Accurately Measuring Contention in Mesh NoCs in Time-Sensitive Embedded Systems

JORDI CARDONA, Universitat Politècnica de Catalunya and Barcelona Supercomputing Center, Spain
CARLES HERNÁNDEZ, Universitat Politècnica de València and Barcelona Supercomputing Center, Spain
JAUME ABELLA, Barcelona Supercomputing Center, Spain
ENRICO MEZZETTI, Barcelona Supercomputing Center, Spain
FRANCISCO J. CAZORLA, Barcelona Supercomputing Center, Spain

The computing capacity demanded by embedded systems is on the rise as software implements more functionalities, ranging from best-effort entertainment functions to performance-guaranteed safety-related functions. Heterogeneous manycore processors, using wormhole mesh (wmesh) Network-on-Chips (NoCs) as the main communication means, and contention block among applications, are increasingly considered to deliver the required computing performance. Most research efforts on software timing analysis have focused on deriving bounds (estimates) to the contention that tasks can suffer when accessing wmesh NoCs. However, less effort has been devoted to an equally important problem, namely, *accurately* measuring the actual contention tasks generate each other on the wmesh which is instrumental during system validation to diagnose any software timing misbehavior and determine which tasks are particularly affected by contention on specific wmesh routers. In this paper, we work on the foundations of *contention measuring* in wmesh NoCs and propose and explain the rationale of a *golden metric*, called task *PairWise Contention* (PWC). PWC allows ascribing the actual share of the contention a given task suffers in the wmesh to each of its co-runner tasks at packet level. We also introduce and formalize a *Golden Reference Value* (GRV) for PWC that specifically defines a criterion to fairly break down the contention suffered by a task among its co-runner tasks in the wmesh. Our evaluation shows that GRV effectively captures how contention occurs by identifying the actual core (task) causing contention and whether contention is caused by local or remote interference in the wmesh.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

Additional Key Words and Phrases: Timing Validation & Verification, Contention, Contention breakdown, NoCs

ACM Reference Format:
Jordi Cardona, Carles Hernández, Jaume Abella, Enrico Mezzetti, and Francisco J. Cazorla. 2023. Accurately Measuring Contention in Mesh NoCs, in Time-Sensitive Embedded Systems. 1, 1 (January 2023), 35 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The trend towards more autonomous software-centric functionalities is on the rise in time-sensitive embedded systems (TSES) in relevant industrial domains such as automotive. This includes automotive artificial-intelligence-based driving assistance systems functionalities, like lane keeping and obstacle detection, and autonomous driving (AD) solutions.

Authors' addresses: Jordi Cardona, jordi.cardona@bsc.es, Universitat Politècnica de Catalunya and Barcelona Supercomputing Center, Barcelona, Spain; Carles Hernández, carles.hernandez@bsc.es, Universitat Politècnica de València and Barcelona Supercomputing Center, Valencia, Spain; Jaume Abella, jaume.abella@bsc.es, Barcelona Supercomputing Center, Barcelona, Spain; Enrico Mezzetti, enrico.mezzetti@bsc.es, Barcelona Supercomputing Center, Barcelona, Spain; Francisco J. Cazorla, francisco.cazorla@bsc.es, Barcelona Supercomputing Center, Barcelona, Spain.

As a result, embedded products comprise more and more complex software components with different timing needs and unprecedented computing performance requirements [2, 3]. On the hardware side, major TSES industry players have adopted (or are on the way to doing so) small multicores, such as the Infineon AURIX family [29] in automotive and the Xilinx Zynq UltraScale+ [64] in avionics, which include few cores (e.g. 3 to 6).

Commercial Off-The-Shelf (COTS) manycores, with larger core counts than already adopted multicores, are of particular interest given their high performance and availability. Yet their timing characteristics may challenge the verification and validation of the timing constraints of critical real-time applications. 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 TSES already deploy NoCs (e.g. to connect 10 to 20 nodes), such as the Kalray MPPA 256 SoC [4] (e.g. a 4x4 mesh) and the Xilinx VERSAL Multi-Processor System-on-Chip (MPSoC) [5].

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. While some existing proposals advocate for hardware/software support to reduce or even eliminate contention [26, 48, 53], hence not requiring to bound or track NoC contention, they are not generally embraced by high-performance and general-purpose manycore due to the difficulties to control NoC traffic, especially in the presence of distributed and coherent cache memories. Other works [9, 30, 50, 62, 63] focus on COTS solutions from the high-performance market, which include minimum hardware support for time predictability, in an attempt to contain non-recurring costs of real-time embedded products [52, 58]. The mechanisms proposed in these works build on generic NoC designs, e.g. wormhole mesh NoCs, and target-specific NoC configurations under which derived contention bounds are tight, e.g. deterministic routing (like XY for mesh networks) with minimum size packets. In this line, we target wormhole mesh (wmesh) NoCs as they are widely implemented in COTS manycores [4, 52, 58] for their high performance and limited implementation costs.

In terms of timing analysis of COTS NoCs components, several works specifically focus on deriving timing bounds for verification purposes. However, fewer efforts address software timing validation methods that follow a requirement-based dynamic testing approach. In this line, safety standards and the associated support documents call for controlling contention on a per-shared resource basis. This is carried out by bounding contention and setting appropriate safety measures in case of overruns, thus in line with the freedom from interference requirement in ISO 26262 in automotive [20], and interference channels mitigation in CAST-32A (now AM(C) 20-193) [16] for avionics. During the validation phase, testing campaigns are carried out in the different integration steps to collect evidence supporting that those assumptions and conclusions reached during the design phase effectively hold. The absence of timing violations during the tests is considered a key element of the evidence on the system's timing correctness [18].

Works deriving bounds to the worst-case contention in NoCs [23, 24, 35] classify contention into direct or indirect depending on whether contending flows share resources over their paths in the NoC or not [55]. 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 TSES 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 [39], must be detected during testing.
- *Correction.* It allows determining the point (nodes) of the mesh where the contention occurs and the tasks' contribution to this contention. This is fundamental to propose and apply corrective measures in case of detected misbehavior (see sections 2.2 and 7). Examples of corrective actions that can be taken include the enforcement of a task mapping where delayed and delaying tasks do not share links/routers or where the offended (offending) task is placed much closer to (farther from) the target node to reduce the contention impact. The same information can also be used to introduce or readjust traffic limitations and priorities, or to recompute offline tasks contention budgets.

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 along the following three axes:

(1) We define a golden metric called *PairWise Contention* (PWC) that captures the slowdown the packets generated by the Task under Analysis (TuA) suffer when accessing the wmesh due to packets from the 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.
(2) 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.
(3) 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 TSES [4, 5].

GRV is instrumental in the development of PWC metrics tailored to specific COTS multicores, hence building only on observation knobs (e.g. performance monitoring counters in the NoC routers) available on the board. The characteristics of the COTS multicore under study can be modeled (e.g. in a timing simulator) which will allow comparing the tailored PWC implementation against GRV with the aim of tuning such PWC implementation until it is close enough to GRV, while adequate for the particular COTS multicore under study.

