Journal of Systems Architecture 134 (2023) 102762

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Response-time analysis of mesh-based many-core systems ☆

David García Villaescusa [a,*], Mario Aldea Rivas [b], Michael González Harbour [b]

[a] Ikerlan, S. Coop., José María Arizmendiarrieta Pasealekua, 2, Arrasate, 20500, Gipuzkoa, Spain
[b] Facultad de ciencias, Universidad de Cantabria, Avenida los Castros 48, 39005, Santander, Spain

ARTICLE INFO

Keywords:
Real-time
Scheduling
Modeling
Network-on-chip
Many-core
MAST
Parallella
Epiphany

ABSTRACT

Scheduling models can be used to evaluate whether a particular system is able to meet its timing constrains. In many-core processors, with tens to hundreds of processors in the same chip, the analysis of the timing behavior needs to include the communications network used to exchange messages between the different processors. This paper presents a schedulability model for many-core systems based on a 2D mesh network-on-chip and store-and-forward switching with a limitation on the maximum link utilization rate that makes the analysis tractable. The model has been applied to the Epiphany many-core processor which has 16 cores connected by a 4 × 4 2D mesh. The analysis results have been tested on the real hardware by executing examples with synthetic task workloads. Those tasks are executed in a micro-kernel RTOS that we have developed. We also describe synchronization mechanisms to send messages between the tasks, and we analyze their timing behavior, so that they can be included in the analysis model.

1. Introduction

In the past, the evolution of processors was mostly related to frequency improvement but, since the processors reached a power consumption too high to dissipate, the designers have been improving the processor's performance by having more cores in the same System on Chip (SoC). These processors are called multi-core processors. They are much faster, efficient and achieve their performance improvement by supporting parallel execution on the same chip. Most of these processors share the memory through a common bus, as shown in Fig. 1. When the number of cores increases the shared bus becomes a bottleneck, as it only allows one communication transaction at a time.

Increasing the number of cores to tens or hundreds in the so-called many-core processors, requires solving the shared bus bottleneck. This paper focuses on processors with a 2D mesh Network-on-Chip (NoC) communication system as shown in Fig. 2. NoCs [1] compared to bus architectures excel in energy efficiency, scalability, reusability, reliability and distribution of cores in a homogeneous way. They also require less wires than a shared bus and their power consumption is linear with the number of cores.

To send messages across the cores, the NoC uses packets, as a unit of information. A packet could be a fraction of a message, the whole message or even contain several messages. There are two main switching policies to send packets through a NoC: wormhole and store-and-forward.

- The wormhole scheme divides each packet into fixed-size flits. The header flit has the routing information and it is the only one governing the route. The rest of the flits of a packet follow the header flit through the NoC in a pipelined way until the last flit, called tail flit, reaches its destination. If the header flit is blocked, the rest of the packet's flits will be also be blocked in the NoC.
- The Store-and-forward (SAF) scheme operates with the full packet. Each router waits for the full packet to arrive before operating with it.

The many-core mesh has one tile per core with a typical configuration shown in Fig. 3, with the core, its local memory and the router connecting the tile's core with the neighbor routers in the mesh.

As we will see in Section 2, "Related Work", the time behavior analysis of systems using processors with multiple cores has already been solved with pessimistic approximations. Part of this pessimism is caused by an effect called "back-pressure", which will be described later.

In this paper we avoid the effects of back-pressure by imposing a packet injection rate restriction on the links of the Noc. The paper

☆ This work was supported in part by the Graduate Grant Program of the Universidad de Cantabria and by the Spanish Government and FEDER funds (MCIN/AEI /10.13039/501100011033/FEDER) "Una manera de hace Europa" under Grant TIN2017-86520-C3-3-R(PRECON-I4).

* Corresponding author.
E-mail addresses: david.garcia@ikerlan.es, dgv65@alumnos.unican.es (D. García Villaescusa), aldeam@unican.es (M. Aldea Rivas), mgh@unican.es (M. González Harbour).

https://doi.org/10.1016/j.sysarc.2022.102762
Received 25 January 2022; Received in revised form 23 September 2022; Accepted 15 October 2022
Available online 7 November 2022

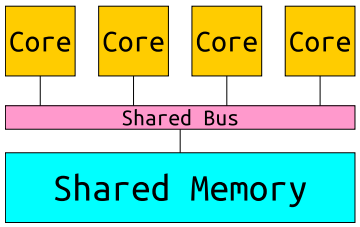

Fig. 1. Multi-core topology. The bus is shared among all the cores and the shared memory.

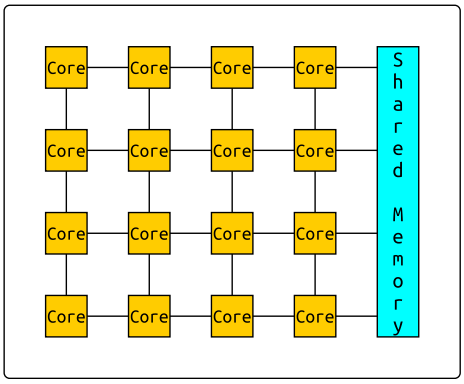

Fig. 2. Epiphany's 2D mesh NoC.

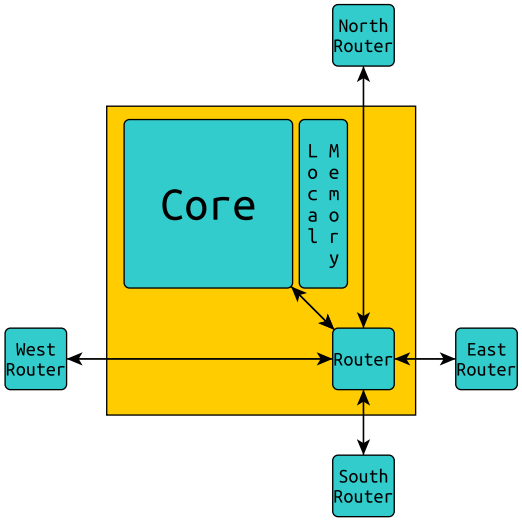

Fig. 3. Many-core tile content. The core is connected to its router which, in turn, is connected to the neighbors' routers.

presents a schedulability model for many-core systems based on a 2D mesh Network-on-Chip and store-and-forward switching. Our model allows us to analyze applications composed of periodic or sporadic end-to-end (*e2e*) flows. In the simplest model, each of these *e2e* flows is composed of a linear sequence of computation steps or threads, each statically mapped to a core, and activated by its predecessor step (see Fig. 4). Other multi-path flows where each step may have several successors and/or several predecessors are also possible, but not considered in this article.

The main contributions of this work are:

- The modeling and analysis technique presented in this paper allows us to study the network and the tasks behavior thanks to the introduction of a limitation on the maximum link utilization rate.
- Our approach includes the modeling of high-level communication mechanisms provided by the operating system (namely Sampling Ports and Queuing Ports).
- Unlike most of the related work, our modeling and analysis technique has been applied and tested on a real platform: the *Epiphany* many-core processor which has 16 cores connected by a 4 × 4 2D mesh.

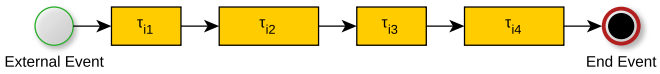

Fig. 4. Example of an *e2e* flow.

The rest of the paper is structured as follows. Section 2 will introduce the related work on modeling many-core systems based on NoCs. Section 3 will present the system model. A comparison with another model from the related work is exposed in Section 4. The adaptation of the model to the Epiphany many-core is shown in Section 5. A simplified application is presented in Section 6 to show the modeling process. High-level communication mechanisms are presented at Section 7 with examples using those communication mechanisms being shown in Section 8. Finally, Section 9 concludes the paper.

