention breakdown metric: where the owner of the last packet granted access to the target output port in a router is regarded as the one causing contention, thus strictly at local router level.

(c) $cont_{\tau_0}$ broken down per contention type (lrc and rrc).

(d) The contention τ_0 suffers from all the other co-running tasks ($cont_{\tau_j \triangleright \tau_0}$) following our PWC definition.

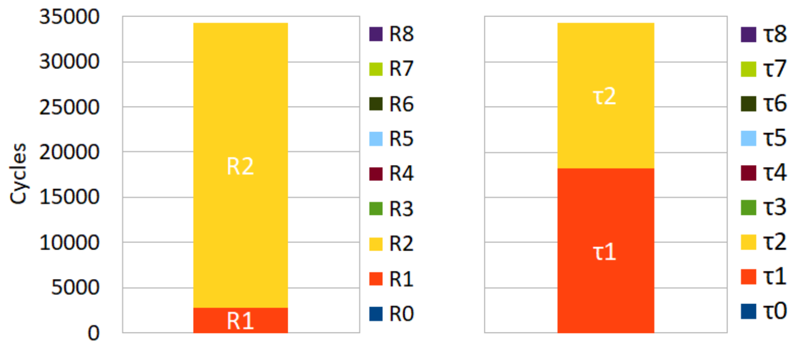
(e) A simultaneous break down ($cont_{\tau_j \triangleright \tau_0}$) per contender, showing lrc and rrc cycles.

Note that the baseline contention breakdown metric can only provide the first two breakdowns, and only GRV can provide the last three.

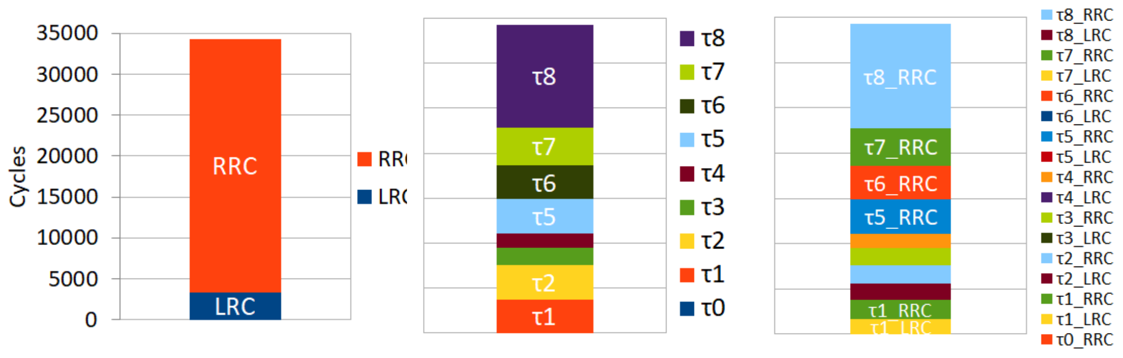
6.5.1.1 *Setup 1 (3x3 Mesh) Result Analysis*

Packets from τ_0 (TuA) in C_0, traverse routers R_0, R_1, R_2 to reach memory, as shown in Figure 6.5a. The high contention experienced by the other cores to reach the memory module at R_8 is expected to translate into high contention in the TuA due to backpressure. This is so since, despite C_0 targets memory module in R_2 and traffic from C_1 and C_2 target memory module in R_8, C_0 and C_1 share the $X-$ input port in R_2 with C_0 targeting the $X+$ output port and C_1 the $Y+$ output port. Hence, if cores C_2 to C_8 experience high contention in their path to R_8, this contention will be back-propagated to C_1 packets by R_2 $Y-$ port and at the same time will end affecting C_0 packets.

Consistent with that analysis, Figure 6.7a shows that packets from τ_0 mostly suffer contention in R_2 (even if they also traverse R_0 and R_1). That happens, as we have already explained, because R_2 is the router, from the C_0 path, which aggregates more traffic. R_2 receives traffic from C_0, C_1 and C_2 but also backpressure from packets from C_3 to C_8 in R_3 and R_8. Hence C_0's packets are only stalled at the router that aggregates more traffic (R_2).



(a) Contention per router. **(b)** Baseline contention per task.



(c) GRV. PWC per contention type. **(d)** GRV. PWC per task. **(e)** GRV. PWC per task and cont. type.

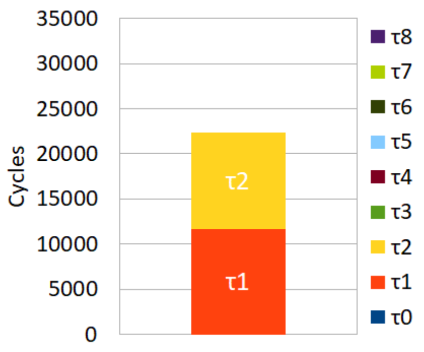
Figure 6.7: 3x3 2DMesh $cont_{\tau_0}$ analysis (Setup1).

The baseline contention breakdown, see Figure 6.7b (y-axis shared with the other figures), ascribes contention to packets from C_1 and C_2 for stalling packets from C_0 as these are the only two cores that physically share links with C_0 path. However, an analysis of the PWC shows that these cores are mostly experiencing backpressure from other cores attempting to reach R_8 memory module, as shown in Figure 6.7c. We see that most of GRV, PWC is rrc, so C_1 and C_2 are not the real source of contention. In fact, if we decrease IR down to 0.1 for C_1 and C_2, contention remains roughly unchanged since C_0 packets have C_1 and C_2 packets in front all the time, but the latter are stalled due to backpressure from the other cores.

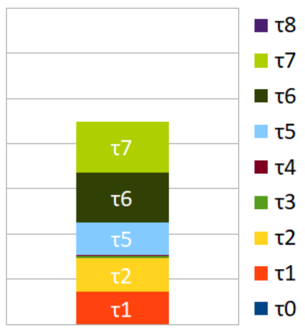
Cores close to the target (e.g. C_8) are expected to produce a larger fraction of the BW in those routers, causing more backpressure on other packets from C_1 and C_2, which propagate backpressure to the packets of the TuA[5]. GRV (Figure 6.7d) shows exactly that this is the case. Contention contribution is mostly dominated by cores in the path of C_1 and C_2 to R_8 memory module (C_1, C_2, C_5 and C_8) and those cores with higher BW to such memory module due to the locally-fair globally-unfair round-robin arbitration (C_5, C_6, C_7 and C_8). Since C_8 is dominant in both causes of contention, it is naturally the core causing the largest fraction of contention on the TuA. Notably, GRV accurately reflects those effects. For completeness, we also show Figure 6.7e, where we see that GRV provides information broken down per contender and contention type (lrc/rrc), being such contention only rrc for cores C_3 to C_8.

6.5.1.2 *Setup 2 (3x3 Mesh) Result Analysis*

Under this setup, C_8 sends packets to a different memory module (the one attached to $R6$). As a result, it cannot cause any contention on C_1 and C_2 (and hence the TuA) due to backpressure. In Figures 6.8a and 6.8b, we observe how τ_8 contention on τ_0 disappears, which matches with the rrc contention (backpressure) τ_8 was creating in setup 1 (see Figure 6.7d purple color). Contention caused by the other cores on the TuA remains roughly the same except for τ_3 and τ_4 contention contribution reduction to the TuA caused by the alignment variation between packets coming from these two tasks w.r.t τ_0's packets. That is confirmed by comparing Figure 6.7 and Figure 6.8.



(a) Baseline contention per task.



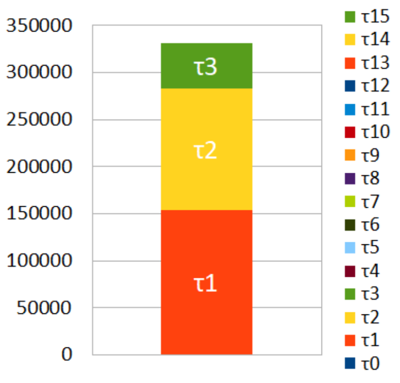
(b) GRV. PWC per task.

Figure 6.8: 3x3 2DMesh $cont_{\tau_0}$ analysis (Setup2).

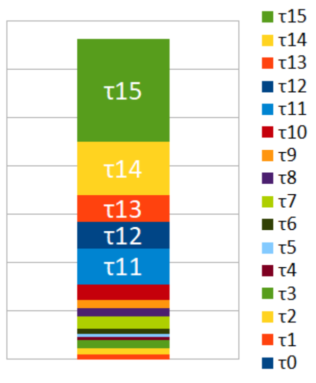
[5]Notice that in all the synthetic traffic scenarios analyzed in this section, CTs contribution to the TuA contention matches the expected BW distribution given by the XY routing and round-robin arbitration used. This is so because NoCs work in a saturation state (e.g. Injection Rate ¿ Ejection Rate) and CTs have a uniform homogeneous synthetic IR=1 using their assigned BW and potentially the remaining BW unused from the TuA.

6.5.1.3 *Setup 3 (4x4 Mesh) Result Analysis*

The peculiarity we observed in this experiment w.r.t the one shown in Setup 1 (3x3) is that the contention that τ_0 suffers because of its co-runners is around 9 times bigger since it has to traverse an additional router, thus with much-decreased BW to reach R_3, and the number of cores creating backpressure is also much higher. This can be observed, for instance, in Figure 6.9a for the baseline contention assignment. As before, with the baseline technique contention is only ascribed to cores sharing routers with the TuA, namely C_1, C_2 and C_3 (see Figure 6.9a). Instead, GRV properly captures the fact that backpressure from other cores is, instead, the one causing contention in the TuA, as shown in Figure 6.9b. Since no further insights are obtained from this setup, we do not deepen on its analysis.



(a) Baseline contention per task.



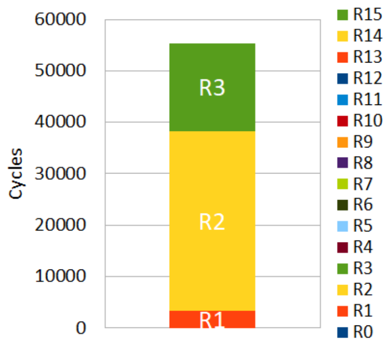
(b) GRV. PWC per task.

Figure 6.9: 4x4 2DMesh $cont_{\tau_0}$ analysis (Setup3).

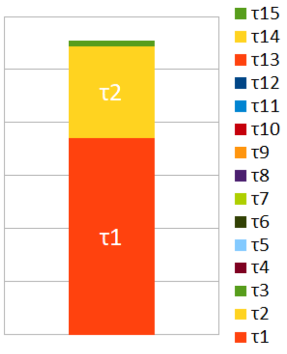
6.5.1.4 *Setup 4 (4x4 Mesh) Result Analysis*

Figure 6.10a shows that packets from the TuA suffer contention mostly in R_2 and R_3. In comparison to Setup 1, R_3 stalls decrease noticeably in favor of R_2 stalls since now C_0, C_1 and C_2 share the $X+$ input port in R_3 whereas in Setup 1 R_2 input port was only shared among C_0 and C_1. That means that when backpreassure is suffered by R_3 output port $Y-$, as before packets from C_0 have higher chances to be stalled in R_2 than before because they are sharing the $X+$ input port of R_3 with 1 more flow than in Setup 1.

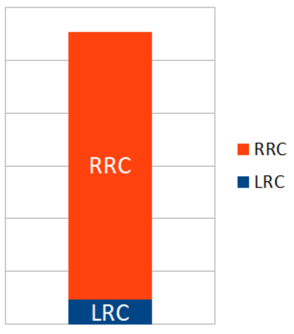
In Figure 6.10b, the baseline solution ascribes contention to tasks τ_1, τ_2 and τ_3 directly sharing links with task τ_0 path, omitting once again that most of the contention that τ_0 incurs is rrc, as captured by GRV in Figure 6.10c. In that case, most of the collisions that packets from τ_0 suffer are due to packets coming from τ_1 (τ_1 predominance in Figure 6.10b). Note also that backpressure makes, again, cores with higher BW in R_{15} memory module cause higher backpressure (rrc) on the TuA (τ_0), with trends similar to those of Setups 1 and 3 (Figure 6.10d). Also, the contention caused by τ_4, τ_5 and τ_6 is negligible due to the fact that their packets do not compete with τ_2 ones, and only do it with τ_3 ones in a router (R_7) still distant from R_{15}, thus with little BW. Overall, packets in cores out of the path to the memory modules produce largely decreasing contention on the TuA as we move from bottom to top in the mesh. Similarly, Figure 6.10e shows at fine grain that only tasks τ_1, τ_2 and τ_3 cause lrc and rrc contention as they directly share links with τ_0, whereas tasks τ_4 to τ_{15} only contribute with rrc.



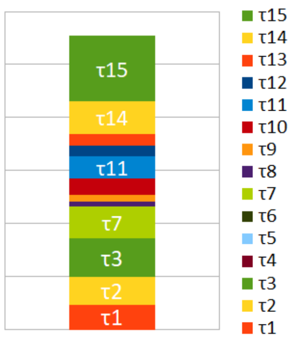
(a) Contention per router.



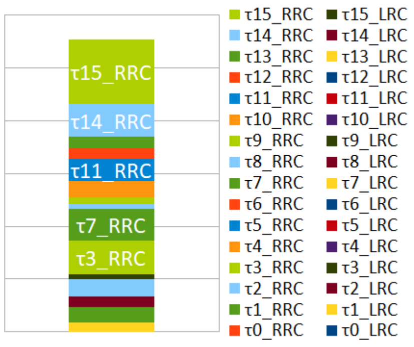
(b) Baseline contention per task.



(c) GRV. PWC per contention type.

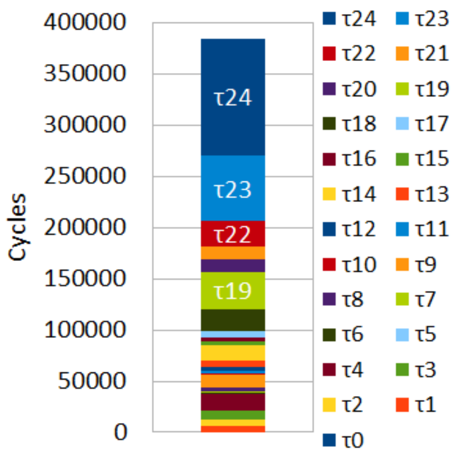


(d) GRV. PWC per task.



(e) GRV. PWC per task and cont. type.

Figure 6.10: 4x4 2DMesh $cont_{\tau_0}$ analysis (Setup4).



(a) PWC metric per task (5x5).

Figure 6.11: 5x5 2DMesh τ_0 contention analysis.

6.5.1.5 *Larger wmesh (Setup 5 and Setup 6)*

These larger scenarios, see Figure 6.11a (5x5), show, as in the previous experiments, that the contention τ_0 incurs depends strongly on the BW assignment each task in the wmesh has. When the TuA experiences remote contention, this remote contention matches the BW distribution the co-runner tasks that generate this remote contention have). More in detail, we observe that tasks running in cores closer to their targeted memory module (R_{24}) namely C_{24}, C_{23}, C_{19}, C_{14} are the ones that generate remote contention to τ_0 because of their bigger BW assignation. Still, GRV with PWC, independently on the mesh size, keeps being able to detect and correctly capture contention a TuA incurs because of other co-runner tasks in the system even if these other co-running tasks do not share any physical link with the TuA (remote contention). Note that results for a 6x6 setup are omitted since they do not provide any further insight.