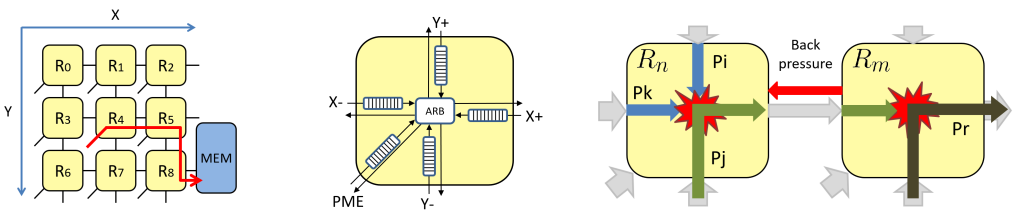
The rest of this paper is structured as follows: Section 2 introduces the relevant background. Section 3 defines PairWise Contention (PWC). Section 4 defines our Gold Reference Value (GRV). Section 5 proposes an off-line method to derive GRV and explains how PWC/GRV applies to other wmesh setups and NoCs. Section 6 assesses how GRV accounts for PWC and provides reliable contention information in wmesh mesh-based systems. Section 7 presents the most relevant related works. Finally, Section 8 summarizes the main conclusions of this work.

2 BACKGROUND

2.1 Wormhole NoCs fundamentals

We target NxM wormhole mesh NoCs, see Figure 1a and Figure 1b, which are widely implemented in COTS manycores [4, 52, 58] as they provide high performance. The main NoCs characteristics are summarized below.

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



(a) 3x3 2Dmesh using XY routing (b) Router input/output ports (c) Backpressure example

Fig. 1. Mesh concepts

Routing. 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 1a, 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 paper to denote the *ordered* set of routers traversed by F_i packets. Deterministic routing policies like XY or YX routing are the preferred policies to allow the Worst Case Delay (WCD) estimation [46, 50] in TSES. 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).

[1]A flit – short of flow control unit – is the atomic element in which packets are split into be transmitted. Only the first flit of a packet is arbitrated, the others follow the head flit as long as buffer space is available in the next router or port.

Switching and control flow. In wormhole switching messages or packets are split into several flits: the header flit that contains the destination information, the body flits that contain the data, and the tail flit that can contain error detection codes information. When the header flit of packet P_i arrives at the 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, traverses the router crossbar. Flits of a packet leave the router when the signals of the control flow mechanism from the next router inform 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 the 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. Wormhole switching is one of the most common approaches used in multicore processors [52, 58]. Indeed, recent works build upon wormhole switching and study contention effects instance buffer backpreassure [59] and how to take them into account in the late stages [23, 24].

Arbitration. Policies like round-robin are locally fair, which favors time predictability, and are easy to implement [31]. Round-robin policy fairly grants access to the input ports contending for the same output port that has a ready packet to be sent at the moment of taking the decision. When the header flit is stalled due to arbitration (contention), body flits are also stalled. Also, note that input port queues are typically sized with enough space to avoid bubbles in the packet transmission.

Design choices. The only requirement of our PWC and GRV is the use of deterministic routing setups that anyway are needed to enable the derivation of tight worst-case traversal times and avoid deadlocks. Our proposed PWC and GRV can handle both highly predictable and less predictable setups as discussed in Section 5.2. Furthermore, we cover virtual channel (VC) extension in PWC and GRV in Section 4.3 and multiple packet size scenarios in Section 6.2. As explained there, those scenarios require no change to PWC and GRV, so the adoption of these features is a matter of whether they can be modeled during the timing analysis (budgeting) phase rather than whether they can be measured during the timing validation with PWC/GRV.

2.2 Software Timing

Timing-related aspects of multicore shared resources in TSES are covered in different stages of the overall software engineering process, namely, verification, validation and operation (see Table 1).

Table 1. Scopes of software timing verification, validation and safety measures.

Stage	Early Stages	Late Stages	Operation
Name	Usually referred to as verification or budgeting	Usually referred to as validation or testing	Safety measures or enforcement
Goal	Feasible schedule maximizing utilization. Build a "correct" system	Gather evidence of schedule being respected. Detect incorrect behavior	Preserve safety. Guarantee safety despite incorrect behavior
Means	Estimate	Track and Report	Track and React
Target	End-to-end timing for tasks	Detailed timing information for each task for diagnosis purposes	End-to-end timing for tasks
Result	Tight upper bounds	Actual contention	Safe operation
Research	Abundant scientific works	Received little attention. ***The target of this work***	Some scientific works

Verification (budgeting) For 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 [7, 27, 28, 53, 60, 67] and software techniques that control the number of requests each task generates as a way to control the contention they can cause on others [6, 11, 14, 32, 38, 43]. In the case of wmesh NoCs, budgeting solutions result in a packet-level analysis of contention to accurately bound the maximum contention experienced per packet and/or for the task as a whole.

Current main techniques used to bound contention in wNoCs can be categorized as follows:

- Scheduling Theory techniques [65] consider tasks contention managing flows priorities, multiple VCs and flit-level preemption so as to reduce contention suffered by high-priority flows.
- Network Calculus [23, 35, 49] techniques derive contention bounds by using mathematical and statistical tools and assuming per flow packets arrival and departure distribution curves.
- Recursive Calculus techniques [12, 37] use branch and prune or branch, prune and collapse algorithms to compute end-to-end packets latency [36].
- Compositional Timing Analysis techniques [51, 59] extend Network Calculus to obtain the worst-case contention delay.
- Statical Timing Analysis Techniques mathematically compute WCD and WCET by systematically considering the worst-possible contention case [46, 50].

Some of those solutions break down contention among direct and indirect categories [23, 24, 65], 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 3.

Validation (testing) mechanisms are used during late design stages to provide evidence on the correctness of the selected time budgets (i.e. WCET or contention estimates). Validation 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. Focusing on total observed contention effects only is generally considered inadequate as observations can hide undesired contention behavior, decreasing the confidence that can be put on testing. In this work, we focus on improving the effectiveness and informative value of validation testing.

Operation (enforcement). Safety measures are deployed to prevent timing violations during system operation by monitoring the use of resources for 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 [20]). In the latter case, the number of accesses to shared hardware resources (or any other usage measure) is monitored [7, 43] to take corrective actions when specific usage thresholds are exceeded.

3 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 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 2 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 [46, 55]. In this case, $Tiso$ = 7000 cycles and WCD vary 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 2 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.

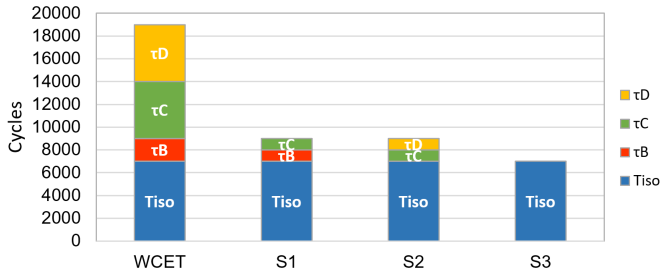


Fig. 2. Comparison between worst-case and observed contention CTs generate on τ_A under scenarios S1, S2 and S3 as introduced early in this Section.

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 [54]. 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.

3.1 PWC for 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 2) 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 1c 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 $cont^{R_n}_{j \triangleright i}$ and $cont^{R_n}_{j \triangleright k}$ are incremented [2].

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 1c, 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, $cont^{R_n}_{j \triangleright i}$ and $cont^{R_n}_{j \triangleright k}$ 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

[2]Note that, a flow is composed of packets, and each packet is composed of flits. However, in several discussions in this paper, 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 2. Definitions used in this paper.

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

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.

3.2 PWC for NoCs

We illustrate PWC for NoCs via the scenario depicted in Figure 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 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).

The contention time that τ_A's packets experience can be broken down following different criteria, as shown in the different bars in Figure 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 4 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.