2. Related work

2.1. NoC timing studies

The first hard-real-time communication studies for multi-hop switched systems were made over Ethernet hardware with bounded end-to-end delays as shown in Zhang [2] and Yiming [3]. The positive impact of increasing the number of switching hubs and the topology influence was studied by Lee [4]. Those studies set the initial considerations that have been evolving through time.

When there are several packets traversing the NoC, determining the exact timing of every packet is an NP-hard problem. Shi [5] mentions that the requirements of real-time applications introduce the need for a scheduling strategy and an analysis approach to predict whether all the real-time packets can meet their timing bounds. In our paper a heuristic analysis of an existing processor is introduced avoiding the usage of heavy computation resources for its calculation.

The test analysis to check whether a multi-core system based on a NoC can fulfill all the constraints of a specific system has been done by Indrusiak [6]. In that paper it is assumed, for the sake of simplicity, that the tasks can send a single message just when they finish their computation process. For the same reasons, we will make a similar assumption in our paper.

Becker presented a work [7] where one application is mapped to one tile, and it has implemented in a Kalray MPPA processor. Becker also faced the orchestration of the access to the shared memory [8] where each core is assigned a private memory bank and access to local and off-chip memory is exclusive among all cores. A non-preemptive time-triggered schedule is utilized to orchestrate the access to the shared memory statically assigned to tasks.

Several Worst-Case Response Time (WCRT) models have been presented for theoretical 2D mesh wormhole NoCs [9,10]. The back-pressure has been identified as a big problem in the calculus of the NoC impact on a particular packet traveling through the NoC, as it may imply worse timing than expected. Back-pressure is defined as the situation when, due to traffic interference in the NoC, a core generating traffic has its execution stalled (causing the core to stop its execution). The traffic can affect the generating core by having the output link to the first router of the message route busy. For those models, each step

of an *e2e* flow produces a single message which is sent immediately after it finishes its computation.

Boyer [11] introduced a network calculus formulation designed to configure the NoC traffic limiters, that also computes guaranteed upper bounds on the NoC traversal latencies. Some approaches and models have been presented and compared on two case studies without any implementation in a real processor.

Another paper considering back-pressure in the NoC analysis has been done by Tobuschat [12]. In their analysis, the authors also determine safe upper bounds on the latency of individual packets. The model of the NoC used in their work is similar to the one we use; they make an evaluation of the analysis results for different buffer depths, including the case of a buffer depth of one, with a queue of a single element per input port and round-robin arbitration, which is what is considered in our paper. Their analysis introduces some pessimism as we will show in the comparison performed in Section 4 with the results of our model.

Both wormhole and SAF schemes have also been studied [13] avoiding the analysis when a communications channel is shared by more than one *e2e* flow. The paper includes a simulator to check the timing behavior accuracy.

Gopalakrishnan [14] studied how to set the time bounds in a real-time system using the various buses linking multiple hosts, by capping the number of hops for a real-time message. Another work done with a certain number of buses linking a distributed heterogeneous system was done by Roy [15].

Nikolić [16] has also studied the NoC timing behavior but that paper also considered for the analysis the number of links traversed and the link traversal delay in the NoC analysis. The routing delay could generate a self-imposed buffering in each traversed router. That is not the case contemplated in our paper, where there is only one buffer at each router link (so we have the same analysis no matter the buffer size of the system). In that paper they got the conclusion that bigger buffers do not necessarily lead to better results.

Most of the work done in NoCs is focused on the Worst-Case Traversal Time (WCTT), considering only the time needed to travel through the NoC rather than the WCRT of a series of tasks located at different tiles of the processor needing to communicate with each other. The analysis of the WCRT of the e2e flows and its tasks is the main target of this paper.

Dasari [17] determined than an upper bound for the extra time due to the core stalls waiting for the data to be transferred over the underlying network should be established. This solution uses a branch-and-prune approach.

Puffitsch [18] also simplifies the execution model to approach the execution of safety critical applications on many-core processors in a predictable manner. The task is scheduled through partitioned non-preemptive off-line scheduling, data and code are stored in the local memories to reduce implicit accesses to external memory as much as possible, tasks communicate via message passing and delays for explicit communication between cores are pre-computed.

A many-core model to accomplish real-time requirements was proposed by Metzlaff [19] based on four assumptions for a many-core:

1. Small and simple cores.
2. A cache-less memory hierarchy.
3. A static switched NoC.
4. An independent task and network communication analysis.

These assumptions are accomplished by the *Epiphany*'s design:

1. The eCore is a simple core.
2. Each eCore has its local memory and messages between cores end up in writing/reading that local memory directly.
3. The NoC has a static routing policy.
4. In this paper we have different models for the tasks and the network, although they are analyzed together.

Some papers simulate behaviors [5,6,9,10,13,14,16,19] while others use switched Ethernet [2–4]. In this paper we implement the modeled system in a 2D mesh based many-core processor to compare the model with the real execution.

A different line of research also done with NoC-based many-core processors is the task allocation. This has been studied by Benchechida [20] with a model with certain similarities to ours. In our paper we do not address task allocation.

2.2. Modeling and analysis tools

The MAST toolset [21] is an efficient set of tools that can be used to apply and compare different schedulability analysis techniques. In this paper we use one of these tools, called offset-based response time analysis [22]. MAST defines an open model for describing event-driven real-time systems. The tool is not directly suitable for many-core processors so an adaptation has been done to allow studying the timing behavior of a many-core processor. This is explained in the following section. Even though we have used MAST, other tool like the ones used in effect chains [23–25] could be used. The main contribution of this paper is in the modeling techniques, rather than in the analysis techniques themselves.

The model defined in this paper includes synchronization mechanisms for many-core processors that have been introduced before [26,27], and in this regard this paper is a continuation of those works.

2.3. Many-core processors

The model presented in this paper has been applied to the Epiphany many-core (described in Section 5). Other many-core processors were considered as platforms for this work and, after a deep study, they were discarded. These many-core processors were:

- **Intel Xeon Phi** is a processor family that has been evolving since 2010. Its architecture consists in a 2D mesh connecting the tiles, each tile with 2 cores. The many-core processor with the Knight Landing architecture [28] has 72 cores. It is not possible to deactivate the L2 cache coherency and there is a dual core processor in each tile.
- The many-core of the **Tilera-GX** architecture [29] is divided in tiles, each tile with a 64-bits core, cache memory (L1 and L2) and a switch used as an interface with the mesh communicating the tiles and providing L2 cache coherency to all the cores. The L2 cache has 256kB size.
- **Kalray** has a many-core MPA-256 [30] which is focused on embedded and real-time systems. It has 288 cores distributed in 16 groups called clusters and 4 I/O subsystems each with 4 cores able to access the external memory. Each cluster has 16 execution cores with direct connection between them plus 1 management core. Each cluster forms a 2D 4×4 NoC mesh.
  The many-core architecture provides a direct partitioned system. Each NoC has bounded messages traversal time. The cores have local memory with 128kB size.

Even though these many-cores were good candidates as evaluation platform for our model, they are relatively complex architectures. We have chosen a simpler and more accessible platform, the Epiphany many-core described in Section 5, which is based on a simple 4×4 2D mesh NoC with no cache coherency between the cores, which simplifies the model and analysis. The straightforward architecture of Epiphany, with only a single mesh type, eases the analysis and modeling of the NoC.

3. System modeling

The model used in this paper to describe the execution and communication platform as well as the application tasks and messages is based on the MAST model, using specific elements for modeling the many-core platform and the communications among the different cores, instead of messages and communication networks used as modeling elements for traditional real-time distributed systems. The MAST model is conceived to describe the fundamental details of the timing behavior of a software system and its hardware platform, with the objective of analyzing its schedulability, i.e., the ability to meet its timing requirements under a particular scenario.