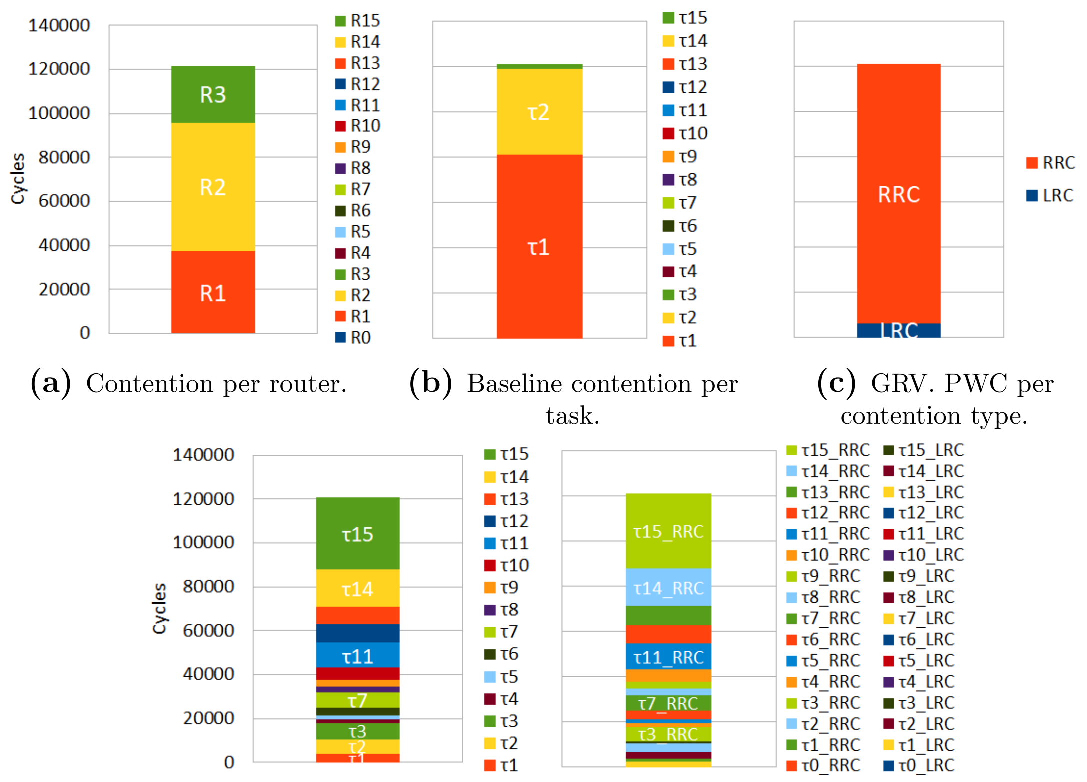
6.5.2 Synthetic Traffic with multiple size packets

PWC and GRV also support analyzing and providing contention breakdowns in NoC setups where packets have variable sizes (e.g. packet size bigger than 1 flit). As shown in the previous sections, PWC is defined at packet level so that the contention suffered and caused by packets is ascribed between packets regardless of their size. Similarly, GRV contention classification criteria are defined at packet level or higher level (e.g. router, flow,...) making packet size orthogonal to GRV. We have analyzed contention setups and scenarios already shown in the previous section using multiple packet sizes. Results obtained do not change substantially. In this section, and for the sake of reducing repetitiveness, we provide results for Setup 4 only. Figure 6.12 shows the contention analysis for Setup 4 (see Figure 6.6b) when synthetic cores send packets with $packetsize = 2$ and $packetsize = 6$ flits (50% of the times each size). Results with bigger packet size than 1 flit in Figures 6.12 show a relevant contention increase for the same number of packets (around 2.4 times) with respect to contention observed in contention analysis ($packetsize = 1$ flit). The increase is explained by the fact that now packets are longer, more flits need to be injected than in the $packetsize = 1$ scenarios, which increases contention, ultimately enlarging arbitration turns. More in detail, Figure 6.12a shows that τ_0 contention still mainly takes place in $R2$ despite now part of the contention suffered by the TuA occurs in $R1$ instead of $R2$. Packets need to wait more time for gaining the arbitration due to other long packets. That favors packets to easily spread into different routers, thus possibly creating and suffering contention in many of them at the same time.

Figure 6.12b does not show any relevant change compared to the $packetsize = 1$ flit analysis while Figure 6.12c shows that rrc portion type is even bigger than before, because of the same effect described in Figure 6.12a.

Figure 6.12d shows that contention suffered by τ_0 still mainly comes from τ_{15}, τ_{14}, τ_7 and τ_3 even though now τ_4 to τ_6 also contribute (i.e. packets size change arbitration alignments and tasks that in some scenarios do not collide with the TuA in others they do). Figure 6.12e confirms the merged effect of increasing rrc contention and each task contention contribution, as already shown in Figures 6.12c and 6.12d, respectively.

Although the size of the packets is transparent for PWC and GRV when analyzing and classifying contention, scenarios with multiple packet sizes are the ones where PWC and GRV can be more useful as the gap between the potential WCD that packets can suffer and the real contention that packets end suffering in operation is bigger. This is so because to compute WCD, the worst-case contention case arises when the packet under analysis always collides with the longest possible packet from other tasks in the NoC (e.g. $packetsize = 6$ flit in this analyzed scenario).



(a) Contention per router. **(b)** Baseline contention per task. **(c)** GRV. PWC per contention type.

(d) GRV. PWC per task. **(e)** GRV. PWC per task and cont. type.

Figure 6.12: 4x4 2DMesh $cont_{\tau_0}$ analysis (Setup4) with multiple packet sizes.

Figure 6.13 shows, from left to right, the WCD τ_0 can suffer when running in Setup 4 with $packetsize = 2$ and $packetsize = 6$, the WCD τ_0 can suffer when running in Setup 4 with $packetsize = 1$ and the GRV. The PWC per task, instead, is already shown in Figure 6.12d. In Figure 6.13 we observe that the WCD global bound with $packetsize = 2$ and $packetsize = 6$ flits (WCD_PS2&6) is 6 times bigger than the WCD global bound with $packetsize = 1$ (WCD_PS1). That matches with the fact that WCD_PS6 always considers τ_0 contention caused by other tasks' packets with long size ($packetsize = 6$). Moreover, contention observed in Setup 4 with packet sizes 2 and 6 (right bar in Figure 6.13), is far from the maximum contention that τ_0 can suffer shown in WCD_PS2&6). It is worth mentioning that WCD computation

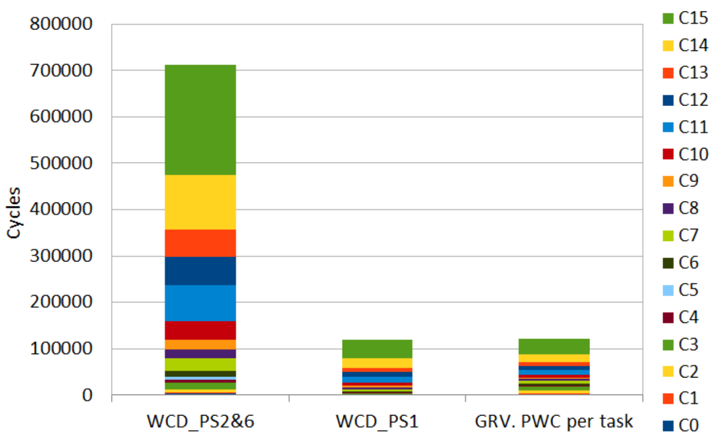


Figure 6.13: Average analysis time per packet in experiments.

techniques aim at bounding worst-case global contention that a task (e.g. τ_0) can suffer because of other tasks. However, they are not meant to compute bounds to the individual maximum contention contribution each task can create on a specific task. This is because WCD is derived under a specific BW distribution (e.g. round-robin arbitration) and, based on that, we can also derive tasks contribution to that WCD. Nevertheless, during tasks' execution, BW distribution can vary (e.g. a task generates fewer petitions not using their assigned BW so that other tasks use their remaining BW) and hence, so will vary tasks' contribution to a specific task without exceeding the global WCD computed.

6.5.3 Hybrid traffic

We have performed several experiments with hybrid traffic based on one mesh architectural setup in order to show how GRV works when NoC has high, medium and low contention using real and synthetic traffic.

As in the previous section, in all cases the TuA (τ_0), which in this case runs a $MatMul$ benchmark, is placed in C_0, but it does not have any packet in-flight restriction. The remaining cores, instead, inject packets at a high, medium and low injection rate (IR = 1, 0.14, 0.11 respectively) also with no packets in-flight restriction. The latter two IRs have been carefully chosen to lead to saturation and no-saturation scenarios respectively as discussed next.

3x3 wmesh: When analyzing real traffic behavior, we define a simpler setup than the ones already shown (see Setup 5, Figure 6.14) that has 1 memory module attached to R_8 targeted by packets from all cores (i.e C_0 to C_8). The aim of this setup and experiments is to show how GRV works when the NoC is working under maximum contention (IR = 1 which makes total contender IR be $IR_{cont} = 8$) due to the IR of the cores from C_1 to C_8, medium saturation (IR = 0.14, $IR_{cont} = 1.12$) and no saturation scenario (IR = 0.11, $IR_{cont} = 0.88$).

Note that, when the total injection rate of the NoC IR_{total} ($IR_{total} = IR_{cont} + IR_{\tau_0}$) is greater than the NoC maximum ejection rate (1 $packet/cycle$), packets saturate the NoC and accumulate in the routers' buffers causing high contention. Otherwise, if the

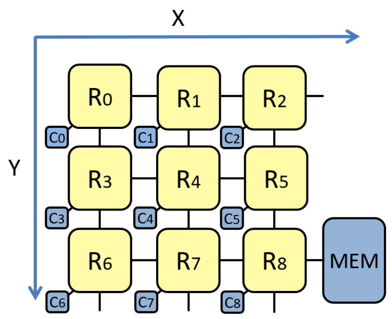
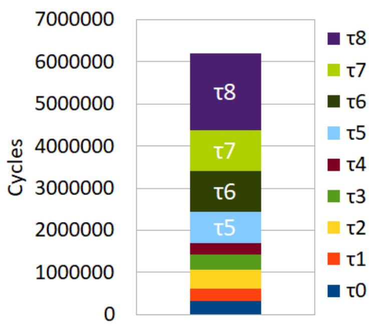


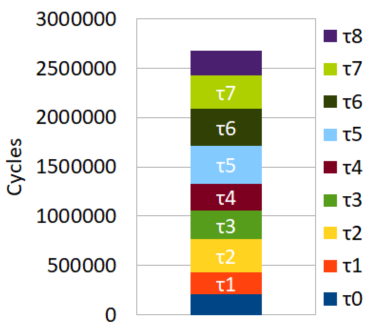
Figure 6.14: Illustrative 3x3 mesh setup evaluated (Setup5).

total injection rate is smaller than the NoC ejection rate, packets do not accumulate and cause lower contention than in the previous case.

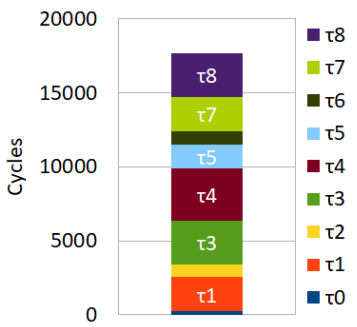
In terms of the results, for setup 5 we show $cont_{\tau_0}$ due to the other co-runner tasks when varying their $IR = 1$, 0.14 and 0.11 respectively in Figures 6.15a-6.15c. As expected, $cont_{\tau_0}$ experienced by τ_0 decreases ad IR for the CTs decreases. In particular, contention is around 6,200,000 cycles for $IR = 1$; 2,700,000 cycles for $IR = 0.14$; and 18,000 cycles only for $IR = 0.11$.



(a) MatMul vs contend IR = 1.



(b) MatMul vs contend IR = 0.14.

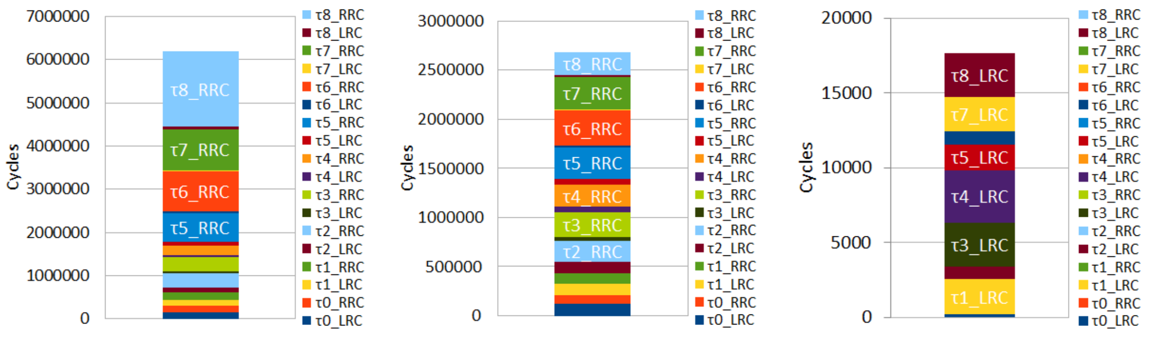


(c) MatMul vs contend IR = 0.11.

Figure 6.15: 3x3 2DMesh PWC $cont_{\tau_0}$ per task analysis (Setup5).

Figure 6.15a for $IR = 1$ shows that, since all tasks target the same memory controller, tasks that are closer to memory, and consequently have more BW according to the round-robin arbitration (τ_8, τ_7, τ_6, τ_5), cause more contention to τ_0. Indeed, tasks contribution perfectly matches round-robin BW distribution to tasks along the NoC.

However, when co-runner tasks IR decreases to $IR = 0.14$ (see Figure 6.15b), tasks contention distribution tends to equalize. That is explained because with lower IR (still sufficient to cause saturation), packets from the routers with the highest BW (e.g. above 0.14) are generated at a lower rate than they are granted in the arbiters. Hence, all routers in general, but those with higher BW in particular, generate lower interference, and since saturation needs contribution from more routers, those receive higher BW. For instance, tasks in cores C_3 to C_8 cause a cumulative IR of 0.84. Therefore, cores C_0 to C_2 have a cumulative BW of at least 0.16 sustainedly. At a lower scale, the very same effect is captured again in Figure 6.15c, where the contention is very low. In that case, TuA packets practically traverse the NoC without contending with packets from other tasks. Hence, contention relates more to whether injected packets collide with others unluckily rather than to saturation. GRV also captures this effect in the lrc and rrc contention types classification (see Figures 6.16a-6.16c). When CTs have high IR (see Figure 6.16a), most of the contention is rrc and is created by the tasks that have more BW in the NoC. However, when the contenders' IR is low (see Figure 6.16c), rrc contention is residual and the low contention that takes place in the NoC is lrc.



(a) MatMull vs contend IR = 1. (b) MatMul vs contend IR = 0.14. (c) MatMul vs contend IR = 0.11.

Figure 6.16: 3x3 2DMesh lrc and rrc PWC $cont_{\tau_0}$ per task analysis (Setup5).

Another relevant result of this setup is that the injection rate of the TuA is not limited by construction. Hence, a packet of the TuA may be produced when older TuA packets are still traversing the NoC. As a consequence, especially for high IR scenarios, one packet of the TuA is more likely to generate backpressure on other packets from the TuA. This is reflected in Figures 6.15a-6.15c, where we see that τ_0 (the TuA) causes contention on τ_0 itself.

6.5.4 Off-line algorithm analysis

We have implemented the off-line GRV computing algorithm described in Section 6.4, where the trace is processed sequentially. The execution time required by the algorithm depends mainly on: (1) the number of packets analyzed with the algorithm, and (2) the size of the mesh where these packets are analyzed. In Figure 6.17 we show the *execution_time/packet* (μseconds/packet) of the experiments done in 3x3, 4x4, 5x5 and 6x6 meshes. Results have been obtained in a laptop with an Intel i7-8650U processor with 16GB of DRAM.

Those per-packet execution times led to 2 seconds to process ≈35,000 packets for Setup 1, and to up to 1 hour and 6 minutes to process more than 14 million packets for 6x6 setups, in all cases requiring less than 200MB of main memory. Moreover, scenarios with a packet size bigger than one flit can be treated as a particular case of increasing the number of flits or packets (in the case of $packetsize = 1$ flit). Note that, for instance, the NoC of the SiPearl Rhea processor from the EPI [29] implements a 6x6 mesh NoC, so setups considered in this chapter are in line with those NoC sizes.