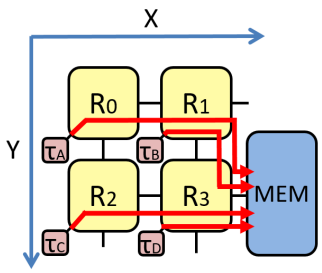


Fig. 3. 2x2 2Dmesh XY-routing setup

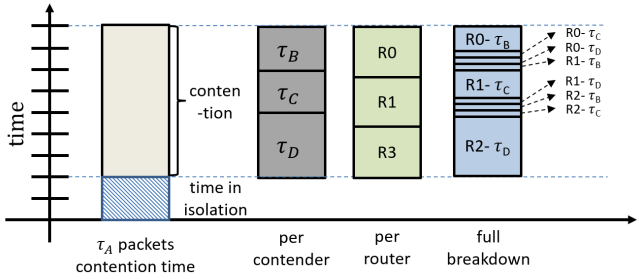


Fig. 4. Synthetic example breaking down of contention into its PWC components

- *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 [23, 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.

4 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.

(1) *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.
(2) *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 paper, 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 $P_i \in F_x$.

$$FTT_x = \sum_{P_i \in F_x} PTT_{P_i} = \sum_{P_i \in F_x} \sum_{R_n \in \widehat{H_i}} PTT_{P_i}^{R_n} \tag{1}$$

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 ($cont_{P_i}^{R_n}$).

$$PTT_{P_i}^{R_n} = zll_{P_i}^{R_n} + cont_{P_i}^{R_n} \tag{2}$$

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[3]. 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 3.2, 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.

[3]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.

- 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.

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

$$cont_{P_i}^{R_n} = lrc_{P_i}^{R_n} + rrc_{P_i}^{R_n} \tag{3}$$

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. 4) with respect to a guilty packet P_j (Eq. 5), as follows:

$$cont_{P_i}^{R_n}(t) = \sum_{P_j} cont_{j \triangleright i}^{R_n}(t) \tag{4}$$

$$cont_{j \triangleright i}^{R_n}(t) = lrc_{j \triangleright i}^{R_n}(t) + rrc_{j \triangleright i}^{R_n}(t) \tag{5}$$

where $lrc_{j \triangleright i}^{R_n}(t)$ and $rrc_{j \triangleright i}^{R_n}(t)$ represent the *PairWise Contention* unfolding of the local and remote contention terms in Eq. 3. Note that, for a given $t \in stalled(P_i, R_n)$, exactly one of the two terms in Eq. 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:

$$cont_{\tau_x} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} cont_{P_i}^{R_n}(t) \tag{6}$$

We can build on Eq. 4 and 5 to narrow the scope of Eq. 6 to model the PWC suffered from τ_x because of τ_y (per contender breakdown in Figure 4):

$$cont_{\tau_y \triangleright \tau_x} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} \sum_{P_j \in F_y} \left(lrc_{j \triangleright i}^{R_n}(t) + rrc_{j \triangleright i}^{R_n}(t) \right) \tag{7}$$

By restricting Eq. 6 and 7, we can also obtain different contention breakdowns, such as those shown in Figure 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 as follows (per router breakdown in Figure 4):

$$cont_{\tau_x} = \sum_{R_n \in \widehat{H_i}} cont_{\tau_x}^{R_n} \tag{8}$$

$$cont_{\tau_x}^{R_n} = \sum_{P_i \in F_x} \sum_{t \in stalled(P_i, R_n)} cont_{P_i}^{R_n}(t) \tag{9}$$

where $cont_{P_i}^{R_n}(t)$ is the *PairWise Contention* 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:

$$cont_{\tau_x} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} \left(lrc_{P_i}^{R_n}(t) + rrc_{P_i}^{R_n}(t) \right) \tag{10}$$

$$cont_{\tau_x}^{lrc} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} \left(lrc_{P_i}^{R_n}(t) \right) \tag{11}$$

$$cont_{\tau_x}^{rrc} = \sum_{P_i \in F_x} \sum_{\substack{R_n \in \widehat{H_i} \\ t \in stalled(P_i, R_n)}} \left(rrc_{P_i}^{R_n}(t) \right) \tag{12}$$

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. 10 in the following manner:

$$cont_{\tau_x}^{lrc,R_n} = \sum_{\substack{P_i \in \widehat{F_x} \\ t \in stalled(P_i, R_n)}} \left(lrc_{P_i}^{R_n}(t) \right) \tag{13}$$

$$cont_{\tau_x}^{rrc,R_n} = \sum_{\substack{P_i \in \widehat{F_x} \\ t \in stalled(P_i, R_n)}} \left(rrc_{P_i}^{R_n}(t) \right) \tag{14}$$

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.

Table 3. Functions used in this paper.

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 4.2.

4.1 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.

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 3.

(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, $lrc_{k \triangleright i}^{R_n}(t) = 1$.

(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 $lrc_{j \triangleright i}^{R_n}(t) = 1$. 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. $lrc_{k \triangleright i}^{R_n}(t) = 1$.

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.

4.2 Defining rrc

As for Section 4.1, 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 [4].

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, $rrc^{R_n}_{g \triangleright i}(t) = 1$.

4.3 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.

[4]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

However, in TSES with different area and energy constraints, simpler solutions [22] 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[5].

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 3. 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 4.2) 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).

5 DERIVING GRV

The use of GRV for PWC can be leveraged in two different stages of the software development cycle, as introduced in Section 2. 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).

5.1 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),

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

Algorithm 1 GRV offline implementation

```
1: procedure COMPUTE_GRV( lrc, rrc)
2:     for each time t in Time do
3:         for each router R_n in ℛ do
4:             GRV(R_n, t, lrc, rrc)
5:         end for
6:     end for
7: end procedure
```

Algorithm 2 GRV routine

```
1: procedure GRV(R_n, t, lrc, rrc)                ▷ Call for Router R_n at time t
2:     P_ready(R_n, t) ← packets ready to be arbitrated in R_n at time t
3:     for p_i ∈ P_ready(R_n, t) do
4:         < CONT, R_guilty, p_guilty > = GRVREC(p_i, R_n, t)
5:         if CONT then
6:             if R_guilty == R_n then
7:                 lrc[R_n][p_i.src][p_guilty.src] + +
8:             else
9:                 rrc[R_n][p_i.src][p_guilty.src] + +
10:            end if
11:        end if
12:    end for
13: end procedure
```

destination, identifier, the router identifier in which the packet is stored at the time of the recording. We generate traces with the gNoCsim simulator [1], as shown in Section 6. 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 1. 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*).

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 2) 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}.

Function *GRVrec* (see Algorithm 3) 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 4.1 and 4.2 respectively.

5.1.1 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 4.2 are exhaustively modeled by Algorithm 3 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.

5.1.2 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 4.2 are exhaustively modeled by Algorithm 3 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

Algorithm 3 GRV Recursive function

```
1:  function GRVrec(p, R_n, t)                    ▷ Call for packet p in Router R_n at time t
2:      p_h ← PH(p, R_n, t)
3:      p_g ← PSO(p_h, R_n, t)
4:      if p_g ≠ ∅ then                           ▷ p_g traverses the output port
5:          return <TRUE, R_n, p_g>
6:      else
7:          if target(p_h) is a router then
8:              p_next ← HN(p_h, R_n+1, t)
9:              return GRVrec(p_next, R_n+1, t)
10:         else
11:             return <FALSE, R_n,p_h>           ▷ Target is a PME
12:         end if
13:     end if
14: end function
```

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*.

5.2 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 [45]. 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 GRV/PWC 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 TSES [8]. We do not target the applicability of GRV/PWC 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 verification and validation required for TSES, as NoC WCD can become too pessimistic to be useful. Hence, the presented GRV/PWC does not directly support adaptive or non-deterministic routings which are out of the scope of this work.

Regarding other NoCs, PWC/GRC can also be applied to other types of distributed NoCs and network configurations such as the ones using virtual cut-through switching [15]. 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 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 [4]) and beyond. We use gNoCsim [1] 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.