The original model and the way to use it in order to have a suitable model for a mesh-based store and forward many-core processor will be explained in the following subsections.

3.1. Basic model elements

The MAST model for the software architecture is based on *e2e* flows activated through external events that determine the workload. Each of these flows is described as a directed acyclic graph in which events are used to interconnect activities performed by the system, these activities are exclusive to . These activities are called steps in the MAST model. Each arrival of an external event activates the execution of an instance of the *e2e* flow.

In its simplest form, the *e2e* flows follow the so-called linear flow model, in which the finalization of a step generates an internal event that can end up triggering the execution of a subsequent step or ending the flow if it is the final step (see Fig. 4).

The model of a many-core processor has n processing resources $PR_1, PR_2, \ldots, PR_n$, one per processing core. Each core has a scheduler with a particular scheduling policy, such as fixed priorities. Since this paper is focusing on many-core processors with 2D mesh-based NoCs, we adapt the notation of a processing resource by defining a PR_i with its coordinates on the mesh using PR_{xy} where x is the column and y is the row of the processor in the mesh.

Γ_i represents an *e2e* flow where each step is noted as τ_{ij}. Each step is statically assigned to a particular processing resource PR_{xy} with a determined priority $prio_{ij}$. Several steps can be mapped on the same core as shown in Fig. 5 where the four steps of an e2e are mapped on two cores.

An *e2e* flow Γ_i is triggered by the external event e_i. This external event could have one of these three different types of activation patterns:

- *Periodic event.* Generated at regular intervals of time with a T_i period.
- *Sporadic event.* Generated at irregular time intervals. They have a minimal time between activations.
- *Aperiodic event.* Generated at irregular time intervals with unbounded arrivals in a given time interval.

Each step belonging to Γ_i has a Worst-Case Execution Time (WCET) C_{ij}, corresponding to the worst measured execution time of a thread in a core or an upper-bound estimation on it, assuming no contention from other threads or messages. The step also has a best-case execution time (BCET) or a lower-bound estimation on it, C^b_{ij}.

Each *e2e* flow Γ_i has a relative deadline D_i, implying that its last step must be completed by this deadline, with respect to the arrival of the external event. The deadline can be arbitrary, i.e., smaller than, equal or larger than the periods.

Each step τ_{ij} has a Worst-Case Response Time (WCRT) R_{ij} and a Best-Case Response Time (BCRT) R^b_{ij}, relative to the arrival of the external event. The WCRT of a linear *e2e* flow, R_i, is the worst-case response time of its last step. The system is considered schedulable if, for each *e2e* flow Γ_i, $R_i \leq D_i$.

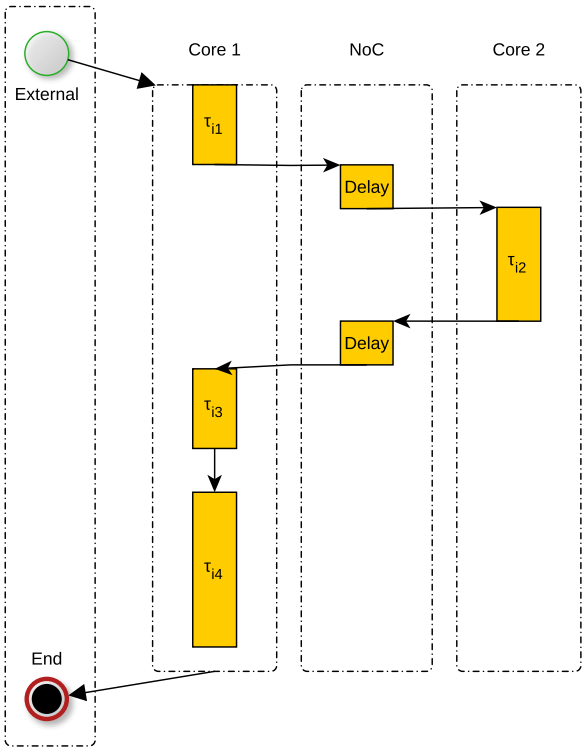

Fig. 5. *e2e* flow mapped representation.

Modeling a many-core processor requires modeling the messages transmitted through the NoC. These messages are generated for every hop in an e2e flow, i.e. when the processing resource of step τ_{ij} is different from the processing resource of the successor step $\tau_{i(j+1)}$. In such situation step τ_{ij} can generate one or more messages, m_{ijk} at the end if its execution. As it will be justified in subsequent sections, the generation time of all the messages is included in the execution time of τ_{ij} and a MAST delay element is introduced to model the message traversal latency through the NoC. A delay in MAST is an event handler that generates its output event after a time interval has elapsed from the arrival of the input event.

As a summary, a flow Γ_i, as the one shown in Fig. 5 has a period T_i, deadline D_i and is formed by an ordered set of steps $\{\tau_{i1}, \tau_{i2}, \ldots, \tau_{ix}\}$. Each step τ_{ij} is defined as $\{C_{ij}, C^b_{ij}, prio_{ij}, PR_{xy}\}$ respectively representing its WCET, BCET, priority and the processing resource where the step is going to be executed. For each hop in the *e2e* flow a delay is introduced in order to model the NoC behavior. The duration of these delay blocks will be discussed in the following sections.

3.2. Network-on-Chip basic parameters

This paper contemplates a 2D mesh but the model that we have developed could be usable for other kinds of NoC with a store-and-forward switching control technique. In our model, messages are divided into packets. A packet is considered indivisible, so it can be said that a packet equals the concept of a flit in other networks. Each packet is routed individually through the links of the network (depicted as arrows in Fig. 6). As shown in Fig. 6, we assume that there is a buffer of one packet capacity per link. The router is in charge of routing the packets stored in its input buffers to the output links. A router will only send packets to those links whose buffer is empty.

The NoC timing is defined by three parameters:

- NoC frequency (F_{NoC}). This value is the number of packets generated by a core that the NoC is able to absorb, per time unit, in

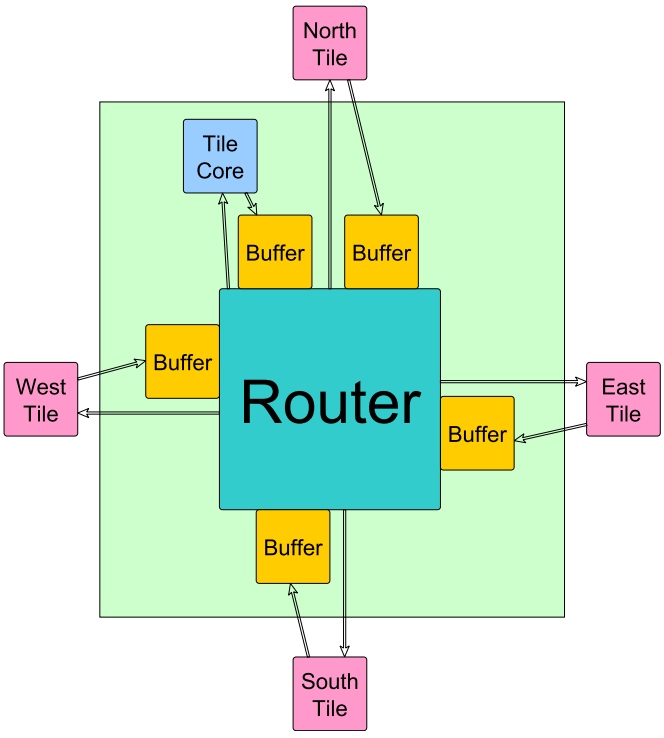

Fig. 6. Router showing its input and output links as well as its buffers.

the absence of contention from other network traffic. The inverse of the NoC frequency is the NoC cycle, that is, the time required by the NoC to absorb one packet generated by a core.