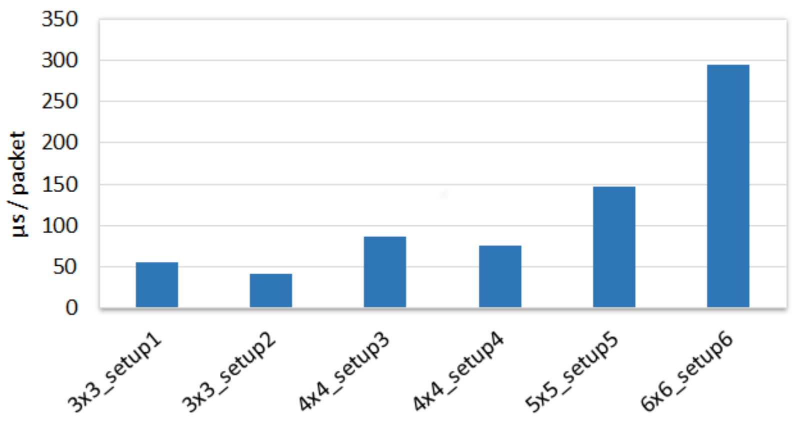


Figure 6.17: Average analysis time per packet in experiments.

6.5.5 Assessment of the GRV properties

As shown in Section 6.3, a reliable GRV must adhere to several properties which we review in light of the results obtained. First, GRV successfully classifies every single cycle of contention attributing it to an appropriate source of such contention, so no contention cycle remains unclassified or is classified twice. And second, unlike the baseline metric, which fails to properly attribute contention to tasks, GRV successfully determines the cores causing such high contention whether it occurs in a given specific router. Hence, GRV allows carrying precise validation and optimization information for tasks running on a mesh-based manycore. Last but not least, our results show the scalability of our approach to derive GRV by considering 3x3 up to 6x6 meshes.

6.6 Conclusions

In this chapter, we define PWC for wmesh, a golden metric that allows ascribing actual shares of the contention a given task suffers from the other co-runner tasks in the wmesh at packet-level. We analyze the challenges of measuring and classifying PWC at packet-level in wmeshes, where the contention is split across the mesh routers and contending flows. We also discuss how this information needs to be combined to be useful for validation & verification purposes. We present GRV, a criterion to fairly break down the contention suffered by a task among its co-runner tasks. GRV ascribes contention cycles to the actual contending packet causing it in the local or remote nodes. Overall, GRV can provide valuable information for performance validation, debugging, and optimization by revealing accurately how contention arises in wmeshes.

Chapter 7

Software Timing Diagnosis on NoC-based Manycores

7.1 Introduction

The use of NoC-based manycores in CRTES demands appropriate support for timing Verification and V&V to provide evidence of compliance against safety guidelines. In this line, while some solutions have been devised for bus-based small multicores [76], they do not match the characteristics of NoCs, whose decentralized arbitration nature and distributed traffic with multiple interdependent contention locations (e.g. routers and buffers) challenges their timing V&V.

This chapter describes a new approach for timing V&V of NoC-based multicores for CRTES. In particular, we show that appropriate statistics and trace-based information, obtained with very limited hardware support, can be exploited to quantify the contention each task produces on each other task in a NoC-based multicore. Such evidence has a twofold use: (1) in the scope of offline analysis, it allows performing timing V&V prior to deployment, as needed for certification purposes; (2) for online analysis, the same information, if collected periodically during execution, allows for detecting potential deadline violations before they actually occur, so that corrective actions can be taken timely before failures can compromise system safety. Overall, these are the main contributions:

- We motivate the need for timing V&V support in NoC-based multicores for CRTES, and how end-to-end execution times fail to detect risky scenarios.
- We define a strategy to accurately estimate contention in NoC-based multicores for CRTES, so that, if high contention occurs, the cause of such contention can be diagnosed.
- We analyze how our proposed approach can be applied to existing NoC specifications.
- We show illustrative applications of our approach and highlight the advantages it brings for accurate modeling of timing interference.

7.2 Timing V&V on NoC-based CRTES

In this chapter, we outline a strategy that must be developed to perform the timing V&V for NoC-based CRTES. In particular, we introduce the system organization, the objective, and the challenges to be faced to realize such a strategy.

7.2.1 System organization

While NoC-based multicores become mandatory to execute safety-related real-time tasks, functional safety requirements impose the use of some hardware support, which is neither available in high-performance multicores, nor expected to be implemented, such as watchdogs and domain-specific safe and secure I/O interfaces. Hence, the usual way to achieve both, functional safety and high performance, consists in connecting a safety-critical MicroController Unit (MCU) as master, and a high-performance device with limited support for safety as slave. This is already the system organization chosen by the Intel-Go platform for the automotive domain [5], where Infineon AURIX and Intel Atom MCUs manage an accelerator card (e.g. based on Intel Arria technology with FPGAs).

7.2.2 The Objective

Timing V&V requires the collection of sufficient evidence to size timing budgets (resource planning), verifying that they hold upon system integration (offline testing), and monitoring during operation (safety measures) that overruns do not occur. Moreover, such timing information must allow diagnosing precisely the cause of an overrun to take corrective actions, both at analysis and during operation, and optimize system integration at analysis time.

To that end, we aim at collecting detailed contention information in the NoC, where all traffic from all PMEs is visible. In particular, we envision collecting information about how much each PME (and so the task using it) delays each other PME, and in what NoC location. This allows obtaining the following certification-, diagnosis-, and optimization-relevant information.

- How much each task uses each part of the NoC, and so, where and to what extent each task is sensitive to experience contention, and by whom. This information is key for WCET estimation to factor in NoC contention.
- How much contention is experienced in practice in each NoC location and what PME caused that contention, either directly or indirectly. During testing, this information helps to gain confidence and produce evidence for certification supporting that overruns cannot occur, or, if they occur during testing, this information will facilitate the diagnosis of the causes, thus helping to fix the issue. Moreover, collecting this information periodically during operation allows for detecting and/or preventing overruns, and eases the process of taking corrective actions so that overruns do not occur again.

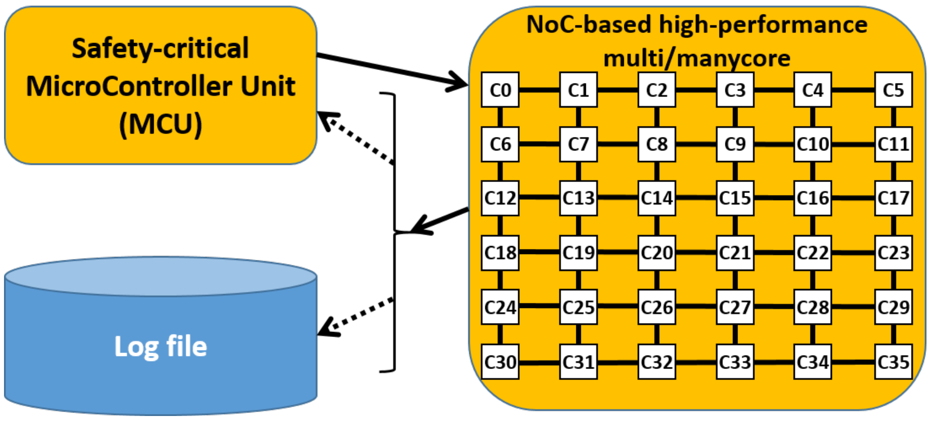


Figure 7.1: General view for resource planning, offline testing and online monitoring.

Overall, the goal is to collect detailed information during resource planning and offline testing. As shown in Figure 7.1, information can be collected (and/or processed) by the MCU or directly dumped into a log file for post-processing. During operation, monitoring must be performed by the MCU periodically.

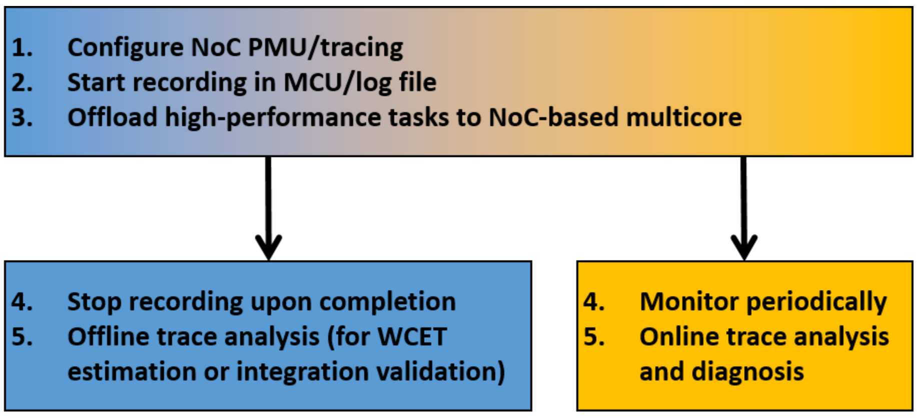

1. Configure NoC PMU/tracing
2. Start recording in MCU/log file
3. Offload high-performance tasks to NoC-based multicore
4. Stop recording upon completion
5. Offline trace analysis (for WCET estimation or integration validation)
4. Monitor periodically
5. Online trace analysis and diagnosis

Figure 7.2: Steps of our strategy for offline/online processes.

The steps of this strategy are summarized in Figure 7.2. First, configuring the PMU and tracing support in the NoC. Second, start recording information, either by sending it back to the MCU or to a log file. And third, offloading the task(s) onto the high-performance NoC-based multicore for its execution. These three steps are shared for resource planning, offline testing, and online monitoring. Then, in the case of resource planning and offline testing, the process of recording information stops upon execution completion, and information collected is processed, either for WCET estimation (resource planning) or for diagnosis/optimization (offline testing). In the case of online monitoring, information is collected by the MCU periodically during the execution of the tasks in the NoC-based multicore, and analyzed online for overrun detection or prevention.

7.2.3 NoC-related Challenges

This strategy based on collecting abundant information, while effective by construction, poses a number of challenges to be tackled before being implemented in practice.

Contention classification. The challenge lies in identifying, for each packet experiencing contention, the packet that causes such contention and providing the contention classification at flow level. This information shows the contention breakdown one flow experiences from the other tasks in the system.

In a NoC, once a request is injected, the information about which initiator (e.g. cores and other I/O masters such as DMAs) is responsible of that petition is lost unless such information is explicitly encapsulated in the request. Actually, in AXI-based NoCs there is no explicit field to incorporate the information about the initiator creating the request. Luckily, more advanced NoC specifications like the AMBA CHI used in the ARM CNM [25] capture that information in one of the mandatory request fields. We elaborate on the accuracy of contention classification for different NoC setups in Section 7.4.

Tracing and monitoring limitations. To properly classify contention, detailed information about when packets arrive at the routers, when they leave, arbitration choices, input/output queues occupancy at the routers, etc. may help to determine what PME causes contention on what other PME and where. In general, either such information is not available or, if available, there are limitations to collecting it. For instance, as discussed before for the case of the CMN NoC, tracing features may provide all information needed to properly classify contention, but not simultaneously due to limited tracing capabilities (i.e. buffering space and BW). Instead,PMUs have the ability to accumulate information in the form of counters at arbitrarily high rates. However, PMUs for decentralized NoCs may lack either sufficient information to classify contention properly, sufficient counters to count all different casuistic, or both of them.

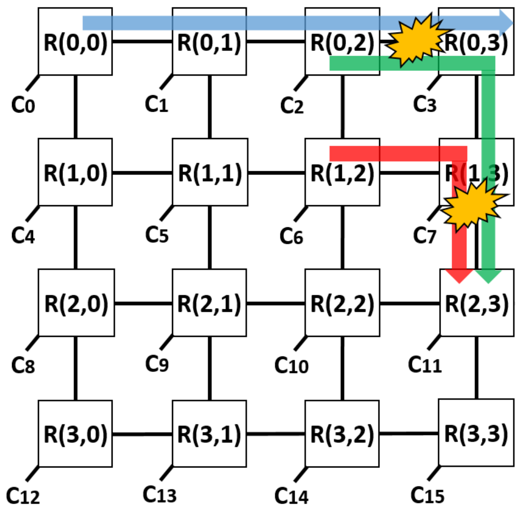
In general, limitations with existing hardware support relate to tracing BW and PMU support. This can be leveraged, to some extent, by deploying software-based solutions able to infer contention details out of limited information (e.g. combining information collected in different runs or incomplete data). To what extent such an approach can be successful is strictly related to the actual hardware support and the computation time that can be devoted to this task offline or online. In Sections 7.3 and 7.4, we analyze and evaluate to what extent contention can be effectively measured with worst-case analysis methods, with basic performance monitors, or using exhaustive tracing information.

7.3 Contention metrics analysis

To analyze the contention we have mainly two different approaches. First, an analytical approach similar to the one presented in [51] to quantify worst-case contention. Second, to collect contention information in the NoC by using NoCs PMUs or using NoC tracing tools.

7.3.1 Worst-Case Delay

Analytical methods to account for the worst-case contention accurately account for the worst-case scenario [51]. However, actual contention is generally much lower than the worst-case one.



(a)

Contention suffered by C_0	From Core	WCD (cycles)	Observed Contention (cycles)
R(0,1)	C2	70	60
	C6	80	55
R(0,2)	C2	50	45
	C6	65	30
TOTAL		**265**	**190**
	C2	**120**	**105**
	C6	**145**	**85**

(b)

Figure 7.3: WCD vs observed contention comparison.

Let us illustrate this with the example in Figure 7.3(a). Consider that flow starting at C_0 (flow of interest) located at $R(0,0)$ (blue flow) sends packets to PME in $R(0,3)$ and C_2 in $R(0,2)$ (green flow) and C_6 in $R(1,2)$ (red flow) (contending flows of C_0) send packets to the PME at $R(2,3)$. To compute the worst contention packets from C_0 can suffer from packets of C_2 and C_6, we traverse all the routers of the flow of interest path looking for flows that physically share resources with C_0. As only C_0 traverses $R(0,0)$ and $R(0,1)$, it cannot suffer contention from any other flow. However, in $R(0,2)$ C_0 flow directly shares the link with C_2 when going to $(0,3)$. So, C_2 causes direct contention to C_0. In order to account for the maximum contention C_2 can create to C_0 we need to consider potential flows that can create contention to C_2 as in turn will be creating contention to C_0. So, taking C_2 as if it was the flow of interest, a direct physical link sharing between C_2 and C_6 is found in $R(1,3)$. Hence, C_0 suffers direct contention from C_2 and indirect contention from C_6 in $R(0,2)$. The same reasoning done for $R(0,2)$ will apply in $R(0,3)$.

Considering which flows interfere with the flow of interest, the routing algorithm and the arbitration protocol used, we can compute the maximum contention a flow can suffer from the other flows. Unfortunately, there is no golden criterion to account for and classify contention for the observed contention that occurs in a real execution. However, if we can access to PMUs or trace information online or offline, we can derive accurately the observed contention breakdown one flow of interest suffers from the other flows in the NoC.

In Figure 7.3(b), we compare the contention breakdown values per core and per router obtained when computing the WCD computed for the Figure 7.3(a) example w.r.t. the contention values obtained when analyzing contention that occurred in an execution trace. Contention breakdown values show that contention values observed in practice are always lower than the computed WCD. The same happens when comparing the TOTAL WCD values (265 cycles) w.r.t. the observed contention values (190 cycles). Moreover, worst-case analysis is not useful to identify which task created more contention to another task in a specific scenario. We observe that behaviour in Figure 7.3(b) TOTAL C_2 and C_6 values since WCD values blame core C_2 from causing most of the contention that C_0 suffers (145 cycles) when in practice is C_2 the one that causes most of the contention to C_0 (105 cycles).

7.3.2 PMU counters

NoCs simulations but also real NoCs as the CMN NoC come with configurable PMCs that allow accumulating pre-configured events that occur in specific routers of the NoC. For instance, CMN NoC provides per task injected packets information in the PMU_XP_TXFLIT_VALID counter. These kinds of counters can provide NoC utilization information if the counter information is collected by the MCU during execution, in a log file, or in the tasks ending. The amount of contention a task experience from its contending tasks is quite related to the NoC utilization these contending tasks make when the delayed task is running. Thus, one can combine PMCs with analytical models [51] to estimate NoC contention. Unfortunately, despite NoC utilization metrics accurately capturing contention effects in highly utilized NoCs, these metrics fail to capture contention in low utilization NoCs.