(1) 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 injection rate, 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.
(2) Hybrid traffic: we use SoCLib [56] 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 [10], whose memory petitions traverse the NoC (implemented with gNoCsim) to reach memory. We model a tile-based manycore 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).

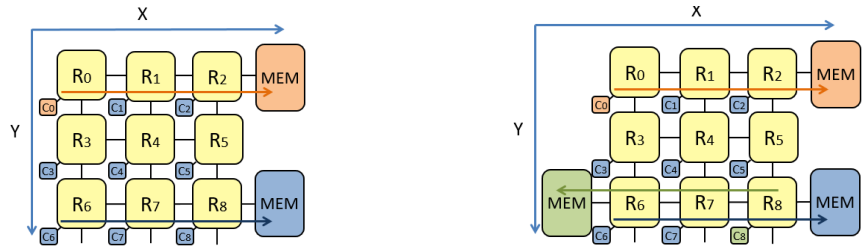
In this section, we mainly focus on packets of 1 flit size (i.e. all flits are header flits and switching and arbitration can take place every cycle), in line with recommendations in [46] to minimize maximum contention. We discuss packets with multiple sizes specifically in Section 6.2.

From gNoCsim, either standalone or integrated with SoCLib, we generate an execution (timing) trace with the information presented in Section 5.1. In order to compute the GRV of the experiments, we have implemented a C++ trace parser based on the pseudocode described in Section 5. The results reported in this section are obtained by applying GRV, as formalized in Section 4, 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.

6.1 Synthetic Traffic

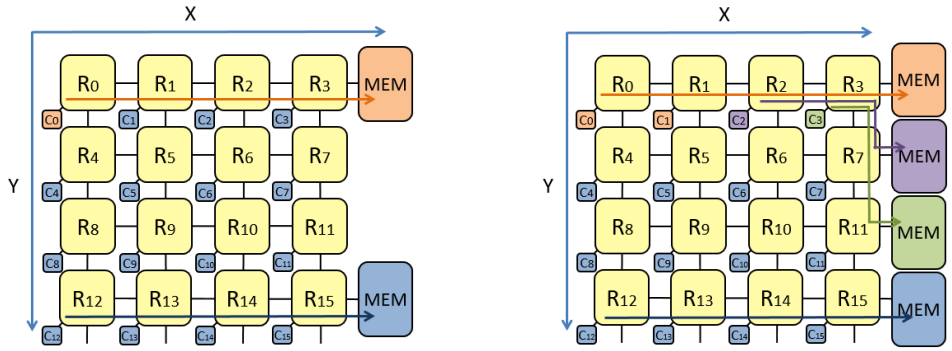
We have performed several experiments with different mesh architectural setups and different injection rates (IR) intended to create high contention in meshes using synthetic traffic. The particular evaluation choices taken allow for determining a priori where contention should occur and what core causes 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



(a) Setup 1 (b) Setup 2

Fig. 5. Illustrative 3x3 mesh setups evaluated



(a) Setup 3 (b) Setup 4

Fig. 6. Illustrative 4x4 mesh setups evaluated

new packet is inserted. The remaining cores, instead, inject packets at a high injection rate (IR = 1) with no packets in-flight restriction.

3x3 wmesh: the first setup (see Setup 1, Figure 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 5b) is like Setup 1 but with C_8 targeting the 3rd memory module attached in R_6.

4x4 wmesh. In Figure 6a (Setup 3) we define a 4x4 2DMesh with two memory controllers, analogous to Setup 1. Setup 4 (see Figure 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:

(1) 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.
(2) 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.

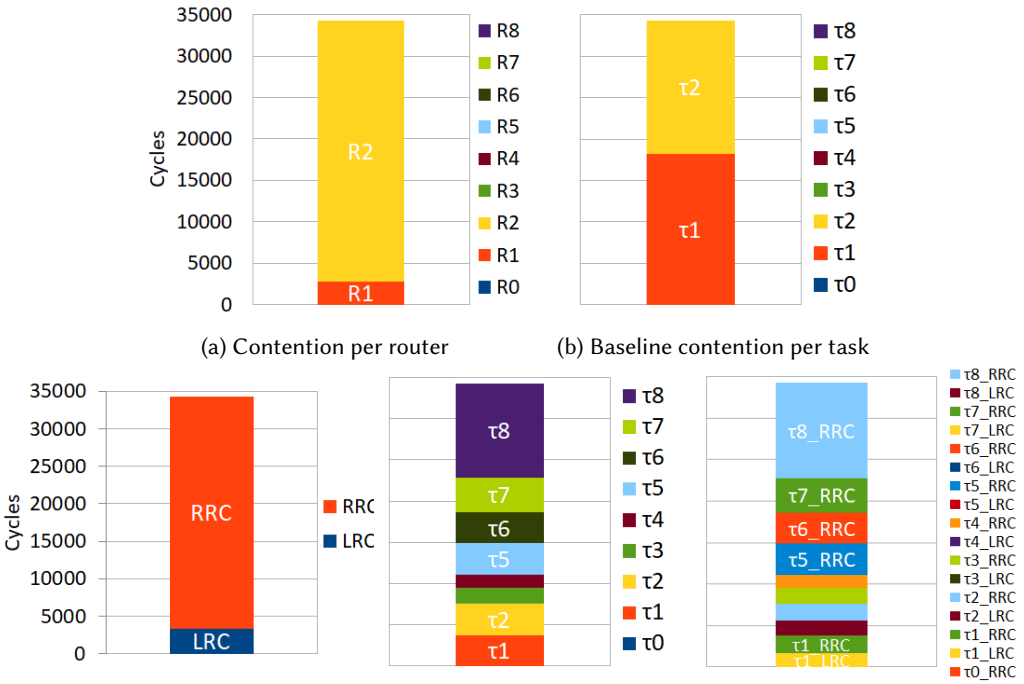
(3) $cont_{\tau_0}$ broken down per contention type (lrc and rrc).
(4) The contention τ_0 suffers from all the other co-running tasks ($cont_{\tau_j \triangleright \tau_0}$) following our PWC definition.
(5) A simultaneous break down ($cont_{\tau_j \triangleright \tau_0}$) per contender, showing lrc 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.

6.1.1 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 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 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

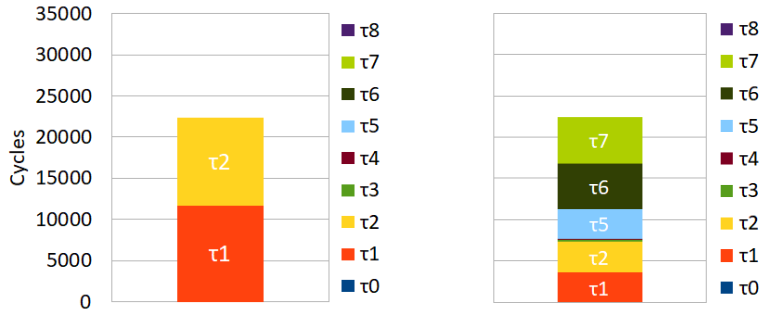
Fig. 7. 3x3 2DMesh $cont_{\tau_0}$ analysis (Setup1)

The baseline contention breakdown, see Figure 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 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 bandwidth in those routers, causing more backpressure on other packets from C_1 and C_2, which propagate backpressure to the packets of the TuA[6]. GRV (Figure 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 bandwidth 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 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.

6.1.2 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 8a and 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 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 7 and Figure 8.