- Hop latency (L_H). The time needed by a packet to go from one router to a neighbor one without any other packet competing for the same resources.
- Routing arbitration latency (L_R). The time required by the router to perform an arbitration decision. This latency is added each time a packet misses its arbitration turn in the router.

There could be more than one NoC used for different types of messages (i.e., write and read messages) with different values for their NoC parameters. Only the messages traveling through the same NoC would share the same NoC resources.

3.3. Modeling the messages

A mesh-based many-core processor has shared memory and reading from or writing to this memory implies messages being sent through the NoC (operations that in some literature are referred to as "pull" and "push"). This means that the execution of a task could include multiple messages to be sent in the middle of this execution, which would complicate modeling the application. Since we assume that each core has its own local memory, for efficiency we assume that the task execution always uses this local memory, we avoid the use of the shared memory, and we limit writing to the local memory of other cores to happen only at the end of the task's job.

In some cases, high-level synchronization primitives are used for the message sent at the end of a step, perhaps implying the use of distributed mutexes or other similar primitives that require remotely reading the local memory of the destination core. In consequence, there could be remote read operations as well as write operations at the end of a step, both implying messages traversing the NoC. We will later describe the high-level synchronization primitives that we have implemented and describe how to model the associated messages.

For each activation, the finalization of a step could activate another step allocated in the same or a different PR. Activation messages between steps mapped at different PRs have to travel through the NoC. A message is composed by a set of packets. The message's destination step will be activated when the last packet of the last message is received.

A key element of our model is the packet generation rate of a message. Given a message, m_{ijk}, we define the packet generation rate of m_{ijk}, denoted ρ_{ijk}, as the inverse of the minimum number of NoC cycles that could elapse between sending two packets of that message to the NoC. When the message is formed by just one packet, ρ_{ijk} will be the inverse of the minimum interval between this packet and the closest packet of the previous or the next messages sent from its PR with the same output link. In the special case that there is only one packet sent from a PR, ρ_{ijk} will be the inverse of the minimum interval possible between the packet sent during a step activation and the packet sent in the next activation. That minimal interval is the best-case response time, $R^{b_{ij}}$, which is multiplied in Eq. (1) by F_{NoC} to obtain the desired unit $cycles^{-1}$:

$$\rho_{ijk} = \frac{1}{R^b_{ij} \cdot F_{NoC}} cycles^{-1} \tag{1}$$

A read operation between PR_{xy} and $PR_{x'y'}$ requires two messages: a message from PR_{xy} to $PR_{x'y'}$ indicating the memory area to read and a second message from $PR_{x'y'}$ to PR_{xy}, with the contents of the read memory area. As a consequence, every read message m_{ijk} generates another write-back message, $m_{ij(k+1)}$, that will write the data swapping the source core and the destination core. The write-back messages are written in the reader's local memory by the network interface, which also means that the read message must have two addresses: where to place the data and where the data is read from. The write-back message will be written directly by the network interface into the local memory of the core executing the read message operation.

As a summary, a message m_{ijk}, the kth message generated at the end of step τ_{ij} to a PR other that its own, is defined as a 4-tuple $\{\rho_{ijk}, \mu_{ijk}, t_{ijk}, d_{ijk}\}$ where:

- ρ_{ijk} the generation rate of the message's packets measured in $cycles^{-1}$.
- μ_{ijk} the number of packets forming the message.
- t_{ijk} the type of the message (read, write or write-back message).
- d_{ijk} the destination step of the message, it takes the source role when the type of the message is write-back.

In our model, the generation of the packets of the messages produced at the end of a step along with the time required to inject them into the NoC will be included in the step's execution time, as the processor is busy during these actions. We will also charge to the step's execution time the interval elapsed to complete a read operation, which includes the times required by the read request message to reach the destination core, the network interface to perform the operation and by the generated write message to travel back through the NoC, since we assume the processor is stalled waiting for this last message to arrive.

The step execution finishes when the last packet of the last message is placed in the router of its core. The next step of the *e2e* will be activated as soon as that packet arrives to its destination core. In our model, the time required by that last packet to travel through the NoC is modeled using a MAST *delay* with a maximum and a minimum relative time.

The minimum interval time assigned to the MAST *delay* element is the time required by the packet to travel through the NoC in absence of other competing packets. In such situation, the traversal time of the last packet of a message m_{ijk} only depends on the number of hops or routers the packet must go through, H_{ijk}, including the origin and destination routers, and the hop latency L_H. This best-case traversal time for the last packet of message m_{ijk}, TT^b_{ijk}, is shown in Eq. (2).

$$TT^b_{ijk} = L_H \cdot H_{ijk} \tag{2}$$

In order to obtain the maximum interval time of the *delay* element we need to estimate the worst-case traversal time of the packet, which should take into account other packets traversing the NoC. This value will be calculated in Section 3.5 after having introduced the maximum packet rate restriction in the next section.

3.4. Maximum packet rate restriction

The interference among packets complicates the calculation of the worst-case traversal time for messages (it is an NP-hard problem [5]). The existing NoC schedulability analysis techniques are pessimistic when bounding the effects of the rest of the messages, to achieve feasible analysis techniques.

In our approach, we simplify the analysis by avoiding the back-pressure thanks to the following rate limitation: let us call $M_{xy\to x'y'}$ the set of higher generation rate messages that use a link between the routers of cores PR_{xy} and $PR_{x'y'}$. When there is more than one message generated from the same PR that uses the same link, only the message with higher generation rate is included in $M_{xy->x'y'}$. We call the accumulated transmission rate of the link, $P_{xy\to x'y'}$, the summation of the generation rates of messages in $M_{xy\to x'y'}$:

$$P_{xy\to x'y'} = \sum_{m_{ijk}\in M_{xy\to x'y'}} \rho_{ijk} \tag{3}$$

We propose as a condition to analyze the schedulability of a system to have every link with an accumulated transmission rate not higher than the inverse of the router arbitration latency $\frac{1}{L_R}$, in other words, that $P_{xy\to x'y'} \leq \frac{1}{L_R}$, for every link in the NoC.

If that condition is verified by all the links we can assure that the network is able to transmit the packets at least at the same rate that they are generated.

The aforementioned restriction allows us to get rid of most of the interference among messages. According to [9] there are two sources of interference among messages to be considered: direct and indirect interference. Direct interference happens among messages that share some link of their route when the link is not able to transmit the packets at the rate they arrive. In such situation back-pressure could be produced in the messages and the message generation in the core could get stalled. For its part, indirect interference could happen when a back-pressured message due to a direct interference among two or more messages ends up affecting the traversal time of another message. For example, suppose we have three messages A, B and C, where messages B and C share the same link and messages A and B share a predecessor link in B's path. If message B suffers back-pressure due to a direct interference with message C, message B will be blocked in the router and will consequently block its corresponding input link. If this input link is shared with message A, then A will suffer an indirect interference from message C, even though it does not share links with it.

Direct interference cannot generate back-pressure when the shared links verify the maximum rate restriction since this restriction ensures that packets will be able to be transmitted through the links of the NoC at the rate they are generated. Indirect interference can neither exist in systems that fulfill this restriction since, in order to have indirect interference, a direct interference must generate back-pressure in the first place.

However, there is a small and bounded source of interference that remains in systems that verify the maximum rate restriction. This kind of interference, related with the routers arbitration policy, will be discussed in the next section.

In consequence, this restriction largely simplifies the analysis. Besides, this limitation is not very restrictive to the system design, as NoCs usually have high capacity. A discussion on the impact of this limitation on the Epiphany architecture is presented in Section 5.4. Moreover, if a system is not analyzable due to the restrictions previously proposed, the different steps of an *e2e* flow could be remapped to fulfill our restrictions.