7.3.3 Trace metrics

Most of the current NoCs as the CMN NoC often offer debug tracing capabilities that allow the creation of watch-point-initiated transactions or event-based interrupts that provide low detail execution information and ease the debug process. These tools also can be used to collect contention information in order to identify the precise cycle, the NoC location, and which packet creates contention with another one. To be able to accurately decompose contention, one of the most relevant information is the packet source information. Even though the last CHI protocol includes a specific request field in NoC petitions to encode the packet source identifier, other AMBA versions and other protocols only provide some width limited fields where this can information can be encoded (e.g. the 4 QoS bits in AXI4). This limitation drastically impacts in the contention breakdown accuracy in systems with more than 16 NoC packet senders.

7.4 Evaluation

In this section, we analyze the impact that different NoC features can have on the accuracy of contention metrics. More in detail, we evaluate the worst-case contention analysis and the measurement-based metrics described in Section 7.3 in different NoC setup scenarios.

7.4.1 Contention Analysis Framework

To analyze the contention, we use two different approaches. First, we use an analytical approach similar to the one presented in [51] to quantify worst-case contention. Second, we derive an iterative methodology that accounts for actual (i.e. not worst-case) contention measurements in the NoC using NoC traces and PMUs counters already presented in Chapter 6.

We evaluate our approach on 2DMesh wNoCs of sizes 4x4 and 6x6 (see NoC setup in Figure 7.3(a) (see TILED MC architecture in Section 3.2). To ease the analysis of the contention occurring in these setups, we build on a 2DMesh wNoCs where all C_X cores target the same Memory module located in R(3,3), in the 4x4 setup, and in R(5,5), in 6x6 mesh size. We assume that all cores in the NoC are running a task and our analysis focuses on the task that runs in C_0 that we name from now on TuA.

For the analytical contention analysis, we computed the WCD packets from the TuA can suffer from packets of all other co-runner tasks in the system as we show in detail in Chapter 4 (see Section 4.2).

To obtain contention measurements, we use gNoCsim [127] modeling the Tile-based architecture described in Section 3.2.

Furthermore, we implemented the PMU_XP_TXFLIT_VALID counter [25] and we use it as a way to approximate how the BW distributes along the mesh (packets_injected_ per_core/total_packets_injected). Then, applying the worst-case contention analytical model, the contention breakdown can be computed using these measurements. We refer to this metric as *PMU*.

Traces have been generated with the simulator containing full information about 3 triggered packet events: packet injection in the NoC, packet input buffer arrival in each router, and packet arrival in the targeted PME. Specifically, full information includes the execution cycles when the event has been triggered and the packet information (source and targeted PME ID and router ID location including virtual channel and virtual network).

Gathered traces information allows for comparing contention-free scenarios with the real scenario traced. Moreover, traced events allow recreating other relevant information to accurately detect and classify contention as buffers occupancy, packets ordering in the buffers and arbitration decisions taken at router level.

So as to evaluate the impact of the packets source 4-bits width encoding of the AXI4 protocol, we created two trace metrics variants: *TS_COMP* that contains the contention breakdown when all packet information is available in the events traced, and *TS_INC* that contains the 4 bit limitation. For TS_INC values, contention information is provided in groups of cores. For example for the 6x6 setup contention is provided for groups C_0-C_2, C_3-C_5, C_6-C_8, C_9-C_{11}, C_{12}-C_{13}, C_{14}-C_{15}, and so on and so forth.

7.4.2 WCD vs Trace vs PMU accuracy impact

Figures 7.4 and 7.5 show the contribution of all tasks in a 6x6 mesh NoC to the worst-case contention suffered by the TuA. As expected, the XY routing and the round-robin arbitration setup give more BW to tasks that are mapped closer to their target in $R(5,5)$ (i.e. C_{35},C_{34},C_{29},C_{33}), which are the ones that can contribute more to the worst-case contention of the TuA.

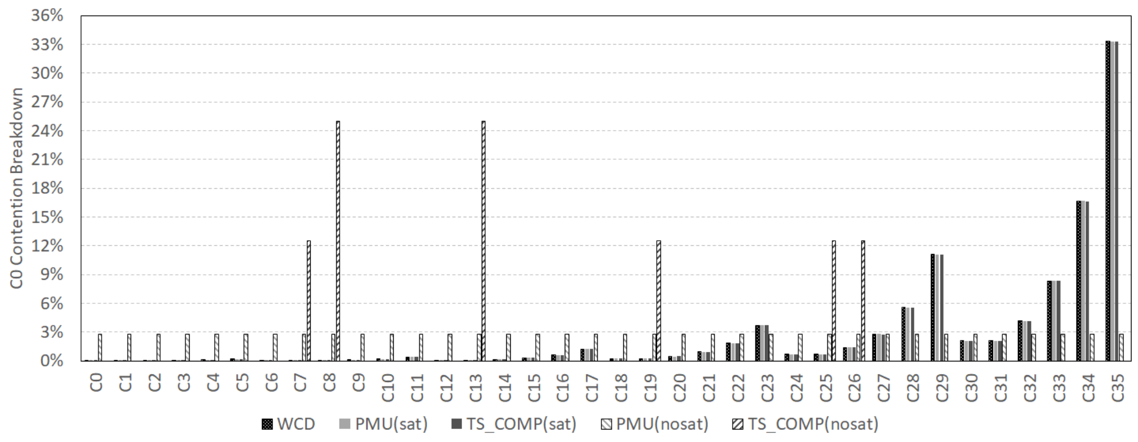


Figure 7.4: 6x6 2DMesh core Trace vs PMU guilty distribution percentages.

When comparing WCD w.r.t PMU and TS_COMP metrics in saturation mode (see Figure 7.4 WCD, PMU(sat) and TS_COMP(sat)), we observe that all metrics are accurate as NoC contention is recreating the worst-case contention scenario. Actually, we observe negligible differences across WCD, PMU and TS_COMP. All metrics show that C_{35} is the core creating the highest contention to the TuA. However, when the NoC works below the saturation threshold (see Figure 7.4 WCD, PMU(nosat) and TS_COMP(nosat)), the PMU(nosat) values no longer represent cores' contention contribution to the TuA as NoC utilization is not bounded to packets contention anymore. PMU(nosat) values show how each core injects ≈3% of the packets in the NoC but only packets from some of the cores (e.g. C_7, C_8, C_{13}) cause contention to C_0 due to their packets arbitration alignment, as shown by TS_COMP(nosat). Similarly, as already explained in Figure 7.3, WCD no longer represents observed contention in the NoC as the cores that potentially can create more contention to the TuA in the worst-case contention scenario (e.g. C_{35}, C_{34}, C_{29}, C_{33} in Figure 7.4) are not the ones that in practice contribute more to the TuA contention (e.g. C_8, C_{13} TS_COMP(nosat) values in Figure 7.4). Trace source complete metric (TS_COMP) is the only contention metric that accurately identifies which tasks contribute more to the TuA contention.

7.4.3 SRC accuracy impact

In Figure 7.5, we show the impact in accuracy when analyzing the contention decomposition of the TuA in a 6x6 mesh size when packets source ID only can be encoded in 4 bits (TS_INC) (e.g., using QoS bits of AXI4) and compare it against the case with unrestricted bits for source ID encoding (TS_COMP). T_INC contention results in the figure are the addition of the contention contribution that all cores have to the C_0 group (C_0, C_1 and C_2) as all of them share the same source ID. T_INC contention of a group is depicted in the first core of the group (e.g., in C_{34} for the group including C_{34} and C_{35}).

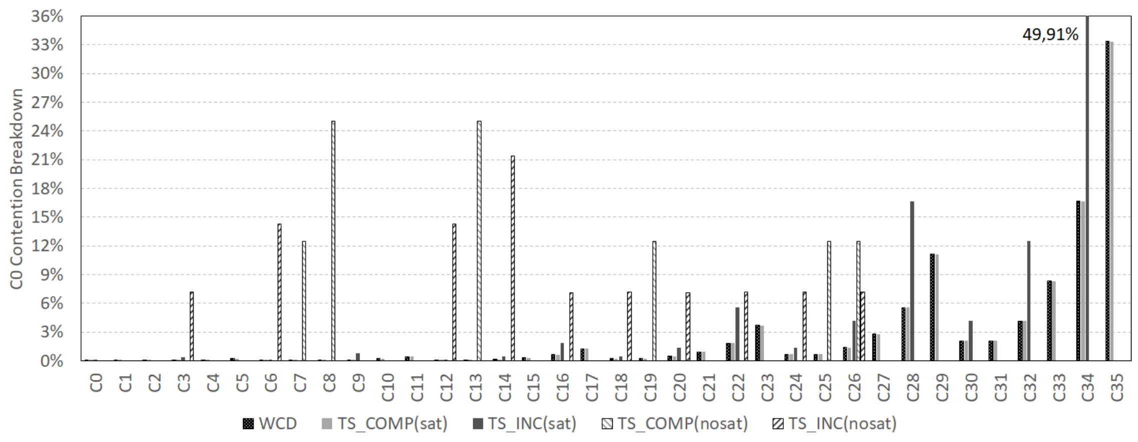


Figure 7.5: 6x6 2DMesh core SRC accuracy impact distribution percentages.

In a saturated NoC (see Figure 7.5 WCD, TS_COMP(sat) and TS_INC(sat)), TS_-INC(sat) values seem to provide quite accurate contention information about each core group contribution to the TuA w.r.t. TS_COMP(sat) values. For instance, C_{34} and C_{35} group contention contribution provided in T_INC(sat) C_{34} matches with the addition of T_COMP(sat) contention contribution of C_{34} and C_{35}. Since all cores in C_0's group have a very similar contention contribution from the other cores of the NoC, results remain being sufficiently accurate.

In a non-saturated NoC (see Figure 7.5 WCD, TS_COMP(nosat) and TS_-INC(nosat)), TS_INC(nosat) values do not match TS_COMP(nosat) values as the group contention breakdown resolution of TS_INC(nosat) is not enough to provide useful contention information. We have also evaluated other grouping setups but the observed results are similar to the ones presented.

7.5 Conclusions

NoC-based multicores and manycores challenge the timing V&V of critical real-time tasks in CRTES, such as those in the avionics and automotive domain, to name a few. Hence, new approaches are needed to achieve high-performance using NoC-based multicores while adhering to safety regulations.

This chapter analyzes the impact of the available NoC-related information on the contention estimates and breakdown, and shows that detailed information (per core, per router, etc.) is key to achieveing sufficient accuracy, as needed for V&V, system optimization, and overrun diagnostics during operation.

Part IV

Contention Aware Optimizations in Enforcement stage

Chapter 8

Maximum-Contention Control Unit (MCCU)

8.1 Introduction

In CRTES, the techniques to derive bounds to the contention tasks can suffer in multicore build on resource quota monitoring and enforcement. Interestingly, SQME have been proposed to handle multicore contention in more generic processors with limited hardware support for time predictability [33, 67]. In general, SQME approaches build on limiting per task (core) maximum shared resources utilization. To that end, the operating system monitors task's activities using the available hardware PMCs and suspends tasks execution when their assigned budget is exhausted. In this chapter, we show that current software-only solutions work well when there is a single resource and type of request to track and bound, but do not scale to the more general case of several shared resources that accept different request types, each with a different associated latency. To handle this (more general) case, we propose a low-overhead hardware support called Maximum-Contention Control Unit (MCCU). The MCCU performs fine-grain tracking of different types of requests, preventing a core to cause more interference on its contenders than budgeted. In this process, the MCCU also helps verifying that individual requests duration does not exceed their theoretical bounds, hence dealing with scenarios in which requests can have an arbitrarily large duration.

In more detail, we make the following contributions to the field of multicore contention bounding in CRTES:

(a) We identify key limitations of SQME. On the one hand, SQME work well only when are applied to one single shared resource in which all request types are assumed to have the same (worst) access latency. However, for the general case where several hardware shared resources accept different request types with different latencies, SQME generate unnecessary interrupts to determine whether tasks have consumed their budget, not only increasing task WCET but also hampering deriving bounds to WCET. On the other hand, SQME build on maximum per-request latency estimates that are derived empirically in real processors [115], but the lack of means to verify that those estimates actual bound maximum latencies which brings some uncertainty on the estimates built on top of those bounds. Moreover, in some AMBA bus implementations, a single request can potentially hold the bus for an unbounded duration. That means that setting quotas on request access counts, as done by SQME, does not suffice to bound contention time.

(b) To overcome the limitations of SQME, we propose to include in multicore processors a Maximum-Contention Control Unit (MCCU). The MCCU is a software-controllable low-overhead hardware unit able to ① accurately handle several resources dealing with several request types. Unlike SQME that may need to trigger several interrupts or perform frequent checks to handle utilization quotas, the MCCU only requires to trigger one interrupt when the quota allocated to a task is actually exhausted, allowing to significantly reduce the overheads of the enforcement mechanism. The MCCU, by monitoring requests duration, also ② allows monitoring seamlessly whether theoretical latency bounds derived empirically are effectively respected at all times and preserves quota enforcement even in the presence of requests with unbounded duration. The MCCU also ③ handles those (AMBA-compliant) scenarios in which a core/master can hold the bus for long time, preventing quota violations.

(c) We show the effectiveness of the MCCU tailoring it for a 4-core multicore setup resembling the Cobham Gaisler NGMP processor for the space domain [50]. Our results for MediaBench show that while SQME can easily generate in the order of dozens of unnecessary interrupts to control quota on access counts, our proposed MCCU generates only interrupts when the task consumes all its contention quota removing any overheads and simplifying timing V&V.

(d) In terms of implementation, the MCCU is a FUB connected as a slave to the AMBA interconnection network, thus it does not require any modification on existing FUBs. In particular, the MCCU interface only needs some addressable space for being configured, snooping AMBA signals and raising specific interrupts whenever appropriate. This is arguably simpler than introducing small modifications in existing FUBs such as caches and memory controllers, which would require expensive and time-consuming re-verification costs.

8.2 SQME Problem Statement

As we introduced in Section 2.2.3.3, existing SQME focus on a single shared resource – usually the memory as it concentrates a big fraction of the contention tasks can suffer – and a single type of request accessing that resource. However, in general, multicores comprise several shared resources $\mathcal{R}$ each of which can accept different types $\mathcal{Y}_r$ of requests. Also requests can have different access times and hence, cause different contention delays. The worst-case (longest) contention τ_b can cause on τ_a, i.e. $\Delta_{b\to a}^{cont}$, is computed as shown Equation 8.1.

$L_{max}^{bus,lh}$	$L_{max}^{bus,sh}$	$L_{max}^{bus,lm}$	$L_{max}^{bus,sm}$
5	10	50	100

Table 8.1: Event and latency values.

$$\Delta_{b\to a}^{cont} = \sum_{r\in\mathcal{R}} \sum_{y\in\mathcal{Y}_r} cr_{b\to a}^{r,y} \times L_{max}^{r,y} \tag{8.1}$$

The contention τ_a can suffer from its contenders $c(\tau_a)$ is:

$$\Delta_{a}^{cont} = \sum_{\tau_x\in c(\tau_a)} \sum_{r\in\mathcal{R}} \sum_{y\in\mathcal{Y}_r} cr_{x\to a}^{r,y} \times L_{max}^{r,y} \tag{8.2}$$

For the single request type model, a contender task τ_b is suspended once it consumes its quota, i.e. it performs $cr_{b\to a}$ accesses. In other words, if τ_b requests are below $cr_{b\to a}$, its contention time quota is not exhausted.

When there are several types of requests, there are several combinations of event counts that lead to a quota violation and hence, event counts need to be conservative to detect any potential violation. This is better explained with an example. Let $e_i^{r,y}$ be the event monitor that counts the number of accesses of type y to resource r from core i and $eb_i^{r,y}$ the respective budget allocated to a given task. For the sake of this example, let us assume a multicore processor with the bus connecting the cores with a shared L2 cache as the only shared resource. Further assume that the L2 is partitioned and each core accesses its own private memory controller, so contention can only happen in the bus. Four types of requests can be sent to the bus: load or store (write) accesses that can hit or miss in the L2 (lh, sh, lm, sm). Each of these requests has a different maximum latency as shown in Table 8.1. Finally, let us assume that τ_b is assigned a contention budget $\Delta_{b\to a}^{cont} = 2000$ cycles.