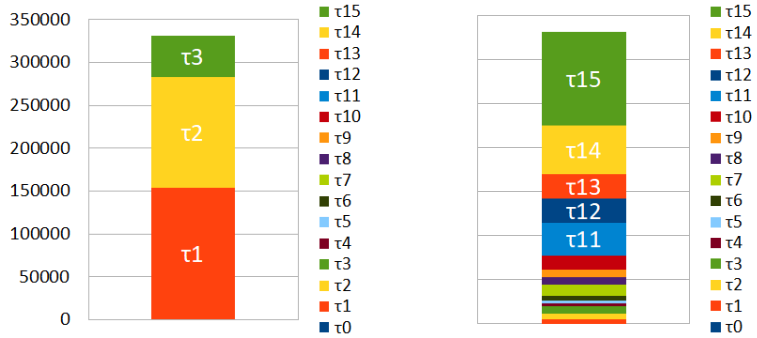
(a) Baseline contention per task (b) GRV. PWC per task

Fig. 8. 3x3 2DMesh $cont_{\tau_0}$ analysis (Setup2)

[6]Notice that in all the synthetic traffic scenarios analyzed in this section, CTs contribution to the TuA contention matches the expected bandwidth 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 bandwidth and potentially the remaining bandwidth unused from the TuA.

6.1.3 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 bandwidth to reach R_3, and the number of cores creating backpressure is also much higher. This can be observed, for instance, in Figure 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 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 9b. Since no further insights are obtained from this setup, we do not deepen on its analysis.



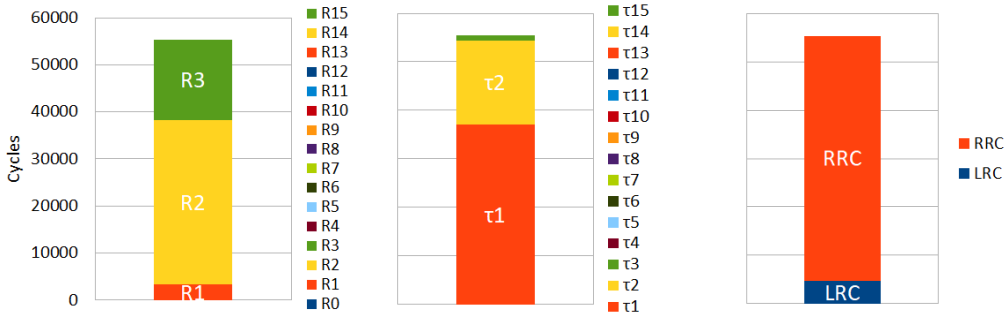
(a) Baseline contention per task (b) GRV. PWC per task

Fig. 9. 4x4 2DMesh $cont_{\tau_0}$ analysis (Setup3)

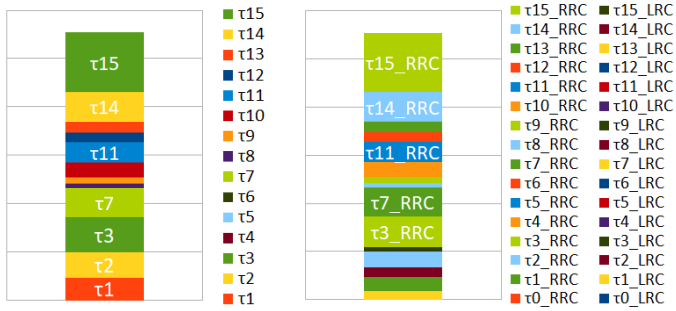
6.1.4 Setup 4 (4x4 Mesh) Result Analysis.

Figure 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 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 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 10b). Note also that backpressure makes, again, cores with higher bandwidth 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 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 bandwidth. 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 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.

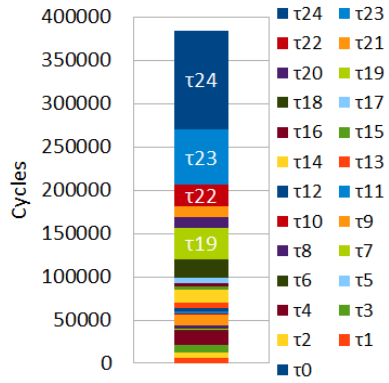


(a) Contention per router (b) Baseline contention per task (c) GRV. PWC per contention type



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

Fig. 10. 4x4 2DMesh $cont_{\tau_0}$ analysis (Setup4)



(a) PWC metric per task (5x5)

Fig. 11. 5x5 2DMesh τ_0 contention analysis

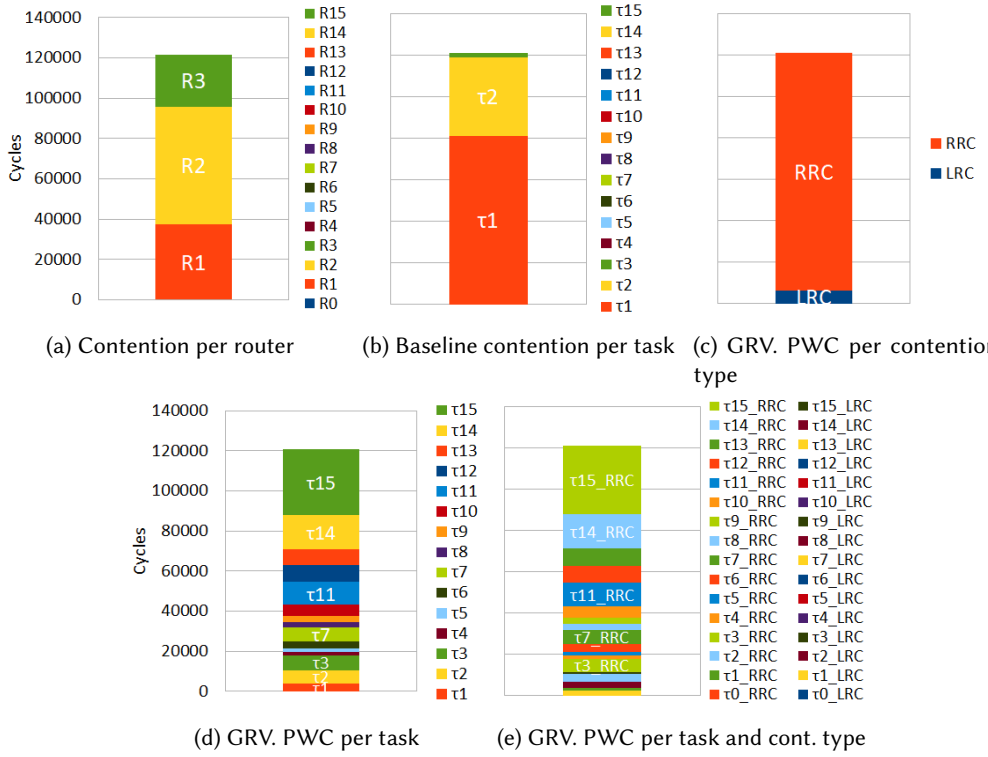
6.1.5 *Larger wmesh (Setup 5 and Setup 6).*

These larger scenarios, see Figure 11a (5x5), show, as in the previous experiments, that the contention τ_0 incurs depends strongly on the bandwidth assignment each task in the wmesh has. When the TuA experiences remote contention, this remote contention matches the bandwidth 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 bandwidth 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.

6.2 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.



(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

Fig. 12. 4x4 2DMesh $cont_{\tau_0}$ analysis (Setup4) with multiple packet sizes

Figure 12 shows the contention analysis for Setup 4 (see Figure 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 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 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 12b does not show any relevant change compared to the $packetsize = 1$ flit analysis while Figure 12c shows that rrc portion type is even bigger than before, because of the same effect described in Figure 12a.