As we will see, when the model is applied to the *Epiphany* many-core, in Section 5, this assumption is reasonable, as the restriction usually imposes no limitations on the software.

3.5. Packet maximum traversal times

If the system being analyzed fulfills the accumulated transmission rate restriction, $P_{xy\to x'y'} \leq \frac{1}{L_R}$, for every link, the interference among packets will be bounded, because only a limited number of packets could affect the shared links on the packet route as the link utilization is limited by the maximum rate. The only interference among packets will be the one caused by the router arbitration policy.

When two or more packets are simultaneously located at the entry buffers of a router and they have the same output link, the router must apply a routing policy for arbitrating access to the shared output link.

The most usual routing policy is round-robin. Using this policy, a packet trying to access an output link of a router will wait, in the worst-case scenario, until the router sends through that link a single packet allocated in each of its input buffers. The delay caused for each competitor packet is the L_R NoC parameter.

Since we assume that the packet routes are known, we can determine the maximum number of competing packets a specific packet could encounter at each of the traversed routers. For each router, r_{xy}, on the route of message m_{ijk}, we call nb_{ijkxy} the number of input buffers used by messages sharing the same output link as the message being analyzed (without taking into account the buffer used by the message itself). The routing arbitration causes interference called I_{ijk}, accumulated for the routers traversed by the message, calculated as:

$$I_{ijk} = \sum_{r_{xy}\in route_{ijk}} nb_{ijkxy} \cdot L_R \tag{4}$$

In consequence, the worst-case traversal time of any packet of message m_{ijk}, is the best-case time obtained in Eq. (2) plus the routing interference:

$$TT_{ijk} = TT^b_{ijk} + I_{ijk} = L_H \cdot H_{ijk} + I_{ijk} \tag{5}$$

An example of the WCTT calculation of a packet in the *Epiphany* architecture will be seen in Section 5.

As it has already been mentioned, round-robin is the most common routing policy, but our model could be applied to any other fair policy. The only requirement to apply our model is to know the maximum delay the policy could suppose to a packet in relation with the maximum number of competing packets.

3.6. Modeling the NoC with MAST elements

Summarizing what has been exposed in this section, when two consecutive steps in an *e2e* flow, τ_{ij} and $\tau_{i(j+1)}$, are allocated to different PRs, at the end of the execution of step τ_{ij} one or more read and write messages will be generated in order to activate step $\tau_{i(j+1)}$. As stated previously, the time required to generate the packets and to inject them into the NoC are included in C_{ij}. In the case of read messages this time also includes the traversal time through the NoC of the read request and of the write-back response message.

As a consequence, the NoC traffic will affect the execution time of the steps. In that way, each read message m_{ijk} will increase the WCET in a value equal to the maximum interference that the last packet of the message can suffer according to Eq. (4), plus the interference suffered by the corresponding write-back message $m_{ij(k+1)}$. Being C_{ij} the worst-case execution time of τ_{ij} when estimated in isolation, we can calculate C'_{ij} as:

$$C'_{ij} = C_{ij} + \sum_{\forall k: t_{ijk}=read} (I_{ijk} + I_{ij(k+1)}) \tag{6}$$

It is important to notice than write messages that are not the response to a read request will never increment C_{ij}. This only would

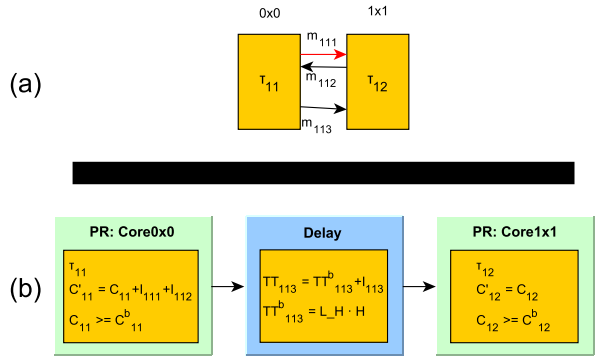

Fig. 7. (a) Two tasks mapped in two different cores sending a read request and a write message. (b) How those two tasks are modeled in MAST.

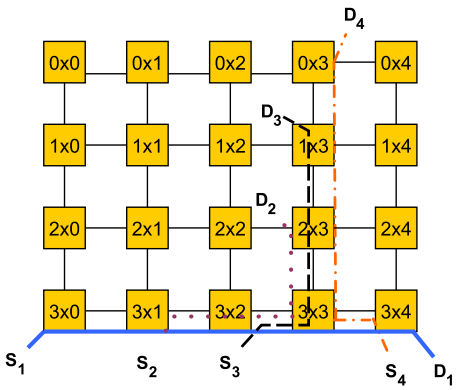

Fig. 8. Evaluation system proposed in [12].

happen in stall situations but thanks to the maximum packet rate restriction this is never going to happen in the systems we are able to model.

The interval between τ_{ij} injecting the last packet into the NoC (and thus, finishing its execution) and the following step $\tau_{i(j+1)}$ being activated is modeled by a MAST *delay* element with minimum and maximum interval times obtained from Eqs. (2) and (5), respectively.

A transformation of two tasks executing in different cores of a many-core processor into a MAST model is shown in Fig. 7. In Fig. 7.a we can see that there are two tasks, τ_{11} executing in core 0×0 and τ_{12} executing in core 1×1. Task τ_{11} sends a read request m_{111}, receiving the write-back message m_{112}. Task τ_{11} ends its execution by sending the write message m_{113}.

In Fig. 7.b we can see how the tasks of Fig. 7.a are modeled using MAST elements:

- Task τ_{11} will be modeled using a step executed in PR_{00} with a WCET C_{11} and a BCET C^b_{11}. As the task is performing a read operation the read request message m_{111} could suffer an interference I_{111} by the NoC and the write-back message m_{112} could suffer an interference I_{112}. This means that the worst-case scenario must include all the possible interference as it is formulated in C'_{11}.
- The time required by the last packet of m_{113} to travel from core 0×0 to core 0×1 is modeled using a delay block. The minimum interval time of the delay is the best-case traversal time of m_{113}, T^b_{113}, calculated using Eq. (2) with H_{113} equal to three (the number of routers traversed by m_{113}, including its own). For its part, the maximum interval time of the delay is the worst-case traversal time of m_{113}, T_{113}, calculated using Eq. (5), which includes the interference suffered by the last packet of m_{113} due to other packets in the NoC.
- Task τ_{12} will be modeled using a step executed in PR_{11}. No NoC interference is possible for this task which does not send any message through the NoC.

4. Modeling and behavior comparison

To validate and evaluate our model, in this section we will apply it to a system described in the literature, in particular in the work performed by Tobuschat [12]. We have chosen this example since it is based on a mesh similar to the one used in our model and because the paper provides enough data to reproduce it.

4.1. System description

The system described in [12], shown in Fig. 8, is made of four *e2e* flows each of them with two tasks. S_1 is the first task and D_1 the second task of the first end to end flow, for the second flow S_2 is the first task and D_2 the second one, etc. The initial task of each e2e flow sends a message of one packet divided in four flits. The message uses 12.5% of the NoC bandwidth (i.e., the initial task generates a packet of 4 flits each 32 cycles). The routers have input buffers with a capacity of four flits and require four cycles to route each packet (one cycle per flit).

Our model cannot be directly applied to this system since we assume that flit and packet are equivalent. In order to do a fair comparison, we will analyze a system with equivalent NoC bandwidth usage and buffer size: in the modeled system, the initial task of each e2e flow generates a packet each 8 cycles and the router input buffers have a capacity of one packet. The basic parameters of the modeled NoC will be: $F_{NoC} = 1cycle^{-1}$, $L_H = 1cycle$ and $L_R = 1cycle$. In order to compare the response times obtained with both models we must take into account that one of our cycles is equivalent to 4 cycles of [12].