Iteration 1: Row 1 in Table 8.2 (quota columns) shows one potential way in which quotas on access counts can be set to prevent τ_b to cause at most $\Delta_{b\to a}^{cont}$ contention cycles on τ_a. In particular ($eb^{bus,lh}$=20, $eb^{bus,sh}$=40, $eb^{bus,lm}$=14, $eb^{bus,sm} = 8$) that for short we represent as (20,40,14,8). Once these values are programmed in the PMCs, τ_b is allowed to run concurrently with τ_a.

Let assume that, during its execution, τ_b makes accesses ($ev^{bus,lh}$=20,$ev^{bus,sh}$=10, $ev^{bus,lm}$=2, $ev^{bus,sm} = 1$) that we represent as $(20, 10, 2, 1)$. When τ_b makes its 20th load hit access to the bus, an interrupt is raised since the $eb^{bus,lh}$ quota is exhausted. The remaining contention budget is derived as:

$$\Delta_{b\to a}^{cont} = 2000 - (20 \times 5 + 10 \times 10 + 2 \times 50 + 1 \times 100) = 1600$$

Iteration 2: Row 2 in Table 8.2, under quota, shows one potential way in which quotas on access counts can be set so that once τ_b runs again, it cannot create more contention than allowed, i.e. $\Delta_{b\to a}^{cont} \leq 1600$: (16, 22, 8, 9). At the end of this second step the accesses performed by τ_b are (6,22,5,2) leaving a contention budget of $1600 - 700 = 900$ cycles.

Quota ($eb^{bus,xx}$)				Consumed ($ev^{bus,yy}$)				$\Delta^{cont}_{b \to a}$
xx=lh	xx=sh	xx=lm	xx=sm	yy=lh	yy=sh	yy=lm	yy=sm	
20	40	14	8	20	10	2	1	1600
16	22	8	9	6	22	5	2	900
10	15	6	4	8	15	4	3	210
4	4	1	1	3	2	0	1	75

Table 8.2: Example of the evolution of the quota assigned to τ_b. Each row corresponds to an *iteration* of the SQME solutions.

This process repeats, generating an interrupt per iteration, until the contention budget is lower than the highest L_{max}.

In general, the need for pre-programming all $eb_i^{r,y}$ leads to cases where one counter is exhausted earlier than the others, thus raising an interrupt *despite contention budget is not yet exhausted*. This process repeats until the remaining budget is small enough not to be worth continuing with the execution. In fact, the smaller the remaining quota is, the more often interrupts are generated normally. Overall, monitoring the use of multiple shared resources in software (i.e. with SQME) just triggering interrupts only when actually exceeding contention quota is not possible. This results in significant overheads as we quantify in Section 8.4.

8.3 MCCU

We propose the MCCU, a software-controllable low-overhead hardware unit to accurately handle several shared resources dealing with several request types, each with different access latency. We implement the MCCU as a new SoC component that is attached to the on-chip bus (e.g. AMBA). This avoids introducing additional modifications to the rest of processor components (FUBs), thus drastically reducing the costs of re-design and re-verification.

8.3.1 Control logic and hardware

For each resource and request type to be tracked, the MCCU keeps $L_{max}^{r,y}$ in a register so a total of $||\mathcal{R}|| \cdot ||\mathcal{Y}_r||$ registers are needed, see Figure 8.1. Also, one quota register per core is required to save the remaining quota cycles Δ^{cont}.

Prior to the execution of the program (e.g. at boot time), all $L_{max}^{r,y}$ are initialized ① sequentially via a single write port, see Figure 8.1. Since this step is done once, its overhead – in the order of dozens or hundreds of processor cycles – is negligible.

At task boundary, or software partition boundary like those in avionics ARINC 653 [143], ② the Operating System (OS) programs the MCCU with the corresponding quota, Δ^{cont}. The MCCU can be attached to the on-chip bus as a slave, e.g. AMBA Peripheral Bus (APB) slave, so that the quota counter and the registers storing the contention associated to each resource and access type can be configured and accessed using specific memory addresses, the APB addressable space.

While less frequent in CRTES, some deployments may allow task migration across cores. This requires that, whenever a task is swapped out, the OS saves its quota register in the corresponding `task struct` that the OS uses for the task. When the task is swapped back in, the OS restores the quota register. Note that $L^{r,y}_{max}$ values are intrinsic to the processor and hence, the OS can keep a single copy that can restore at boot time.

Upon each access of type y to resource r ($< r, y >$), the corresponding $L^{r,y}_{max}$ latency for that access is retrieved ③. Then, ④ the upper-bound latency of the access is subtracted from the remaining quota, which is properly updated. If the quota remaining is zero or it is close enough (e.g. it is below the latency of some access types), ⑤ an interrupt is triggered by forwarding this signal to the interrupt controller, indicating that the quota of τ_b has been exhausted.

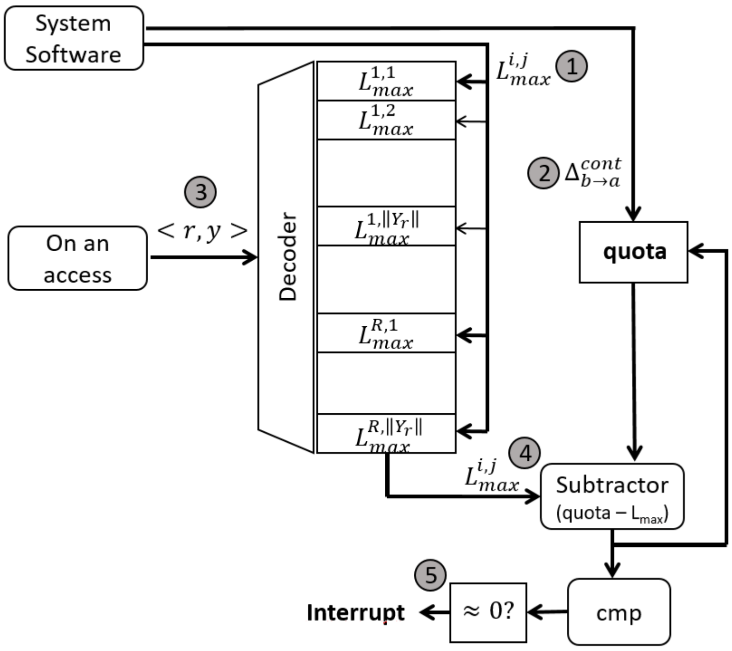


Figure 8.1: Main blocks of the MCCU.

8.3.2 Extension for Several Contenders

Interestingly, when dealing with several tasks, the table with maximum contention values per resource and access type does not need to be replicated. Instead, we only need to set a quota register per task, and use the task identifier (AMBA master signal) to retrieve the appropriate quota value and update it conveniently, see Figure 8.2. This approach allows setting specific quotas for each task so that, if one of them overruns its budget, the appropriate interrupt is triggered, but the other tasks remain unaffected. This approach is also suitable for scenarios with several critical tasks and contenders since the MCCU allows to set quotas for all (monitored) contending tasks and the maximum contention they could generate is independent of the particular TuA. Hence, the MCCU does not monitor contention for each TuA and contender task pair, but instead, it only monitors contention generated per task individually. For instance, let us assume a 4-core processor where τ_a is allowed to run without any quota control, τ_b – despite being critical – has a specific quota to limit the contention it may cause on τ_a (e.g. 10,000 cycles), and τ_c and τ_d are non-critical tasks.

In this context, contention quota for τ_c and τ_d must be set as the lowest contention they are allowed to produce individually on τ_a and τ_b. Also τ_b quota corresponds to the maximum contention it is allowed to cause on τ_a. Finally, since τ_a has no quota, it is programmed either with a special value so that it is ignored, or it is simply set to the maximum value possible (e.g. $2^{32} - 1$) so that it cannot overrun its quota in practice. Overall the contention each task τ_x can generate ($\Delta_{x\rightarrow}$) is:

$$\Delta_{a\rightarrow} = 2^{32} - 1 \qquad \Delta_{b\rightarrow} = 10,000;$$
$$\Delta_{c\rightarrow} = min(\Delta_{c\rightarrow a}, \Delta_{c\rightarrow b}); \quad \Delta_{d\rightarrow} = min(\Delta_{d\rightarrow a}, \Delta_{d\rightarrow b})$$

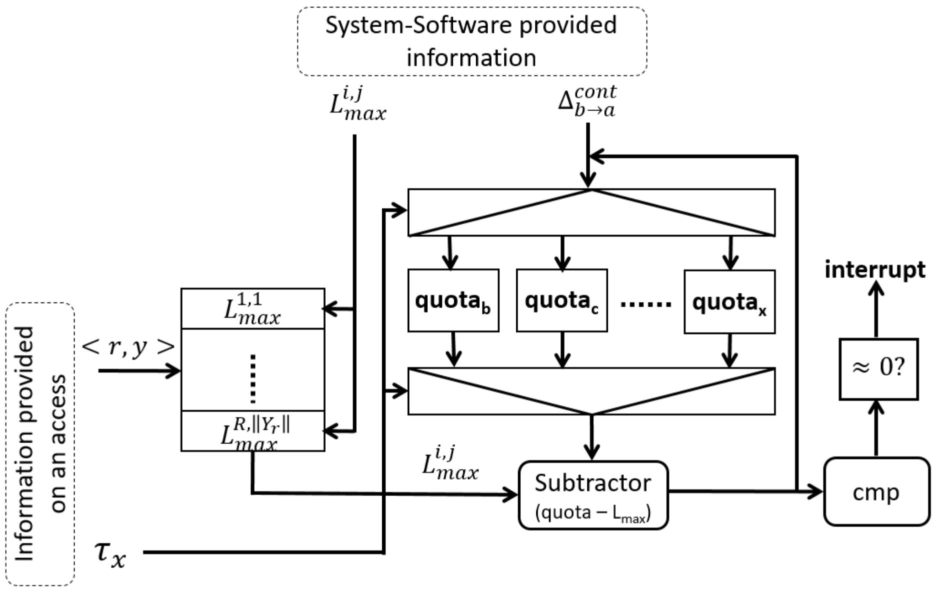

System-Software provided information
$L_{max}^{i,j}$
$\Delta_{b\rightarrow a}^{cont}$
Information provided on an access
$< r, y >$
$L_{max}^{1,1}$
$L_{max}^{R,\|Y_r\|}$
τ_x
quota$_b$
quota$_c$
quota$_x$
$L_{max}^{i,j}$
Subtractor (quota – L$_{max}$)
cmp
≈ 0?
interrupt

Figure 8.2: MCCU for multiple tasks.

8.3.3 Seamless verification of $L_{max}^{i,j}$ bounds

$L_{max}^{i,j}$ is estimated empirically in real architectures, which always leaves some residual risk that estimates do not bound maximum latencies. While extensive testing helps reducing this residual risk, the MCCU allows for a seamless verification of $L_{max}^{i,j}$ bounds with a minor extension. In particular, upon each access to shared resources, the MCCU receives as input the request type and retrieves the corresponding $L_{max}^{i,j}$ bound. As shown in the bottom part of Figure 8.3, by simply monitoring when requests start and finish with a request duration counter, rdc, the request duration can be compared with $L_{max}^{i,j}$ and, upon an exceedance, raise an interrupt.

This allows for detecting inaccuracies in the derivation of $L^{r,y}_{max}$ that might not be detected during the testing phase, which could affect the reliability of the software-estimated quotas.

In the context of AMBA, *locked* transfers allow masters to keep the ownership of the bus until the transfer completes. This enables masters to keep the bus locked indefinitely, without the arbiter being able to relinquish the grant from that master. Lock transfers are typically used to manage atomic operations such as read-modify-write, needed for synchronization. To handle locked transfers with the MCCU, we can set up a specific L_{max} register for locked transfers (or several of them if multiple types of locked transfers exist), and manage them as any other type of request. Additionally, since locked requests may be unbounded, we keep track of the duration of the in-flight request with a request duration counter (rdc), see Figure 8.3. At request completion time, we check whether it exceeds the corresponding $L^{r,y}_{max}$. We also check whether the locked request takes longer than *any* $L^{r,y}_{max}$ even if it is not yet finished. In both cases, an interrupt is raised, indicating that evidence has been found that the derived bounds have been violated. This is fundamental to increase confidence – reduce the residual risk in accordance with standards such as ISO 26262 in automotive.

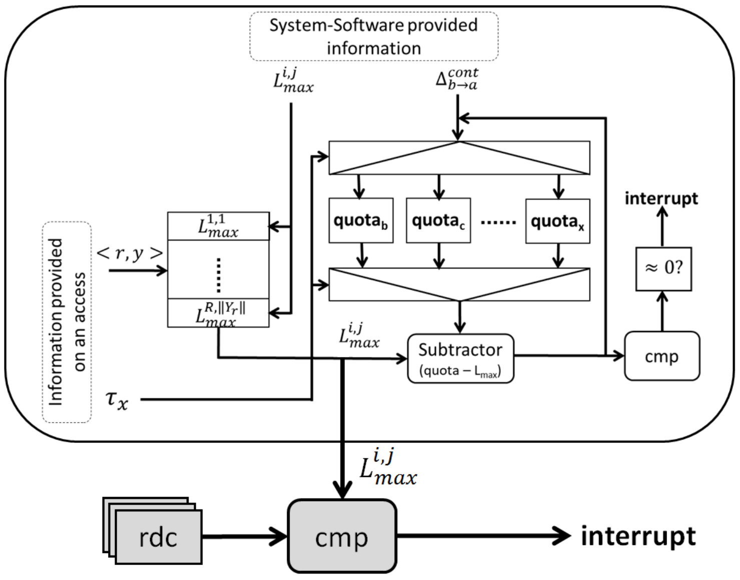


Figure 8.3: Blocks to check $L^{i,j}_{max}$ is not exceeded.

8.3.4 Hardware Cost Analysis

Area. the MCCU requires ① a $L^{r,y}_{max}$ register for each resource and request type to be tracked. Note however, that registers are not replicated per core. Hence the number of registers needed is given by $\Delta^{cont}_{b\to a}$ and $||\mathcal{R}|| \cdot ||\mathcal{Y}_r||$. In addition, a ② 'quota' register is needed per core to keep the contention budget each core has available. Also ③ a request duration counter, rdc, per core is needed. In our reference SoC architecture, see Figure 8.4, the access to the bus does not implement split bus transactions so that by controlling the access to the bus we also control the access to memory. If split bus transactions are implemented, then the MCCU is also connected to the L2-to-memory interface for monitoring purposes. The L2 cache is partitioned as it is the case in many commercial architectures (e.g. NGMP [50] and ARM A9 [144]). Overall the MCCU tracks 4 bus access types: loads/stores that hit/miss in the L2. The 4 L_{max} registers (often tens of cycles) are normally encoded with just 1 byte each. Also, few bytes are needed (e.g. 4) for the quota register for each of the $Nc = 4$ cores. Thus, hardware overhead is negligible (up to 24 bytes in our reference SoC: 2×4 bytes for maximum latencies and 4×4 bytes for quota registers).

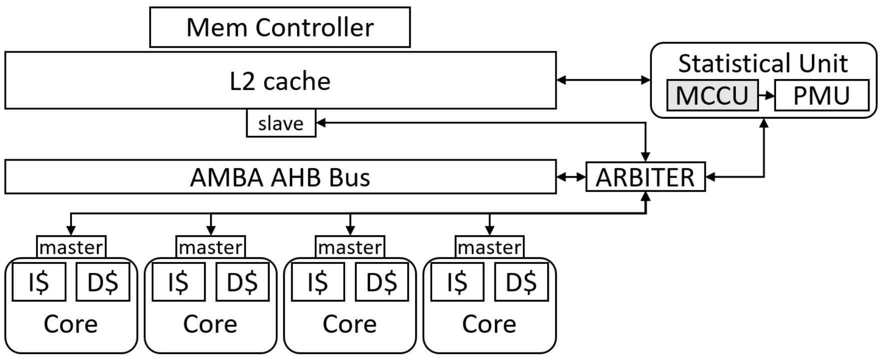


Figure 8.4: Reference SoC architecture. PMU keeps the events and PMCs.