Figure 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 12e confirms the merged effect of increasing rrc contention and each task contention contribution, as already shown in Figures 12c and 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).

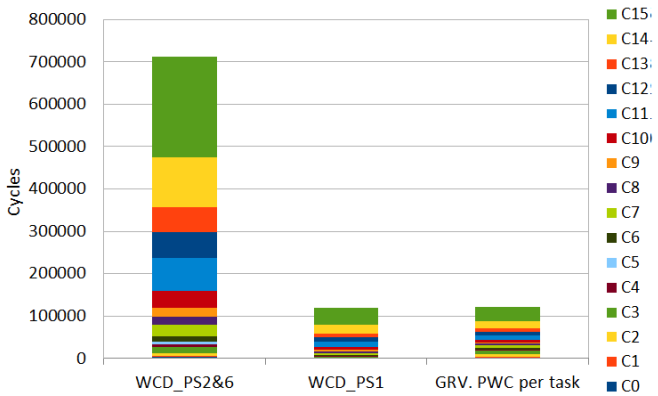


Fig. 13. Average analysis time per packet in experiments

Figure 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 12d. In Figure 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 13), is far from the maximum contention that τ_0 can suffer shown in WCD_PS2&6). It is worth mentioning that WCD computation 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 bandwidth distribution (e.g. round-robin arbitration) and, based on that, we can also derive tasks contribution to that WCD.

Nevertheless, during tasks' execution, bandwidth distribution can vary (e.g. a task generates fewer petitions not using their assigned bandwidth so that other tasks use their remaining bandwidth) and hence, so will vary tasks' contribution to a specific task without exceeding the global WCD computed.

6.3 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 injection rates have been carefully chosen to lead to saturation and no-saturation scenarios respectively as discussed next.

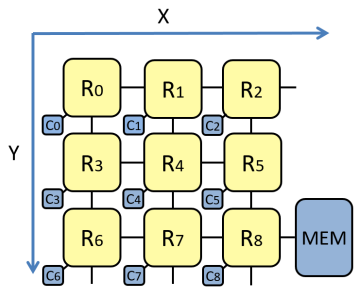


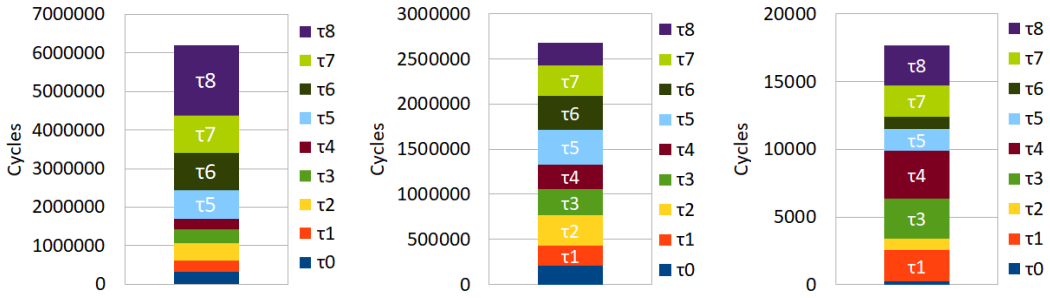
Fig. 14. Illustrative 3x3 mesh setup evaluated (Setup5)

3x3 wmesh: When analyzing real traffic behavior, we define a simpler setup than the ones already shown (see Setup 5, Figure 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 injection rate 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 total injection rate is smaller than the NoC ejection rate, packets do not accumulate and cause lower contention than in the previous case.

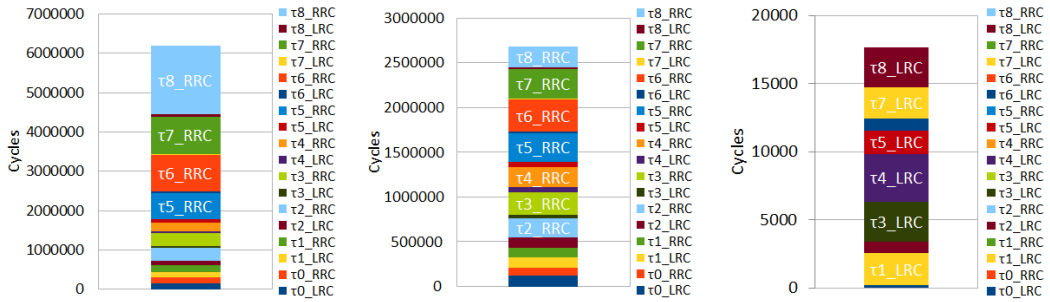
In terms of the results, for setup 5 we show $cont_{\tau_0}$ due to the other co-runner tasks when varying their IR = 1, 0.14 and 0.11 respectively in Figures 15a-15c. As expected, $cont_{\tau_0}$ experienced by τ_0 decreases ad IR for the CTs decreases. In particular, contention is around 6,200,000 cycles for IR = 1; 2,700,000 cycles for IR = 0.14; and 18,000 cycles only for IR = 0.11. Figure 15a for IR = 1 shows that, since all tasks target the same memory controller, tasks that are closer to memory, and consequently have more bandwidth 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 bandwidth distribution to tasks along the NoC.

However, when co-runner tasks IR decreases to IR = 0.14 (see Figure 15b), tasks contention distribution tends to equalize. That is explained because with lower IR (still sufficient to cause



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

Fig. 15. 3x3 2DMesh PWC $cont_{\tau_0}$ per task analysis (Setup5)



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

Fig. 16. 3x3 2DMesh lrc and rrc PWC $cont_{\tau_0}$ per task analysis (Setup5)

saturation), packets from the routers with the highest bandwidth (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 bandwidth in particular, generate lower interference, and since saturation needs contribution from more routers, those receive higher bandwidth. 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 bandwidth of at least 0.16 sustainedly. At a lower scale, the very same effect is captured again in Figure 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 16a-16c). When CTs have high IR (see Figure 16a), most of the contention is rrc and is created by the tasks that have more bandwidth in the NoC. However, when the contenders' IR is low (see Figure 16c), rrc contention is residual and the low contention that takes place in the NoC is lrc.

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 15a-15c, where we see that τ_0 (the TuA) causes contention on τ_0 itself.

6.4 Off-line algorithm analysis

We have implemented the off-line GRV computing algorithm described in Section 5, 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 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 European Processor Initiative [17] implements a 6x6 mesh NoC, so setups considered in this paper are in line with those NoC sizes.

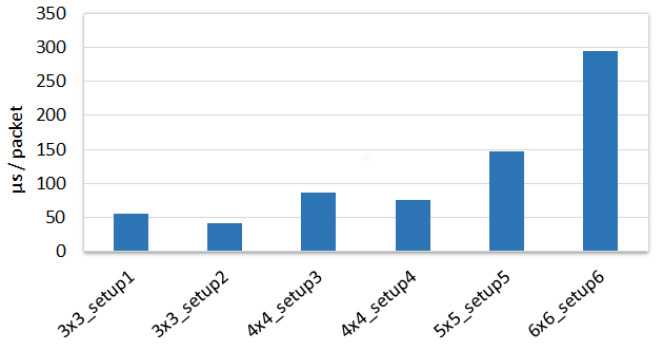


Fig. 17. Average analysis time per packet in experiments

6.5 Assessment of the GRV properties

As shown in Section 4, 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.

7 RELATED WORK

Interference-free designs have been considered the default choice to implement multicore designs for TSES in the past [26, 27, 53, 67]. Unfortunately, real-time specific designs (e.g. time-triggered ones and those based on TDMA) involve high non-recurrent costs that jeopardize their general adoption in the context of TSES [34, 57]. Furthermore, other works have arisen in the same direction for mixed-criticality COTS systems processors[21].