The analysis in [12] only includes the messages traversal times, it does not include the task scheduling in the processors. In order to model an equivalent system, we will consider the first task of each e2e flow to have a worst-case execution time of one cycle (just the time required to generate one message) and the last task to have an worst-case execution time of zero cycles. We have chosen an arbitrary value for the e2e flow deadlines to be the same as the period.

4.2. Rate analysis

Before performing the analysis, we must verify the system fulfills the maximum packet rate restriction. As stated before, the first task of each flow generates a packet each 8 cycles, in consequence, all messages have a $\rho = \frac{1}{8}$ cycles^{-1} $= 0.125$ cycles^{-1}.

Fig. 9 shows the accumulated transmission rate of each link measured in cycle^{-1}. As it can be seen in the figure, all the accumulated transmission rates are below the limit that, for this NoC, is $\frac{1}{L_R} = 1$ cycle^{-1}.

4.3. System modeling with MAST elements

The e2e flow Γ_1 of the evaluation system is modeled as shown in Fig. 10. The rest of e2e flows (Γ_2, Γ_3 and Γ_4) are quite similar, changing the index referring to the flow and the TT_b for the delays m214 and m314 which are 4 cycles. The start event of each flow has a period of 8 cycles and triggers the initial task that, in turn, executes for 1 cycle to generate a message. The internal events that connect each step or delay with the next are represented with big dots.

The best-case traversal time of the messages is calculated with Eq. (2) with $L_H = 1$ cycle, consequently, it takes 5 cycles for flows 1 and 4 (whose messages travel through five routers) and 4 cycles for flows 2 and 3 (whose messages travel through four routers). The worst-case traversal time of the messages, that includes the worst-case interference of other messages due to routing arbitration, is calculated

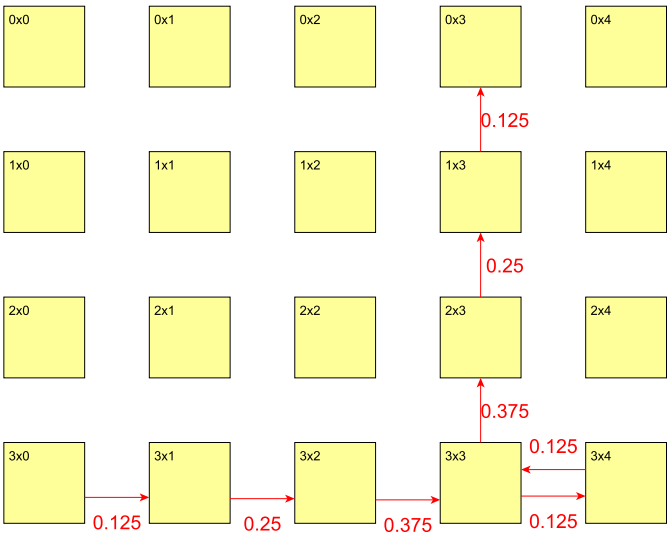

Fig. 9. Accumulated transmission rate of each link of the evaluation system.

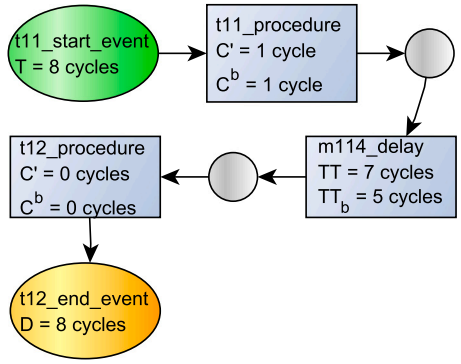

Fig. 10. MAST model of Γ_1 in the evaluation system.

Table 1
Comparison among our model and the results provided in [12].

Flow	MAST model		Tobuschat model [12]	iSLIP [31]
	WCRT	BCRT		
Γ_1	8 cycles	6 cycles	70 cycles (aprox.)	12.5 cycles (aprox.)

with Equations 4 and 5 (with $L_R = 1$ cycle). For example, in the calculus of the worst-case traversal time of flow 1, its best-case time is augmented by 2 due to the interference with flows 2 and 3.

Finally, the message of each flow triggers the activation of the final task with a worst-case execution time of 0 cycles.

4.4. Response times comparison

Table 1 allows us to compare the worst-case response times obtained with our model (based on MAST elements) with the analysis proposed in [12] and the values obtained with an iSLIP algorithm proposed in [31] that is used in [12] as reference value. Values obtained from [12] have been divided by 4 to be comparable with our concept of "cycle". The results from [12] are approximate, since the paper only provides a graph, and does not provide exact numerical values. We can also obtain the WCRT and the BCRT of the rest of the flows of the system.

The pessimism introduced in the results from [12] when compared to our results is partly due to the consideration of the back-pressure effects. We avoid considering the back-pressure in our analysis by checking that the analyzed system does not exceed the maximum rate restriction at any of its links.

We can conclude that our analysis is much less pessimistic, obtaining shorter response times even when compared with the ISLIP algorithm [31] which assumes infinite buffers.

5. Applying the model to the epiphany processor

This section describes the application of the model described in Section 3 to the *Epiphany* processor [32].

5.1. Epiphany many-core

The *Epiphany III* processor is a many-core with 16 cores connected by a NoC placed in a 4×4 2D mesh as Fig. 2 shows, where every square is a tile that contains the router connected to the links connecting to the neighbor tiles and the execution core of the tile itself. Each core of the *Epiphany* is an *eCore* that executes its instructions in order, with a frequency of 600 MHz. Each core has 32 KB of local memory. The architecture is supported by GCC and has libraries for OpenMP and MPI.

The routers use round-robin arbitration to dispatch the packets arriving at them. Only the packets that share the destination link may get blocked in the router. As there are five input links, in a worst-case scenario all five input links would simultaneously have a packet with the same output link destination, and by the round-robin arbitration one of the packets would have to wait for the other four to be transmitted.

The NoC implements an XY routing algorithm for the packets, which means that the packets travel in first place through the columns of the mesh and only after they reach the destination column they start traveling through the rows.

A shared memory of 1 GB 32-bit wide DDR3L SDRAM is located on the East side of the *Epiphany* mesh as show in Fig. 2. This shared memory is not modeled in this paper.

The design could grow as it has been shown with a 1024 cores version [33]. Unfortunately, the *Epiphany V* is not available in any development board.

The inside of a tile is shown in Fig. 3, with the connections to the routers of the neighbor tiles. The routers are connected by full-duplex bidirectional links. There is a buffer per link of a single packet capacity as it has been shown at Fig. 6. Any packet that wants to use a busy link will be stopped and will have to wait there until the link is available and the router chooses it.

The router chooses a packet (if any is available) for every free output link. If the link corresponding to the packet's route is busy the router stops the packet and blocks the previous link in the route. This could lead to a core stall if the link blocked is the one directly connected to the *eCore* that is generating the message.

An *Epiphany* core sees the local memory of the rest of cores mapped as its own memory (with a special address).

The *Epiphany* processor is integrated into the *Parallella* development board [34], which has the size of a credit card and needs just 5 W to work. Apart from the *Epiphany* processor, the *Parallella* board also has a Zynq ARM dual-core processor that loads the code that will be executed to every *Epiphany* core and starts the execution of each core.

5.2. Basic parameters of the Epiphany NoC

The NoC works at the same frequency as the *eCores*, consequently for this NoC, $F_{NoC} = 600$ MHz. Hereafter we will use the term "cycle" to refer to both NoC or processor cycles (1 cycle $= \frac{1}{6.0e8} s \simeq 1.667$ ns).