Delay. As shown in Figure 8.3, combinational logic consists of few decoders, multiplexors and comparators, as well as a subtractor one of whose operands (L_{max}) has few bits (at most 1 byte). Hence, the area and latency of this logic is well below that of an integer Arithmetic Logic Unit, whose number, size and complexity of combinational blocks is fairly larger.

The use of a MCCU has other implications in terms of latency similar in nature to those of a PMU, but of lower magnitude. In particular, a PMU has low complexity itself, but its input consists of signals monitoring events occurring all along the processor, thus potentially taking several cycles to arrive to the PMU, which leads to event counts that can never be up-to-date and reflect event counts with some (yet limited) delay. In the case of the MCCU, events monitored are typically available as part of the AMBA AHB bus interface shown in Figure 8.4. Thus, connecting the MCCU to the AHB, either directly as a slave or as part of another slave (e.g. along with the PMU), makes input signals available almost immediately when they arrive at the AHB bus. The internal processing of the MCCU, Figure 8.3, is simple enough to fit in one cycle or, if needed, be pipelined when the target processor frequency is high. Finally, on a quota violation there may be some delay to propagate the interrupt to the interrupt controller which, again, is limited to very few cycles in practice.

Overall, the end-to-end delay since an access starts until a potential interrupt is triggered may be in the order of 10 cycles, which could at most allow another access (e.g. the slowest one) to start, thus leading to a slight quota overrun of some tens of cycles. In the context of microcontrollers operating at several hundreds of MHz at least, such quota overrun could cost at most around 100ns, which is an insignificant impact for systems whose response time is in the order of hundreds of milliseconds (e.g. a braking system).

Scalability. For larger multicores, the $L_{max}^{r,y}$ registers do not depend on the number of cores, but on the number and type of requests. Only one quota register is needed per core. Hence, the absolute hardware cost increases negligibly. Latency wise, signal propagation may increase MCCU latencies. Also, MCCU accesses latency can increase due to access serialization to the MCCU to keep a single read/write port. This can delay by few cycles MCCU access, in the worst case, when several cores try to access the MCCU simultaneously. In both cases, the impact of the increase of few processor cycles is negligible as described in the previous paragraph.

8.4 Evaluation

8.4.1 Experimental framework

We evaluate the MCCU (see Figure 8.4) on the reference NGMP multicore architecture [50] described in Section 3.2. In particular, we use its implementation in a SoCLib-based [126] (see Section 3.3) that allows the L2 cache to be way-partitioned so that each core receives one way.

The MCCU tracks the 4 access types in Section 8.2: lh, sh, lm and sm. Since the L2 is writeback, L2 misses evicting dirty data generate two memory accesses: one to write back dirty data and one to retrieve the data requested.

Following the methodology of previous works [21, 115], the latencies for the different types of request to the bus have been derived empirically, resulting in these values: $L_{max}^{bus,lh} = 10$, $L_{max}^{bus,sh} = 3$, $L_{max}^{bus,lm} = 32$, and $L_{max}^{bus,sm} = 37$.

We use the MediaBench benchmark suite [130] described in Section 3.4. We only excluded `mpeg2.encode` benchmark due to issues executing it in our reference platform.

8.4.2 Scenarios evaluated

As explained in Section 8.3.2, the contention quota assigned to each (contender) task is determined by the minimum amount of contention its sibling tasks can afford.

We analyze several scenarios in which each (contender) task is allowed to cause variable contention on the TuA. In some scenarios, the contender task does not exceed its assigned contention quota (and hence should not be suspended). In particular, we allocate a quota 5%, 10%, ...25% higher than needed, represented in the corresponding figures as 1.05, 1.1, ... 1.25 respectively. In other scenarios, the contender task exceeds its quota, so it must be suspended. In particular, it is allocated a quota smaller than the actual contention it generates according to the model (Equation 8.2). In particular, we cover the values 0.8, 0.85, 0.9, 0.95, 1.0.

For SQME, given a maximum contention budget τ_b can generate on τ_a, $\Delta_{b \to a}^{cont}$, there are several ways in which bounds to access counts can be assigned. For instance, assuming a single resource with two request types of latencies 10 and 20 cycles, and a contention quota of 100 cycles, the possible ways in which bounds to access counts can be assigned are: $(10, 0), (8, 1), (6, 2), (4, 3), (2, 4), (0, 5)$. Determining the best access bounds a priori is challenging since the number of accesses of each type of a program can depend on factors such as input data and vary throughout program execution. Hence, any a-priori choice, which we refer to as request bound breakdown scenarios ($rbbs$), is arbitrary.

We have explored two $rbbs$. In both cases, in the first iteration, we distribute contention quota homogeneously across request types. Hence, each request type y_r is given a *FairShare* of the overall quota: $FairShare = (\Delta^{cont}_{b\to a})/(||\mathcal{R}|| \cdot ||\mathcal{Y}_{\nabla}||)$.

For each request type $eb^{r,y}$, requests are allowed as defined next: $eb^{r,y} = \lfloor FairShare/L^{r,y}_{max} \rfloor$. Whenever the quota for a given request type is exhausted, in the following iterations we either follow the same approach based on a fair quota share ($rbbs_{fair-fair}$), or we assign the quota proportionally to the consumed quota across accesses so far ($rbbs_{fair-prop}$).

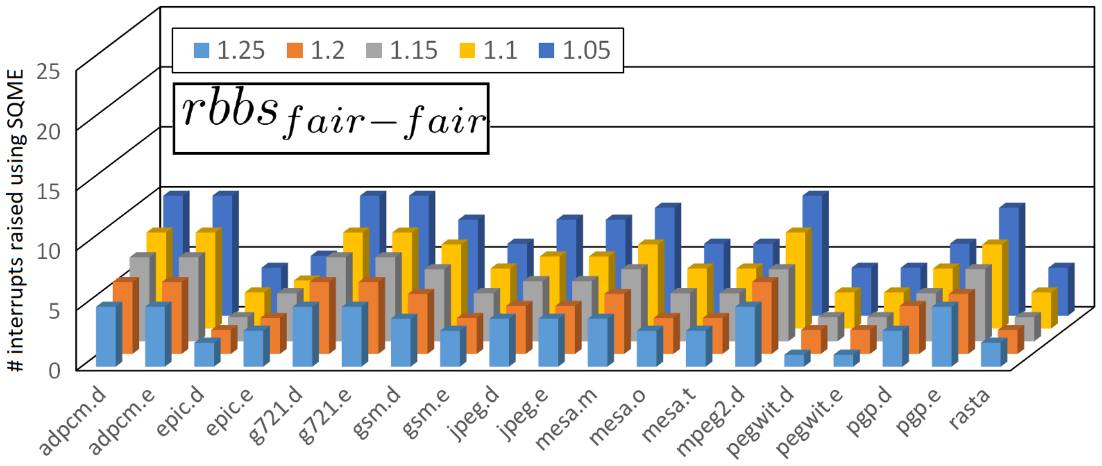
For instance, let us assume that the quota is 1,000 cycles and two request types with $L^{r,y}_{max}$ 10 and 20 cycles respectively. In this case, we allocate 500 cycles to each request type in the first iteration, thus granting 50 and 25 accesses respectively. Let us now assume that the task performs 50 and 5 accesses respectively, thus exhausting the quota for the first access type. The remaining quota would be 400 cycles. Under $rbbs_{fair-fair}$, we would allocate 200 cycles to each request type, so 20 and 10 accesses. Under $rbbs_{fair-prop}$, instead, we would take into account that each access type has used 500 and 100 cycles respectively, so we would split the remaining 400 cycles keeping the same ratio, so 333 and 67, thus granting 33 and 6 accesses of each type respectively.

8.4.3 Sufficient contention quota

SQME results

Figure 8.5 shows the number of interrupts raised by each task under the different scenarios in which tasks have sufficient budget. It is worth mentioning that the latency of an interrupt is the time between the start of an Interrupt Request (IRQ) and the completion of the respective Interrupt Service Routine (ISR). The direct cost of interrupts together with their impact on processor state (e.g. cache state) impact execution time (and WCET) in non-obvious ways. Moreover, just predicting a priori the number of interrupts to account for them in the WCET is challenging. Therefore, in addition to their actual cost, unnecessary interrupts caused by SQME challenge the derivation of reliable and tight WCET estimates.

In Figure 8.5, the z-axis shows the quotas that range from 1.05x to 1.25x of the actual quota each benchmark (on the x-axis) would need in practice. As shown, the number of unnecessarily-generated interrupts is high for both $rbbs_{fair-fair}$ (top plot) and $rbbs_{fair-prop}$ (bottom plot) software approaches and increases as the contention quota decreases down to 1.05. For $rbbs_{fair-fair}$, interrupts are quite stable around 5, while for $rbbs_{fair-prop}$, on average, the number of interrupts decreases, and in many cases is just 1. For some benchmarks such as `epic.d` and `pgp.e`, they increase



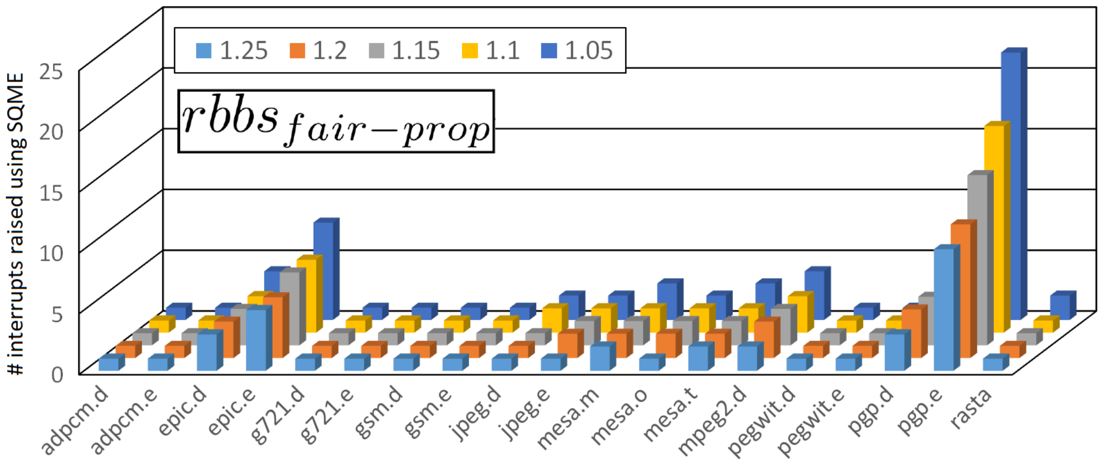


Figure 8.5: Number of interrupts generated by SW solutions.

significantly due to the varying behavior of the benchmarks over time. In either case, software-only solutions (SQME) need to allocate quota statically to request types, thus causing unnecessary interrupts due to their inability to allocate quota dynamically to the request types that occur in practice.

Resorting back to the single-request solution. A simple approach overcomes the problem of distributing the quota across request types, consists of assuming a minimal hardware support that adds the desired events in a given counter. In our case, $ev^{comb} = ev^{lh} + ev^{sh} + ev^{lm} + ev^{sm}$. That is, all requests are assumed to be of the same type. In the CRTES domain, this necessarily means to assume that all requests are from the worst type, so with the highest latency, which requires reformulating Equation 8.2 as follows.

$$\Delta_{b\to a}^{cont} = \max_{r\in\mathcal{R}, y\in\mathcal{Y}_r} (L_{max}^{r,y}) \times \left(\sum_{r\in\mathcal{R}} \sum_{y\in\mathcal{Y}_r} cr_{b\to a}^{r,y} \right) \tag{8.3}$$

While this solution would certainly avoid generating unnecessary interrupts, assuming that all requests are from the worst type implies that quota is consumed – pessimistically – much faster, leading to a single (but much earlier) interrupt.

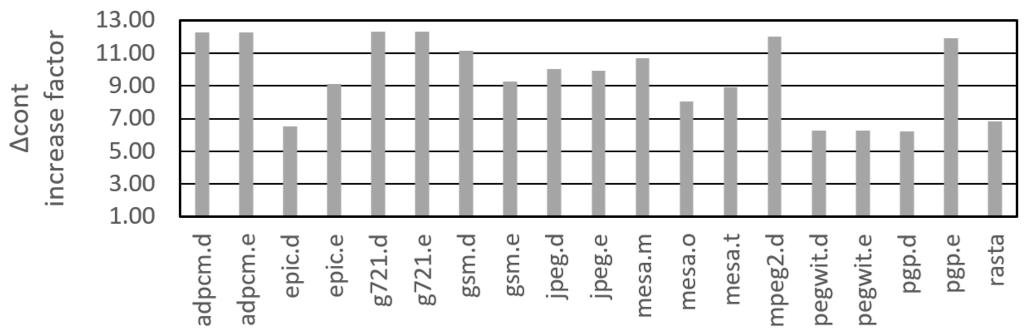


Figure 8.6: Increase in $\Delta_{b\to a}^{cont}$ due to the simple support for the single-request approach w.r.t. MCCU.

In order to show the impact of this approach in our reference architecture, we have measured in Figure 8.6 the increment in contention delay, $\Delta_{b\to a}^{cont}$ in Equation 8.3, with respect to Equation 8.1, the latter of which is captured by the hardware model in the MCCU.

The observed increase for MediaBench in our reference architecture is $9.6x$ on average and as high as $12.3x$. This indicates that quota is consumed at a $9.6x$ higher rate with this approach, which is simply an unaffordable overhead that makes this solution not attractive either. In fact, this would require that the TuA could afford $9.6x$ higher contention than with the MCCU.

MCCU

For the MCCU, zero unnecessary interrupts are generated. Since quota suffices, the contender task will always complete its execution before exhausting its quota.

8.5 Conclusions

Existing SQME techniques focus on reducing the impact of contention on WCET estimates of monitored tasks when accessing on a single shared resource with a single type of request accessing this resource (e.i same latency). Nevertheless, they do not scale well when tracking contention coming from multiple shared resources each with multiple petition types. To cope with this case, we have presented the MCCU, a low-overhead hardware component that allows allocating contention quotas to tasks for shared hardware resources and monitoring whether those quotas are exhausted. The MCCU raises one interrupt only when strictly needed, hence avoiding the limitations of software-only solutions that allocate individual quotas per resource and access type resulting in frequent unnecessary interrupts. Moreover, the MCCU allows assessing seamlessly whether the maximum latency per request type estimated is exceeded, thus detecting potential issues in the WCET estimation process. Finally, the hardware overhead of the MCCU is quite limited and does not require re-designing or re-verifying existing FUBs, which would challenge its adoption due to cost reasons.

Part V

Conclusions and Future Work

Chapter 9

Conclusions and Future Work

Satisfying the increasing demand for performance in CRTES calls for the use of NoC-based multicores and manycores. While these kinds of systems are popular in the high-performance domain due to their high average performance, they challenge the entire V&V process and hamper deriving evidence of systems' timing and functional correctness.

Focusing on the timing correctness, the adoption of COTS high-performance hardware and particularly of interconnects, specially NoCs, challenges the entire software timing development process:

- timing analysis when deriving tight WCET estimates during budgeting and verification,
- timing violations detection and correction during validation,
- deploying safety measures (i.e. enforcement) to prevent timing failures at operation.

These difficulties are caused by the contention to which tasks are exposed in the interconnect, which tasks execution time behavior dependent on each other, and are exacerbated by the distributed nature of NoCs.

9.1 Contributions

In this Thesis, we tackle all these three challenges by developing strategies to tightly derive WCET estimates that favor systems' performance fairness and by proposing practical solutions to monitor and control contention in CRTES. More in detail:

Timing Validation (WCET reduction)

- Our first contribution, EOmesh, overcomes the inability of wmeshes trying to fairly distribute BW to minimize WCET estimates when using XY routing. Our proposal uses a smart combination of XY-YX routing that correctly balances the number of flows mapped in each router along with a weighted arbitration re-computation that fairly distributes (global) BW increasing performance and reducing WCET of single-thread and parallel applications. EOmesh also alleviates the number of flit stalls by correctly mapping in a balanced way the flows in the mesh (XY-YX routing) and accordingly recomputing the arbitration weights. These changes have a noticeable impact in reducing the obtained WCD and WCET while increasing the system's guaranteed performance.