Also in the context of wormhole NoCs, several works [44] 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 wormhole NoCs in hard real-time systems, using virtual channel prioritization coupled with router architectures that support flit-level

preemption has been proposed to limit inter-task interferences in the NoC [41, 55]. Nevertheless, COTS wormhole NoCs 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 [11, 14, 32, 40, 43]. These techniques require bounding the longest contention latency a request can suffer [19, 32, 42], and tracking the maximum number of accesses a task can perform to shared hardware resources [11]. Authors in [11, 43, 47] use PMCs to implement a software approach to enforce quotas on the maximum contention created by tasks during operation. In order 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 [6, 38, 43, 66] or by hardware means [7].

Several works show that worst NoC interference can also be analytically or experimentally derived in regular wormhole NoCs available in COTS manycores [33, 46, 50]. The impact of NoC interference in task execution time has been analyzed in [61] 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 TSES.

So that to fill this gap, different techniques already presented in Section 2, focused on modeling wormhole NoCs time and contention having complex effects when having multiple VCs, flows serialization or different buffer or packets sizes, have been proposed: Scheduling Theory [65], Compositional Performance Analysis [51, 59], Recursive Calculus [12, 36, 37], other that systematically consider the worst-possible contention case [46, 50], or Network Calculus [35, 49] to model worst-case latency of wormhole NoCs included in existing manycore products under certain load conditions [13]. In this line, many recent works extending Network Calculus have been presented [23–25, 55]. 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 [55]. However, testing information during the validation phase is needed to assess whether system integration and verification were performed as planned. Overall, to our knowledge, the current paper is the first work proposing a mechanism to measure and break down contention across contending tasks in a fair manner for validation purposes, thus complementing those works targeting system design and verification.

8 CONCLUSIONS

In this paper, we define pairwise contention (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.

ACKNOWLEDGMENTS

This work has been supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB-C21 funded by MCIN/AEI/10.13039/501100011033 and the European Research Council (ERC) under the EU's Horizon 2020 research and innovation programme (grant agreement No. 772773).

REFERENCES

[1] 2010-2012. *NaNoC Project.* https://sites.google.com/site/nanocproject/

[2] 2018. Apollo, an open autonomous driving platform. https://developer.apollo.auto/index.html.

[3] 2018. NVIDIA drive platforms. https://www.nvidia.com/en-us/self-driving-cars/.

[4] 2022. Kalray MPPA® Manycore. A Massively Parallel Processor Array Architecture. (2022). https://www.kalrayinc.com/products/mppa-technology

[5] 2022. 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] A. Agrawal, G. Fohler, J. Freitang, J. Nowotsch, S. Uhrig, and M. Paulitsch. 2017. Contention-Aware Dynamic Memory Bandwidth Isolation with Predictability in COTS Multicores: An Avionics Case Study. In *Euromicro Conference on Real-Time Systems (ECRTS)*.

[7] J. Cardona, C. Hernández, J. Abella, and F. J. Cazorla. 2019. Maximum-Contention Control Unit (MCCU): Resource Access Count and Contention Time Enforcement. *Design, Automation Test in Europe Conference Exhibition (DATE)* (2019), 710–715.

[8] J. Cardona, C. Hernandez, E. Mezzetti, J. Abella, and F. J. Cazorla. 2018. 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 (TCAD)* 37, 11 (2018), 2451–2461. https://doi.org/10.1109/TCAD.2018.2857298

[9] J. Cardona, C. Hernandez, E. Mezzetti, J. Abella, and F. J. Cazorla. 2018. NoCo: ILP-Based Worst-Case Contention Estimation for Mesh Real-Time Manycores. In *2018 IEEE Real-Time Systems Symposium (RTSS)*. https://doi.org/10.1109/RTSS.2018.00043

[10] Cobham Gaisler. 2011. *Quad Core LEON4 SPARC V8 Processor.* https://www.gaisler.com/index.php/products/components/gr740

[11] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee. 2011. Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus. In *IEEE TrustCom*.

[12] D. Dasari, B. Nikoli'c, V. N'elis, and S. M. Petters. 2014. NoC Contention Analysis Using a Branch-and-prune Algorithm. *ACM Trans. Embed. Comput. Syst.* 13, 3s, Article 113 (March 2014), 26 pages. https://doi.org/10.1145/2567937

[13] B. Dupont de Dinechin and A. Graillat. 2017. Network-on-chip Service Guarantees on the Kalray MPPA-256 Bostan Processor. In *Proceedings of the 2Nd International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems* (Stockholm, Sweden) *(AISTECS '17)*. ACM, New York, NY, USA, 35–40. https://doi.org/10.1145/3073763.3073770

[14] E. Díaz, M. Fernández, L. Kosmidis, E. Mezzetti, C. Hernández, J. Abella, and F. J. Cazorla. 2017. MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding. In *International Conference on Reliable Software Technologies*.

[15] J. Duato, S. Yalamanchili, and N. Lionel. 2002. *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[16] European Union Aviation Safety Agency (EASA). 2022. *AMC 20-193 Use of multi-core processors.* https://sites.google.com/site/nanocproject/

[17] EPI Consortium. 2019-2025. European Processor Initiative. https://www.european-processor-initiative.eu/.

[18] M. Feilhauer, J. Haering, and S. Wyatt. 2016. Current approaches in HiL-based ADAS testing. *SAE International Journal of Commercial Vehicles* 9, 2 (2016), 63–70.

[19] M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla. 2012. Assessing the Suitability of the NGMP Multi-core Processor in the Space Domain. In *EMSOFT*.

[20] International Organization for Standardization. 2018. *ISO 26262. Road Vehicles - Funtional Safety.*

[21] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. 2013. Scheduling of Mixed-Criticality Applications on Resource-Sharing Multicore Systems. In *Proceedings of the Eleventh ACM International Conference on Embedded Software* (Montreal, Quebec, Canada) *(EMSOFT '13)*. IEEE Press, Article 17, 15 pages.

[22] F. Gilabert, M. E. Gómez, S. Medardoni, and D. Bertozzi. 2010. 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 (NOCS)*. 165–172. https://doi.org/10.1109/NOCS.2010.25

[23] F. Giroudot and A. Mifdaoui. 2018. Buffer-Aware Worst-Case Timing Analysis of Wormhole NoCs Using Network Calculus. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 37–48. https://doi.org/10.1109/RTAS.2018.00010

[24] F. Giroudot and A. Mifdaoui. 2019. 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* (Toulouse, France) *(RTNS '19)*. Association for Computing Machinery, New York, NY, USA, 19–29. https://doi.org/10.1145/3356401.3356408

[25] F. Giroudot and A. Mifdaoui. 2020. Graph-Based Approach for Buffer-Aware Timing Analysis of Heterogeneous Wormhole NoCs Under Bursty Traffic. *IEEE Access* 8 (2020), 32442–32463. https://doi.org/10.1109/ACCESS.2020.2973891

[26] K. Goossens, J. Dielissen, and A. Radulescu. 2005. AEthereal network on chip: concepts, architectures, and implementations. *IEEE Design Test of Computers* 22, 5 (Sep. 2005), 414–421. https://doi.org/10.1109/MDT.2005.99

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

[28] C. Hernández and J. Abella and F. J. Cazorla and A. Baradizbanyan and J. Andersson and F. Cros and F. Wartel. 2017. Design and Implementation of a Time Predictable Processor: Evaluation With a Space Case Study. In *Euromicro Conference on Real-Time Systems (ECRTS)*.