The *Epiphany* processor has three independent 2D meshes as can be seen in Fig. 11.

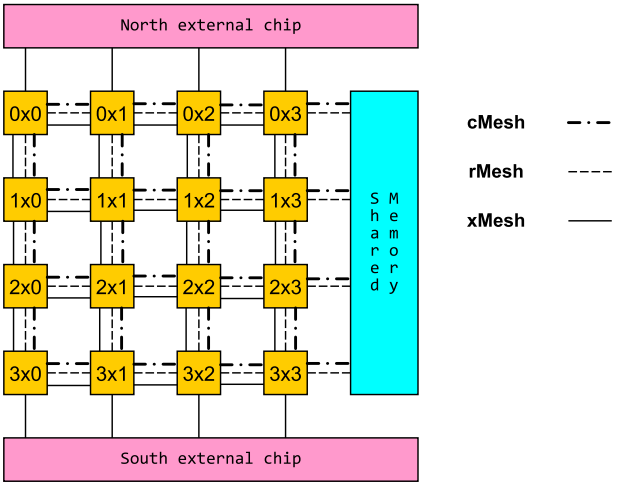

Fig. 11. Epiphany's 2D mesh NoCs.

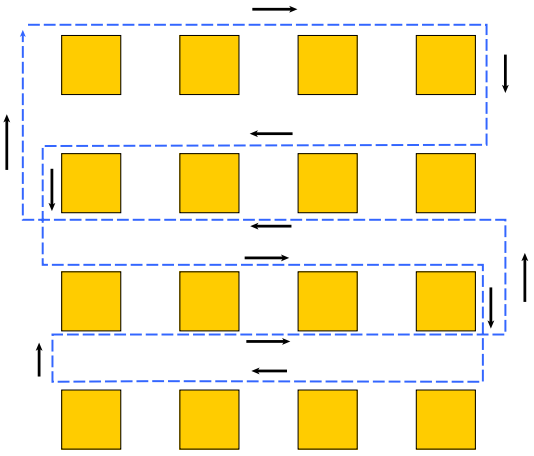

Fig. 12. Message direction in the test example for the evaluation of the maximum rate restriction.

- *cMesh*: used to write packets. The router arbitration on this mesh could cause a delay of one cycle ($L_{R_w} = 1$ cycle). The router can route a 64-bits packet per cycle and output link.
- *rMesh*: used for read requests. The router arbitration on this mesh could cause a delay of eight cycles ($L_{R_r} = 8$ cycles). The router can route a 64-bits packet per cycle and output link. Read requests are always one packet long.
- *xMesh*: used to write packets destined for off-chip resources or for another chip in a multi-chip system configuration. These features are not modeled in this paper.

A packet traveling by *cMesh* or *rMesh* traverses the network, with a latency of 1.5 cycles per routing hop ($L_H = 1.5$ cycles).

Both NoCs, *cMesh* and *rMesh*, have round-robin routing. Besides, both networks are independent so the traffic of one NoC does not affect the other NoC.

5.3. Modeling the messages

The cores in *Epiphany* can communicate by writing and reading on the local memory of a remote core, which leads to two kinds of messages in the NoC. A write operation requires a write message which travels through the *cMesh* from the writing core to the target tile. For its part, a read operation requires two messages: a read request that travels through the *rMesh* from the reading core to the target tile and a write-back message that returns through the *cMesh* from the target tile to the reading core. The target tile's core is not involved in the operation, as the router is able to access the local memory directly. Both, the read request and the write-back message, are one packet (64 bits) long. When a core wants to read a remote memory area larger than 64 bits, the operation is split in a number of 64-bits read requests (each of them with its corresponding 64-bits write-back message) that are executed one after the other (the read request m_{ijk} will not be generated until the write-back message $m_{ij(k-1)}$ has arrived).

From the aforementioned behavior it can be deduced that, for long read operations, the generation rate of the read request packets is the inverse of the number of cycles (c) between the generation of two consecutive read requests plus the best-case traversal times of the read request packet and its corresponding write-back packet (Eq. (7)). Similarly, Eq. (8) can be used to calculate the generation rate of the write-back packets corresponding to a long read operation.

$$\forall m_{ijk} : t_{ijk} = read, \rho_{ijk} = \frac{1}{TT^b_{ijk} + TT^b_{ij(k+1)} + c} \tag{7}$$

$$\forall m_{ijk} : t_{ijk} = write - back, \rho_{ijk} = \frac{1}{TT^b_{ijk} + TT^b_{ij(k-1)} + c} \tag{8}$$

Regarding write messages, the compiler transforms long write operations into calls to the `memcopy` C standard function. By analyzing the assembler code of a `memcopy` function we have found that it is able to generate one packet on the NoC every three cycles. Another way to write consecutive packets is to perform consecutive variable assignments. By analyzing the assembler code of multiple variables assignment instructions we have found that this generates one packet on the NoC every two cycles. This 1/2 value is the fastest packet generation rate that standard user's code is able to produce on the *cMesh*, and it will be used only when writing variable after variable.

As we mentioned above, the fixed routing of the Epiphany processor is XY routing, which means that any packet will take the same route over and over again for the same source and destination cores, independently of the traffic at the NoC. So, a message m_{ijk} between the cores PR_{xy} and $PR_{x'y'}$ performs the deterministic number of hops obtained in Eq. (9) (notice that this value also coincides with the number of traversed routers, including the source core's router).

$$H_{ijk} = |x - x'| + |y - y'| + 1 \tag{9}$$

5.4. Maximum packet rate restriction

Since the *Epiphany*'s meshes are independent, the maximum rate restriction is calculated for each of them. That is, when Eq. (3) is applied to a link in the *cMesh*, only write (and write-back) messages are considered. Likewise, only read messages are considered when Eq. (3) is applied to a link in the *rMesh*. In order to verify the maximum packet rate restriction every link in the *cMesh* must have an accumulated transmission rate not higher than $\frac{1}{L_{R_w}}$ and every link in the *rMesh* must have an accumulated transmission rate not higher than $\frac{1}{L_{R_r}}$.

In order to show that this limitation is not very restrictive in the *Epiphany* many-core, let us consider an example situation with the following characteristics:

- For simplicity we only consider write messages (the type of messages most frequently used by the applications).
- All the messages are generated at a 1/3 rate using the `memcopy` standard C function.
- Messages, on average, go through three routers of the NoC before reaching their destination.

In such situation, and considering that the 4 × 4 Epiphany mesh hash 48 links, the system could be able of supporting an application with up to 48 messages being sent per NoC cycle without breaking the maximum packet rate restriction. Moreover, note that the actual