- On our second contribution, NoCo, we have gone one step further in reducing WCET estimates in 2Dmesh-based CRTES. While other solutions only target optimizing one or two NoC parameters at the same time, our NoCo solution is a stochastic ILP optimization that finds mesh NoC configurations with minimal WCD by optimizing the three main NoC parameters (tasks' mapping, NoC routing and arbitration weights allocation) at the same time. With this work, we can conclude that minimal WCD and WCET can only be obtained when optimizing mapping, routing and arbitration weights at the same time after exhaustively modeling their interrelations and feasible NoC configurations.

Timing Verification (Monitoring, testing and contention analysis)

- In our third contribution, PWC-GRV, we focus on defining techniques for measuring and analyzing actual contention in 2Dmesh NoC-based systems. Whereas a lot of research has been done to tightly derive WCET, less effort has been devoted to developing techniques that monitor and control that estimated budgets are met in the testing phase. Since the budgeting techniques are not valid to measure actual contention in NoCs, our work develops a new metric (PWC) that bounds packets' slowdown from one task (contention) to packets from other tasks causing the slowdown. Furthermore, we propose a criterion (GRV) that allows breaking down contention suffered by a task distinguishing the source of the task causing the contention and the location in the NoCs where the contention occurs. With this work, we propose to our knowledge, one of the first methodologies to measure and classify actual contention for detection and correction testing purposes in mesh NoCs. We can conclude that monitoring and classifying contention in NoCs is possible when fine-grain information about the packets' status along the NoC can be traced or monitored. PWC-GRV contention classification proves to be useful both in simple and complex NoC setups with multiple memory controllers.
- On the fourth contribution, ACMNoC, we identify monitoring and tracing limitations in real NoCs when building contention metrics, and we quantify the accuracy impact these limitations have on the contention metrics used to detect and correct timing violations. We also compare our PWC-GRV methodology w.r.t other typical NoC metrics. We conclude that only when fine-grain information is available (per core, per router, etc.) at a sufficient speed, contention breakdowns obtained are accurate enough to be useful for V&V and timing diagnostics.

Safety Measures (Enforcement, monitoring and contention control)

- Our last contribution, MCCU, focuses on contention monitoring and enforcement in bus-based CRTES. We propose a SW/HW solution that tracks and limits fine-grain contention preventing cores to create more contention than the one budgeted to other cores. MCCU improves existing SQME techniques as it only raises an interrupt when the global quota is finally exhausted. With this work, we can show that solutions to enforce contention such as MCCU, can be achieved by introducing limited hardware overhead without requiring re-designing or re-verifying existing FUBs.

Overall, the set of proposed solutions ease the adoption of NoC interconnects to fully exploit its performance benefits and contain the impact they have on timing budgeting and verification, time validation, and timing enforcement.

9.2 Impact and Future Work

The work done in this Thesis complements and extends some research lines already opened by other PhD students in the Universitat Politècnica de Catalunya (UPC) that were also targeting the same challenges in CRTES. At the same time, this Thesis also opens other new research lines regarding monitoring, enforcing and analyzing contention in distributed NoCs. The work carried out in this Thesis has had a significant impact in the scientific community as reflected by the high-quality publications carried out that are summarized in Table 1.1.

The *EOmesh* and *NoCo* proposals are extensions to the previous works carried out on wmeshes in the same research group. This work has been done related to the real challenge that processor designers will face when adopting NoCs in their CRTES and then try to obtain evidence of the systems' timing correctness required by CRTES before deployment. The results of this Thesis will be very valuable to the CRTES industry when moving from simple multicores to higher node count manycore platforms.

The work presented in proposals *PWC-GRV* and *ACMNoC* represents a new research line started in the scope of the EPI [29]. The goal of this research was to analyze the timing behavior of the interconnects of the EPI GPP to ease its applicability in the context of safety-related applications. These two contributions are a starting point for further works as it sets the baseline for monitoring and analyzing contention in distributed NoCs. More in detail, our work provides hints and shows that indeed is it possible to analyze timing behavior and contention at fine grain by monitoring NoCs for validation purposes. However, there is still a long way to go in implementing these metrics and methodologies in real NoCs facing the tracing/monitoring limitations and using the currently available information provided by the NoCs. With the work done in this Thesis, we expect to somehow influence the design of the next-generation GPP EPI processors. We have shown that including or extending PMCs that take into account the basic information needed to create the timing breakdowns has little overheads and enables the possibility of using such devices in the context of time-critical applications.

The work done in the *MCCU* proposal has pioneered the design of hardware quota monitoring units, and has set the ground for other works [145, 146, 147]. Actually, the MCCU mechanism has been implemented in RTL in multiple industry-ready platforms supported by partners like Cobham Gaisler in the De-RISC [148] and SELENE SoCs [149]. This mechanism is currently being tested by relevant industrial partners in the safety-critical systems domain in the context of the H2020 SELENE project [150]. Thus, we consider that is very likely that the MCCU or a similar mechanism will be incorporated in the next generation of SoCs targeting safety-related applications. Furthermore, the MCCU in combination with the analysis we have done at X and Y sets the ground for the design of new enforcement techniques to control and monitor the contention in distributed NoCs. We consider this new line of research that originates from this Thesis will be very relevant in the near future.

Bibliography

[1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3), may 2008. xi, 18

[2] NXP Semiconductors. *T2080 Product Brief*, 2014. `https://www.nxp.com/docs/en/product-brief/T2080PB.pdf`. xi, 20

[3] Arm Developers. *Fixed Virtual Platforms (FVP) Reference Guide*, 2020. `https://documentation-service.arm.com/static/5f4d313eca7b6a33993762d1?token=`. xi, 20

[4] *Embedded Systems: Technologies and Markets*. `https://www.bccresearch.com/market-research/information-technology/embedded-systems-techs-markets-report.html`. 3

[5] Intel Company. Intel go automated driving solution product brief. 2017. `https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/01/intel-go-product-brief.pdf`. 3, 8, 21, 41, 63, 116

[6] S. Ramos and T. Hoefler. Capability models for manycore memory systems: A case-study with Xeon Phi KNL. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 297–306, May 2017. 3, 5, 21, 22, 51, 59

[7] Kalray Inc. Kalray mppa® manycore. a massively parallel processor array architecture. 2021. `https://www.kalrayinc.com/products/mppa-technology`. 3, 6, 21, 83, 84, 101

[8] E. Francis. Autonomous cars: no longer just science fiction. *Automotive Industries*, 193, 2014. 3, 5, 6

[9] P Campbell. Bmw sets 2021 goal for driverless cars. *Financial Times*, July 2016. 3, 5, 6

[10] International Organization for Standardization. *ISO 26262. Road Vehicles - Funtional Safety*, 2018. 4, 9, 10, 28

[11] RTCA and EUROCAE. *DO-178C / ED-12C, Software Considerations in Airborne Systems and Equipment Certification*, 2011. 4

[12] European Cooperation for Space Standardization (ECSS). *ECSS-Q-ST-80C, Development and Maintenance of SW Standard for Space*, 2017. 4

[13] European Cooperation for Space Standardization (ECSS). *ECSS-E-ST-40C, SW engineering Standard for Space*, 2009. 4

[14] *High-performance embedded systems for the aerospace industry.* https://innovacion.upv.es/en/casos-de-colaboracion/sistemas-embebidos-de-altas-prestaciones-para-la-industria-aeroespacial/. 5

[15] Robert N. Charette. *How software is eating the car*, June 2021. https://spectrum.ieee.org/software-eating-car. 5

[16] M. Harris. *Mentor Graphics Moves Into Automated Driving*, April 2017. https://spectrum.ieee.org/cars-that-think/computing/embedded-systems/mentor-graphics-moves-into-automated-driving. 5

[17] Robert N. Charette. *This Car Runs On code*, February 2009. https://spectrum.ieee.org/this-car-runs-on-code. 5

[18] ARM. ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade, 2015. https://www.arm.com/company/news/2015/04/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade. 5

[19] T. Ungerer, F. J. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, J. Wolf, H. Cassé, I. Guliashvili S. Uhrig, M. Houston, F. Kluge, S. Metzlaff, and J. Mische. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro*, 30(5):66–75, 2010. 5, 8, 24, 25, 28

[20] Tilera. *TILE-Gx Processors Family*, 2013. https://caxapa.ru/thumbs/281914/TILE-Gx_Processor_PB025_v4_0.pdf. 5, 8, 21, 22, 41, 51, 59, 63

[21] M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla. Assessing the suitability of the ngmp multi-core processor in the space domain. In *EMSOFT*, 2012. 5, 19, 26, 28, 29, 135

[22] H. Yun P. K. Valsan and F. Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *RTAS*, 2016. 5, 7, 9, 19

[23] *AMBA: The Standard for On-Chip Communication.* https://developer.arm.com/documentation/ihi0011/a/Introduction-to-the-AMBA-Buses/Overview-of-the-AMBA-specification. 5

[24] ARM developer. *ARM CoreLink CCN-508 Cache Coherent Network Reference Manual*, August 2017. 5

[25] Arm Ltd. *Arm CoreLink CMN-600 Coherent Mesh Network – Technical Reference Manual*, 2018. https://developer.arm.com/documentation/100180/0302/?lang=en. 6, 34, 118, 121

[26] Xilinx. *System-Level Benefits of the Versal Platform*, 2022. https://www.xilinx.com/content/dam/xilinx/support/documentation/white_papers/wp539-versal-system-level-benefits.pdf. 6, 83, 84

[27] Tom's Hardware. SiPearl's 72-Core Rhea HPC SoC to Be Made Using TSMC's N6 Node, 2021. https://www.tomshardware.com/news/sipearl-rhea-n6-open-silicon-research. 6

[28] European Processor Initiative. Shooting for the moon. taking european technology to the next level. *HiPEAC info magazine*, 65(65):12–15, January 2022. 6

[29] EPI Consortium. European Processor Initiative. https://www.european-processor-initiative.eu/. 6, 112, 143

[30] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008. 6, 7, 18

[31] R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun. Wcet(m) estimation in multi-core systems using single core equivalence. In *ECRTS*, 2015. 6, 25, 27, 29

[32] J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. J. Cazorla. Bounding Resource Contention Interference in the Next-Generation Microprocessor (NGMP). In *ERTS*, 2015. 6, 8, 19, 25, 26, 28, 29

[33] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. In *ECRTS*, 2014. 6, 10, 25, 26, 27, 28, 29, 67, 127

[34] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for wcet analysis of hard real-time multicore systems. In *ISCA*, 2009. 6, 8, 29

[35] A. Bender. MILP based task mapping for heterogeneous multiprocessor systems. In *Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European*, pages 190–197, Sep 1996. 6, 26

[36] F. Siyoum, B. Akesson, S. Stuijk, K. Goossens, and H. Corporaal. Resource-efficient real-time scheduling using credit-controlled static-priority arbitration. In *RTCSA*, 2011. 6

[37] S. Chattopadhyay, C. Lee Kee, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk. A unified wcet analysis framework for multi-core platforms. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 99–108, 2012. 7, 18

[38] P. Cousot and R. Cousot. Basic concepts of abstract interpretation. In Renè Jacquart, editor, *Building the Information Society*, pages 359–366, Boston, MA, 2004. Springer US. 7, 18

[39] J. Abella, C. Hernández, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-askasua, J. Perez, E. Mezzetti, and T. Vardanega. Wcet analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, 2015. 7, 8, 10, 18

[40] E. Mezzetti and T. Vardanega. On the industrial fitness of wcet analysis. 2011. 7, 18

[41] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):966–978, 2009. 7

[42] S. Bünte, M. Zolda, M. Tautschnig, and R. Kirner. Improving the confidence in measurement-based timing analysis. In *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 144–151, 2011. 7

[43] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 91–101, 2012. 7, 18, 19

[44] J. Abella, M. Padilla, J. D. Castillo, and F. J. Cazorla. Measurement-based worst-case execution time estimation using the coefficient of variation. *ACM Trans. Des. Autom. Electron. Syst.*, 22(4), jun 2017. 7, 18, 19

[45] J. Pérez-Cerrolaza, R. Obermaisser, J. Abella, F.o J. Cazorla, K. Grüttner, I. Agirre, H. Ahmadian, and I. Allende. Multi-core devices for safety-critical systems: A survey. *ACM Comput. Surv.*, 53(4):79:1–79:38, 2020. 7

[46] R. Wilhelm. Mixed feelings about mixed criticality (invited paper). In *WCET*, volume 63 of *OASIcs*, pages 1:1–1:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 7

[47] S. Hahn, J. Reineke, and R. Wilhelm. Towards compositionality in execution time analysis: Definition and challenges. *SIGBED Rev.*, 12(1):28–36, mar 2015. 8, 24, 26

[48] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken. Compsoc: A template for composable and predictable multi-processor system on chips. *ACM Trans. Design Autom. Electr. Syst.*, 14:2:1–2:24, 2009. 8, 25, 28

[49] M. Schoeberl and A. Rocha. T-crest: A time-predictable multi-core platform for aerospace applications. In *DASIA*, 2014. 8, 9, 25, 28

[50] Cobham Gaisler. *Quad Core LEON4 SPARC V8 Processor*, 2011. `https://www.gaisler.com/index.php/products/components/gr740`. 8, 101, 128, 134, 135

[51] M. Panic, C. Hernández, E. Quiñones, J. Abella, and F. J. Cazorla. Modeling high-performance wormhole NoCs for critical real-time embedded systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, April 2016. 8, 22, 23, 24, 25, 26, 27, 41, 48, 51, 59, 61, 63, 85, 102, 119, 120, 121

[52] J. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001. 9, 10, 27, 83

[53] F. Jafari, Z. Lu, A. Jantsch, and M. H. Yaghmaee. Buffer optimization in network-on-chip through flow regulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(12):1973–1986, 2010. 9

[54] B. Nikolić, S. Tobuschat, L. Soares Indrusiak, R. Ernst, and A. Burns. Real-time analysis of priority-preemptive nocs with arbitrary buffer sizes and router delays. *Real-Time Syst.*, 55(1):63–105, January 2019. 9, 26

[55] F. Giroudot and A. Mifdaoui. Buffer-aware worst-case timing analysis of wormhole nocs using network calculus. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 37–48, 2018. 9, 22, 25, 27, 83, 90

[56] F. Giroudot and A. Mifdaoui. Graph-based approach for buffer-aware timing analysis of heterogeneous wormhole nocs under bursty traffic. *IEEE Access*, 8:32442–32463, 2020. 9, 27

[57] M. Shekhar, H. Ramaprasad, and F. Mueller. Network-on-chip aware scheduling of hard-real-time tasks. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, pages 141–150, June 2014. 9, 59

[58] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip: concepts, architectures, and implementations. *IEEE Design Test of Computers*, 22(5):414–421, Sept 2005. 9, 25, 28

[59] T. Picornell, J. Flich, C. Hernández, and J. Duato. Dcfnoc: A delayed conflict-free time division multiplexing network on chip. In *DAC*, June 2019. 9

[60] F. Giroudot and A. Mifdaoui. Tightness and computation assessment of worst-case delay bounds in wormhole networks-on-chip. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, RTNS '19, page 19–29, New York, NY, USA, 2019. Association for Computing Machinery. 9, 22, 27, 83

[61] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, pages 161–170, April 2008. 9, 10, 25, 26, 27, 83, 85, 90

[62] D. Rahmati, S. Murali, L. Benini, F. Angiolini, G. De Micheli, and H. Sarbazi-Azad. Computing accurate performance bounds for best effort networks-on-chip. *IEEE Transactions on Computers*, 62(3):452–467, March 2013. 9, 22, 24, 25, 27, 41, 59, 63

[63] Y. Xiao, S. Nazarian, and P. Bogdan. Self-optimizing and self-programming computing systems: A combined compiler, complex networks, and machine learning approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(6):1416–1427, 2019. 9