[29] Infineon. 2022. AURIX Multicore 32-bit Microcontrollers for automotive and industrial applications. https://www.infineon.com/dgdl/Infineon-TriCore_Family_BR-ProductBrochure-v01_00-EN.pdf?fileId=5546d4625d5945ed015dc81f47b436c7

[30] F. Jafari, Z. Lu, A. Jantsch, and M. H. Yaghmaee. 2010. Buffer Optimization in Network-on-Chip Through Flow Regulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 29, 12 (2010), 1973–1986. https://doi.org/10.1109/TCAD.2010.2063130

[31] J. Jalle, J. Abella, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla. 2013. Deconstructing bus access control policies for Real-Time multicores. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*. 31–38. https://doi.org/10.1109/SIES.2013.6601468

[32] J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F.J. Cazorla. 2015. Bounding Resource Contention Interference in the Next-Generation Microprocessor (NGMP). In *Embedded Real-Time Systems (ERTS)*.

[33] A. E. Kiasari, Z. Lu, and A. Jantsch. 2013. An Analytical Latency Model for Networks-on-Chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21, 1 (2013), 113–123. https://doi.org/10.1109/TVLSI.2011.2178620

[34] A. Kostrzewa, S. Saidi, L. Ecco, and R. Ernst. 2015. Flexible TDM-Based Resource Management in on-Chip Networks. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems* (Lille, France) *(RTNS '15)*. Association for Computing Machinery, New York, NY, USA, 151–160. https://doi.org/10.1145/2834848.2834851

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

[36] S. Lee. 2003. Real-time wormhole channels. *Journal Of Parallel And Distributed Computing* 63 (2003), 299–311.

[37] M. Liu, M. Becker, M. Behnam, and T. Nolte. 2017. 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)*. 275–282.

[38] R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun. 2015. WCET(m) Estimation in Multi-core Systems Using Single Core Equivalence. In *Euromicro Conference on Real-Time Systems (ECRTS)*.

[39] E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla. 2018. High-Integrity Performance Monitoring Units in Automotive Chips for Reliable Timing V&V. *IEEE Micro* 38, 1 (2018), 56–65. https://doi.org/10.1109/MM.2018.112130235

[40] T. Moseley, J.L. Kihm, D.A. Connors, and D. Grunwald. 2005. Methods for modeling resource contention on simultaneous multithreading processors. In *IEEE International Conference on Computer Design (ICCD)*.

[41] B. Nikolić, S. Tobuschat, Leandro Soares I., R. Ernst, and A. Burns. 2019. Real-Time Analysis of Priority-Preemptive NoCs with Arbitrary Buffer Sizes and Router Delays. *Real-Time Syst.* 55, 1 (Jan. 2019), 63–105. https://doi.org/10.1007/s11241-018-9312-0

[42] J. Nowotsch and M. Paulitsch. 2012. Leveraging Multi-core Computing Architectures in Avionics. In *European Dependable Computing COnference EDCC*. IEEE Computer Society, 132–143.

[43] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt. 2014. Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement. In *Euromicro Conference on Real-Time Systems (ECRTS)*.

[44] U. Y. Ogras, P. Bogdan, and R. Marculescu. 2010. An Analytical Approach for Network-on-Chip Performance Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 29, 12 (2010), 2001–2013. https://doi.org/10.1109/TCAD.2010.2061613

[45] M. Panić, C. Hernandez, J. Abella, A. Roca, E. Quiñones, and F. J. Cazorla. 2016. Improving performance guarantees in wormhole mesh NoC designs. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1485–1488.

[46] M. Panic, C. Hernandez, E. Quinones, J. Abella, and F. J. Cazorla. 2016. Modeling High-Performance Wormhole NoCs for Critical Real-Time Embedded Systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 1–12. https://doi.org/10.1109/RTAS.2016.7461342

[47] R. Pellizzoni, A. Schranzhofer, Jian-Jia Chen, M. Caccamo, and L. Thiele. 2010. Worst case delay analysis for memory interference in multicore systems. In *Design, Automation Test in Europe Conference Exhibition (DATE)*.

[48] T. Picornell, J. Flich, C. Hernández, and J. Duato. 2019. DCFNoC: A Delayed Conflict-Free Time Division Multiplexing Network on Chip. In *IEEE Design Automation Conference (DAC)*.

[49] Y. Qian, Z. Lu, and W. Dou. 2009. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In *IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*.

[50] D. Rahmati, S. Murali, L. Benini, F. Angiolini, G. De Micheli, and H. Sarbazi-Azad. 2013. Computing Accurate Performance Bounds for Best Effort Networks-on-Chip. *IEEE Trans. Comput.* 62, 3 (March 2013), 452–467. https://doi.org/10.1109/TC.2011.240

[51] E. A. Rambo and R. Ernst. 2015. Worst-case communication time analysis of networks-on-chip with shared virtual channels. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. 537–542. https://doi.org/10.7873/DATE.2015.0023

[52] S. Ramos and T. Hoefler. 2017. Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 297–306. https://doi.org/10.1109/IPDPS.2017.30

[53] M. Schoeberl and A. Rocha. 2014. T-CREST: A time-predictable multi-core platform for aerospace applications. In *DASIA*.

[54] A. Serrano-Cases, J. M. Reina, J. Abella, E. Mezzetti, and F. J. Cazorla. 2021. Leveraging Hardware QoS to Control Contention in the Xilinx Zynq UltraScale+ MPSoC. In *33rd Euromicro Conference on Real-Time Systems, ECRTS 2021, July 5-9, 2021, Virtual Conference (LIPIcs, Vol. 196)*, Björn B. Brandenburg (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 3:1–3:26. https://doi.org/10.4230/LIPIcs.ECRTS.2021.3

[55] Z. Shi and A. Burns. 2008. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip (NOCS 2008)*. 161–170. https://doi.org/10.1109/NOCS.2008.4492735

[56] SoCLib. 2003-2012. -. http://www.soclib.fr/trac/dev.

[57] J. Sparsoe. 2012. Design of Networks-on-Chip for Real-Time Multi-processor Systems-on-Chip. In *International Conference on Application of Concurrency to System Design (ACSD)*.

[58] Tilera. 2013. *TILE-Gx Processors Family*. https://caxapa.ru/thumbs/281914/TILE-Gx_Processor_PB025_v4_0.pdf

[59] S. Tobuschat and R. Ernst. 2017. Real-time communication analysis for Networks-on-Chip with backpressure. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. 590–595. https://doi.org/10.23919/DATE.2017.7927055

[60] 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. 2010. Merasa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability. *IEEE Micro* 30, 5 (2010), 66–75.

[61] X. Xiang, S. Ghose, O. Mutlu, and N. Tzeng. 2016. 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)*. 456–463. https://doi.org/10.1109/ICCD.2016.7753327

[62] Y. Xiao, S. Nazarian, and P. Bogdan. 2019. 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 (2019), 1416–1427. https://doi.org/10.1109/TVLSI.2019.2897650

[63] Y. Xiao, Y. Xue, S. Nazarian, and P. Bogdan. 2017. A load balancing inspired optimization framework for exascale multicore systems: A complex networks approach. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 217–224. https://doi.org/10.1109/ICCAD.2017.8203781

[64] XILINX. 2018. 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

[65] Q. Xiong, F. Wu, Z. Lu, and C. Xie. 2017. Extending Real-Time Analysis for Wormhole NoCs. *IEEE Trans. Comput.* 66, 9 (2017), 1532–1546. https://doi.org/10.1109/TC.2017.2686391

[66] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. 2013. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*.

[67] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee. 2014. FlexPRET: A processor platform for mixed-criticality systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 101–110.