[64] M. Panić, C. Hernández, J. Abella, A. Roca, E. Quiñones, and F. J. Cazorla. Improving performance guarantees in wormhole mesh NoC designs. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1485–1488, March 2016. 9, 11, 23, 26, 41, 47, 59, 65, 75, 76, 100

[65] European Union Aviation Safety Agency (EASA). *AMC 20-193 Use of multi-core processors*, 2022. `https://www.easa.europa.eu/en/downloads/134960/en`. 9, 28

[66] EASA, FAE. AMC 20-193 Use of multi-core processors, Annex I to ED Decision 2022/001/R. Technical report, European Union Aviation Safety Agency, 2022. 9

[67] H. Yun, R. Yao, G. Pellizzoni, M. Caccamo, and L. Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *RTAS*, 2013. 9, 10, 27, 29, 127

[68] A. Agrawal, G. Fohler, J. Freitag, J. Nowotsch, S. Uhrig, and M. Paulitsch. Contention-Aware Dynamic Memory Bandwidth Isolation with Predictability in COTS Multicores: An Avionics Case Study. In *ECRTS*, 2017. 9, 25, 27, 29

[69] E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla. High-integrity performance monitoring units in automotive chips for reliable timing v&v. *IEEE Micro*, 38(1):56–65, 2018. 10, 84

[70] A. Burns, J. Harbin, and L.S. Indrusiak. A wormhole NoC protocol for mixed criticality systems. In *2014 IEEE Real-Time Systems Symposium*, pages 184–195, Dec 2014. 10, 11, 25

[71] L. S. Indrusiak, J. Harbin, and A. Burns. Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 47–56, July 2015. 10, 11, 25

[72] S. Kotz and S. Nadarajah. *Extreme value distributions: theory and applications.* World Scientific, 2000. 19

[73] G. Fernandez, J. Jalle, J. Abella, E. Quiñones, T. Vardanega, and F. J. Cazorla. *Computing Safe Contention Bounds for Multicore Resources with Round-Robin and FIFO Arbitration*, 2017. 19

[74] J. Jalle, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Bus designs for time-probabilistic multicore processors. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2014. 19

[75] J. Jalle, J. Abella, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla. Ahrb: A high-performance time-composable amba ahb bus. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium, 2014*, 2014. 19

[76] J. Jalle, M. Fernández, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. J. Cazorla. Contention-aware performance monitoring counter support for real-time mpsocs. In *11th IEEE Symposium on Industrial Embedded Systems, 2016*, 2016. 19, 115

[77] M. Slijepcevic, C. Hernández, J. Abella, and F. J. Cazorla. Design and implementation of a fair credit-based bandwidth sharing scheme for buses. In *DATE*, 2017. 19

[78] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In *IEEE TrustCom*, 2011. 19, 25, 26, 28, 29, 67

[79] G. Ravindran and M. Stumm. A performance comparison of hierarchical ring- and mesh- connected multiprocessor networks. *HPCA*, 1997. 19

[80] M. Slijepcevic, M. Fernadez, C. Hernández, J. Abella, E. Quiñones, and F. J. Cazorla. ptnoc: Probabilistically time-analyzable tree-based noc for mixed-criticality systems. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 404–412, Aug 2016. 19

[81] E. Salminen, T. Kangas, V. Lahtinen, J. Riihimäki, K. Kuusilinna, and Timo D. Hämäläinen. Benchmarking mesh and hierarchical bus networks in system-on-chip context. *Journal of Systems Architecture*, 2007. 19

[82] J. Flich and D. Bertozzi, editors. *Designing network on-chip architectures in the nanoscale era*. Chapman & Hall/CRC computational science series. Chapman and Hall/CRC, 2011. 19

[83] S. Tobuschat and R. Ernst. Real-time communication analysis for networks-on-chip with backpressure. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 590–595, March 2017. 22, 26, 27

[84] J. Jalle, J. Abella, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla. Deconstructing bus access control policies for real-time multicores. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 31–38, June 2013. 23, 24

[85] H. Park and K. Choi. Position-based weighted round-robin arbitration for equality of service in many-core network-on-chips. In *NoCArc'12: Proceedings of the 5th International Workwhop on the NoC Architectures*, NoCArc, 2012. 23, 24, 48, 49, 65

[86] GENESYS. GENeric Embedded SYStem Platform, 2010. 24

[87] C. Hernández, J. Abella, F. J. Cazorla, A. Bardizbanyan, J. Andersson, F. Cross, and F. Wartel. Design and implementation of a time predictable processor: Evaluation with a space case study. In *ECRTS*, 2017. 25, 28

[88] C. Shaver M. Zimmer, D. Broman and E. A. Lee. Flexpret: A processor platform for mixed-criticality systems. In *RTAS*, pages 101–110, 2014. 25, 28

[89] E. Díaz, M. Fernández, L. Kosmidis, and E. Mezzetti. MC2: Multicore and cache analysis via deterministic and probabilistic jitter bounding. In *Ada-Europe*, 2017. 25, 26, 29

[90] Y. Qian, Z. Lu, and W. Dou. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In *IEEE/ACM NoCS*, 2009. 25, 27

[91] S. Lee. Real-time wormhole channels. *Journal Of Parallel And Distributed Computing*, 63:299–311, 2003. 25, 27

[92] T. Ferrandiz, F. Frances, and C. Fraboul. A sensitivity analysis of two worst-case delay computation methods for spacewire networks. In *ECRTS*, 2012. 25

[93] D. Dasari, B. Nikoli'c, V. N'elis, and S. M. Petters. Noc contention analysis using a branch-and-prune algorithm. *ACM Trans. Embed. Comput. Syst.*, 13(3s), March 2014. 25, 27

[94] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The nostrum backbone-a communication protocol stack for networks on chip. In *17th International Conference on VLSI Design. Proceedings.*, pages 693–696, 2004. 25

[95] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 152–160, May 2012. 25

[96] R. Obermaisser, C. E. Salloum, B. Huber, and H. Kopetz. The time-triggered system-on-a-chip architecture. In *2008 IEEE International Symposium on Industrial Electronics*, pages 1941–1947, June 2008. 25

[97] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood. Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip. *SIGARCH Comput. Archit. News*, 41(3):583–594, June 2013. 25

[98] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos. PhaseNoC: TDM scheduling at the virtual-channel level for efficient network traffic isolation. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1090–1095, March 2015. 25

[99] M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch. Mixed-criticality embedded systems - A balance ensuring partitioning and performance. In *2015 Euromicro Conference on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28, 2015*, pages 453–461, 2015. 25, 59

[100] J. Sparsø. Design of networks-on-chip for real-time multi-processor systems-on-chip. In *2012 12th International Conference on Application of Concurrency to System Design*, pages 1–5, June 2012. 25, 28

[101] H. Kashif and H. Patel. Buffer space allocation for real-time priority-aware networks. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, April 2016. 25

[102] B. Nikolić, H. I. Ali, S. M. Petters, and L. M. Pinho. Are virtual channels the bottleneck of priority-aware wormhole-switched noc-based many-cores? In *21st International Conference on Real-Time Networks and Systems, RTNS 2013, Sophia Antipolis, France, October 17-18, 2013*, pages 13–22, 2013. 25

[103] Z. Shi and A. Burns. Real-time communication analysis with a priority share policy in on-chip networks. pages 3–12, July 2009. 25, 26

[104] Q. Xiong, Z. Lu, F. Wu, and C. Xie. Real-time analysis for wormhole NoC: Revisited and revised. In *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, pages 75–80, May 2016. 25

[105] L. S. Indrusiak, A. Burns, and B. Nikolić. Buffer-aware bounds to multi-point progressive blocking in priority-preemptive nocs. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 219–224, March 2018. 25

[106] E. A. Rambo and R. Ernst. Worst-case communication time analysis of networks-on-chip with shared virtual channels. In *Design, Automation and Test in Europe 2015*, DATE, 2015. 26, 27

[107] T. Kranich and M. Berekovic. Noc switch with credit based guaranteed service support qualified for GALS systems. In *DSD*, 2010. 26

[108] D. Crupnicoff, S. Das, and E. Zahavi. Deploying quality of service and congestion control in infiniband-based data center networks. *White paper, Mellanox Technologies*, 2005. 26, 41

[109] P. Kumar Sahu and S. Chattopadhyay. A survey on application mapping strategies for network-on-chip design. *Journal of Systems Architecture*, 59(1):60 – 76, 2013. 26

[110] C. Chou and R. Marculescu. Contention-aware application mapping for network-on-chip communication architectures. In *2008 IEEE International Conference on Computer Design*, pages 164–169, Oct 2008. 26, 59

[111] H. Yu, Y. Ha, and B. Veeravalli. Communication-aware application mapping and scheduling for NoC-based MPSoCs. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 3232–3235, May 2010. 26, 59

[112] C. Zimmer and F. Mueller. Low contention mapping of real-time tasks onto TilePro 64 core processors. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 131–140, April 2012. 26, 59

[113] U. Y. Ogras, P. Bogdan, and R. Marculescu. An analytical approach for network-on-chip performance analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(12):2001–2013, 2010. 26

[114] T. Moseley, J.L. Kihm, D.A. Connors, and D. Grunwald. Methods for modeling resource contention on simultaneous multithreading processors. In *IEEE ICCD*, 2005. 26

[115] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *EDCC*. IEEE Computer Society, 2012. 26, 28, 29, 128, 135

[116] R. Pellizzoni, A. Schranzhofer, Jian-Jia Chen, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *DATE*, 2010. 26, 29

[117] J. Cardona, C. Hernández, J. Abella, and F. J. Cazorla. Maximum-contention control unit (mccu): Resource access count and contention time enforcement. *DATE*, pages 710–715, 2019. 27

[118] A. E. Kiasari, Z. Lu, and A. Jantsch. An analytical latency model for networks-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):113–123, 2013. 27

[119] X. Xiang, S. Ghose, O. Mutlu, and N. Tzeng. A model for application slowdown estimation in on-chip networks and its use for improving system fairness and performance. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 456–463, Oct 2016. 27

[120] Q. Xiong, F. Wu, Z. Lu, and C. Xie. Extending real-time analysis for wormhole nocs. *IEEE Transactions on Computers*, 66(9):1532–1546, 2017. 27

[121] M. Liu, M. Becker, M. Behnam, and T. Nolte. A tighter recursive calculus to compute the worst case traversal time of real-time traffic over nocs. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 275–282, 2017. 27

[122] A. Kostrzewa, S. Saidi, L. Ecco, and R. Ernst. Flexible tdm-based resource management in on-chip networks. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, page 151–160, New York, NY, USA, 2015. Association for Computing Machinery. 28

[123] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13. IEEE Press, 2013. 28

[124] *LEON 4 Processor*, June 2008. `https://www.gaisler.com/index.php/products/processors/leon4`. 32

[125] *Proxima project*, 2016. `http://proxima-project.eu/`. 32

[126] SoCLib. -, 2003-2012. http://www.soclib.fr/trac/dev. 34, 51, 74, 97, 101, 135

[127] *NaNoC Project*, 2010-2012. `https://sites.google.com/site/nanocproject/`. 34, 51, 63, 74, 121

[128] J. Jalle, J. Abella, L. Fossati, M. Zulianello, and F. J. Cazorla. Validating a timing simulator for the NGMP multicore processor. In *DASIA*, 2016. 34

[129] *Cluster Investigació UPC SERT*, 2020. `https://www.ac.upc.edu/ca/nosaltres/serveis-tic/altas-prestaciones`. 36

[130] Chunho Lee et al. Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, pages 330–335, Dec 1997. 36, 135

[131] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007. 36, 37

[132] M. Panić, E. Quiñones, P. G. Zaykov, C. Hernández, J. Abella, and F. J. Cazorla. Parallel many-core avionics systems. In *2014 International Conference on Embedded Software (EMSOFT)*, pages 1–10, Oct 2014. 42, 59

[133] S. Royuela, X. Martorell, E. Quiñones, and L. M. Pinho. OpenMP tasking model for ada: Safety and correctness. In Johann Blieberger and Markus Bader, editors, *Reliable Software Technologies – Ada-Europe 2017*, pages 184–200, Cham, 2017. Springer International Publishing. 52, 75

[134] Lei Yang, Weichen Liu, Peng Chen, Nan Guan, and Mengquan Li. Task mapping on SMART NoC: Contention matters, not the distance. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017. 59

[135] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, and T. Skeie. Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pages 10 pp.–, April 2006. 64, 69

[136] M. Lodde and J. Flich. An NoC and cache hierarchy substrate to address effective virtualization and fault-tolerance. In *2013 Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS), Tempe, AZ, USA, April 21-24, 2013*, pages 1–8, 2013. 74

[137] Infineon. AURIX Multicore 32-bit Microcontrollers for automotive and industrial applications, 2022. `https://www.infineon.com/dgdl/Infineon-TriCore_Family_BR-ProductBrochure-v01_00-EN.pdf?fileId=5546d4625d5945ed015dc81f47b436c7`. 83

[138] XILINX. Rockwell Collins Uses Zynq UltraScale+ RFSoC Devices in Revolutionizing How Arrays are Produced and Fielded: Powered by Xilinx. 2018. `https://www.xilinx.com/video/corporate/rockwell-collins-rfsoc-revolutionizing-how-arrays-are-produced.html`. 83

[139] A. Serrano-Cases, J. M. Reina, J. Abella, E. Mezzetti, and F. J. Cazorla. Leveraging hardware qos to control contention in the xilinx zynq ultrascale+ mpsoc. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems, ECRTS 2021, July 5-9, 2021, Virtual Conference*, volume 196 of *LIPIcs*, pages 3:1–3:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. 86

[140] F. Gilabert, M. E. Gómez, S. Medardoni, and D. Bertozzi. Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip. In *2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, pages 165–172, 2010. 96

[141] J. Cardona, C. Hernández, J. Abella, and Francisco J. Cazorla. Eomesh: Combined flow balancing and deterministic routing for reduced wcet estimates in embedded real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2451–2461, 2018. 100

[142] J. Duato, S. Yalamanchili, and N. Lionel. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. 101

[143] ARINC Inc. *ARINC Specification 653: Avionics Application Software Standard Standard Interface, Part 1 and 4, Subset Services*, June 2012. 130

[144] ARM Ltd. *ARM Cortex-A9 Technical Reference Manual r4p1*, 2016. 134

[145] G. Cabo, F. Bas, R. Lorenzo, D. Trilla, S. Alcaide, M. Moretó, C. Hernández, and J. Abella. Safesu: an extended statistics unit for multicore timing interference. In *2021 IEEE European Test Symposium (ETS)*, pages 1–4, 2021. 144

[146] G. Cabo, S. Alcaide, C. Hernández, P. Benedicte, F. Bas, F. Mazzocchetti, and J. Abella. Safesu-2: a safe statistics unit for space mpsocs. In *2022 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1085–1086, 2022. 144

[147] S. Alcaide, G. Cabo, F. Bas, P. Benedicte, F. Fuentes, F. Chang, I. Lasfar, R. Canal, and J. Abella. Safex: Open source hardware and software components for safety-critical systems. In *FDL 2022 17th Forum on specification and Design Languages, Linz (Austria), September*, 2022. 144

[148] N. Wessman, F. Malatesta, S. Ribes, J. Andersson, A. García-Vilanova, M. Masmano, V. Nicolau, P. Gomez, J. L. Rhun, S. Alcaide, G. Cabo, F. Bas, P. Benedicte, F. Mazzocchetti, and J. Abella. De-risc: A complete risc-v based space-grade platform. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 802–807, 2022. 144

[149] C. Hernández, J. Flich, R. Paredes, C. Lefebvre, I. Allende, J. Abella, D. Trillin, M. Matschnig, B. Fischer, K. Schwarz, J. Kiszka, M. Rönnbäck, J. Klockars, N. McGuire, F. Rammerstorfer, C. Schwarzl, F. Wartet, D. Lüdemann, and M. Labayen. Selene: Self-monitored dependable platform for high-performance safety-critical systems. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 370–377, 2020. 144

[150] *SELENE project*, 2022. `https://cordis.europa.eu/project/id/871467`. 144