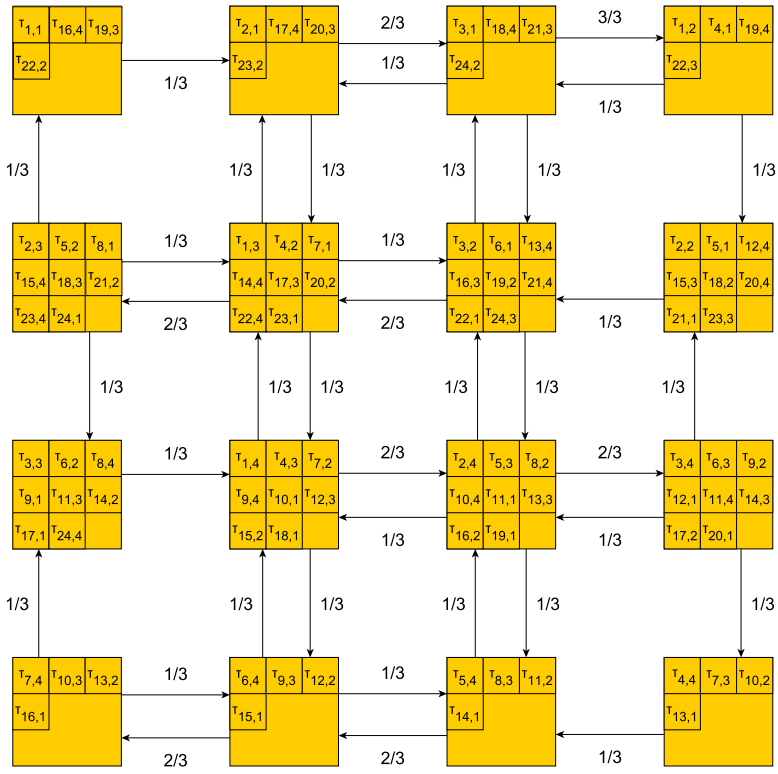

Fig. 13. Test example for the evaluation of the maximum rate restriction.

number of messages could be much longer since, as stated in Section 3.5, when two or more messages coming from the same core share a link, only the message with higher generation rate is included in the calculation of the accumulated transmission rate of the link (the reason is that messages coming from the same core cannot be generated at the same time and, therefore, they cannot compete among them).

To endorse the statement of the low impact of the rate restriction, let us consider an example distribution of e2e flows that maximizes the traffic on the NoC. In this example, e2e flows are made up of four task with a distance of three hops between tasks. The tasks in the flows are distributed along the lines shown in Fig. 12, each flow starting in correlative cores.

As can be seen in Fig. 13, 24 flows (96 tasks) can be placed following the aforementioned distribution without breaking the maximum rate restriction. Note that although the distribution of the e2e flows shown in Fig. 12 has been designed to accommodate a high load of tasks and messages, this is not an optimal distribution an, in fact, most links are below the maximum rate. The development of an allocation algorithm is out of the scope of this paper and is left as future work.

This example shows that the maximum rate restriction is not a significant limitation even in a system with a high computational and communications load.

5.5. Packet maximum traversal times

As a consequence of the *Epiphany*'s NoC round-robin arbitration the biggest possible delay suffered by a packet happens when it is waiting for the rest of the inputs links of the router to be routed to the same output link. Therefore, it has to wait for 4 packets at most.

To calculate the worst-case traversal time Eq. (5) is used with the following variables:

- $L_H = 1.5$ cycles, which is the latency per hop in the Epiphany processor.
- H_{ijk}, which is the number of routers the message has to go through between the source and the destination core obtained using Eq. (9).
- I_{ijk}, which is the packet interference calculated with Eq. (4). For the *Epiphany* processor it is transformed into Eq. (10) if it is a read request or into Eq. (11) if it is a write or a write-back message.

$$I_{ijk} = \sum_{r_{xy} \in route_{ij}} nb_{ijkxy} \cdot L_{R_r} = \sum_{r_{xy} \in route_{ij}} nb_{ijkxy} \cdot 8 \tag{10}$$

$$I_{ijk} = \sum_{r_{xy} \in route_{ij}} nb_{ijkxy} \cdot L_{R_w} = \sum_{r_{xy} \in route_{ij}} nb_{ijkxy} \cdot 1 \tag{11}$$

6. Simple example

This section will guide the process of modeling the example system shown in Fig. 14 formed by two *e2e* flows having three tasks each. The tasks are scheduled under a fixed-priority non-preemptive scheduling policy with the objective of having an example closer to the behavior of M2OS-mc, which is the operating system used in our implementation. M2OS-mc [26] is a real-time operating system designed for mesh based many-core using a microkernel approximation. The model allows specifying preemptive scheduling policies also. Every message in this system is a write message as it is the simplest way to implement an *e2e* flow on the *Epiphany* processor. The parameters of the example system are shown in Table 2. The rates of all the messages are $\frac{1}{3}$ cycle^{-1} since we are assuming they are generated by the `memcopy` C standard function that, as it was previously stated, is able to generate a packet every three cycles.

Before being able to execute any MAST analysis we need to know if the maximum rate limitation is fulfilled for our system ($P_{xy \to x'y'} \leq \frac{1}{L_{R_W}}$, for every link in the NoC). The results of the rate study are shown in Fig. 15, which only displays links with rate higher than zero.

In Fig. 15, the rates of the links with only one message going through them (only one arrow) have an accumulated rate of 0.33 cycle^{-1}. The link from 0 × 1 to 1 × 1 has a rate of 0.66 cycle^{-1} as

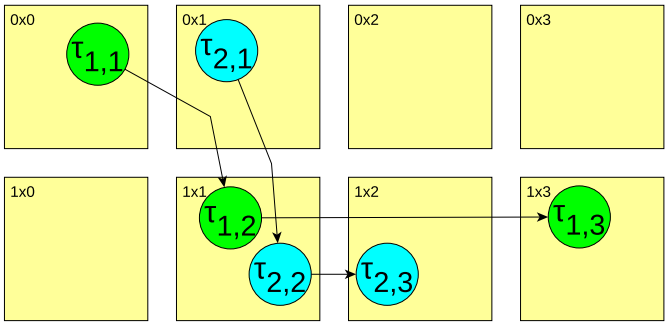

Fig. 14. Example system formed by two *e2e* flows with three tasks each.

Table 2
Parameters of the example system.

Γ_1	$(T_1 = 50\ \mu s \quad D_1 = 50\ \mu s)$
τ_{11}	$C_{11} = 5\ \mu s\ \ C^b_{11} = 4\ \mu s\ \ Prio_{11} = 3$
m_{111}	$\mu_{111} = 2\ \ \rho_{111} = 0.33\ \text{cycle}^{-1}$
τ_{12}	$C_{12} = 3\ \mu s\ \ C^b_{12} = 2\ \mu s\ \ Prio_{12} = 3$
m_{121}	$\mu_{121} = 1\ \ \rho_{121} = 0.33\ \text{cycle}^{-1}$
τ_{13}	$C_{13} = 7\ \mu s\ \ C^b_{12} = 6\ \mu s\ \ Prio_{13} = 3$
Γ_2	$(T_1 = 160\ \mu s \quad D_1 = 160\ \mu s)$
τ_{21}	$C_{21} = 13\ \mu s\ \ C^b_{21} = 12\ \mu s\ \ Prio_{21} = 2$
m_{211}	$\mu_{211} = 4\ \ \rho_{211} = 0.33\ \text{cycle}^{-1}$
τ_{22}	$C_{22} = 11\ \mu s\ \ C^b_{22} = 10\ \mu s\ \ Prio_{22} = 2$
m_{121}	$\mu_{121} = 5\ \ \rho_{121} = 0.33\ \text{cycle}^{-1}$
τ_{23}	$C_{23} = 17\ \mu s\ \ C^b_{23} = 16\ \mu s\ \ Prio_{23} = 2$

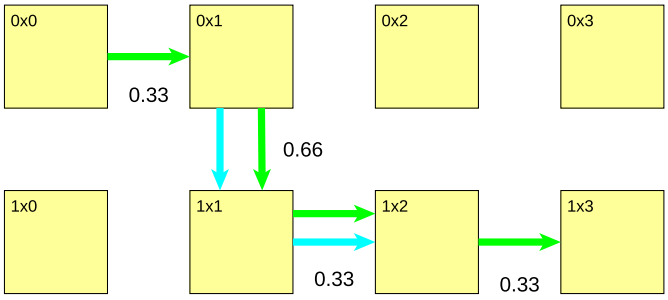

Fig. 15. Accumulated transmission rate for the links of the example system (units in cycle^{-1}).

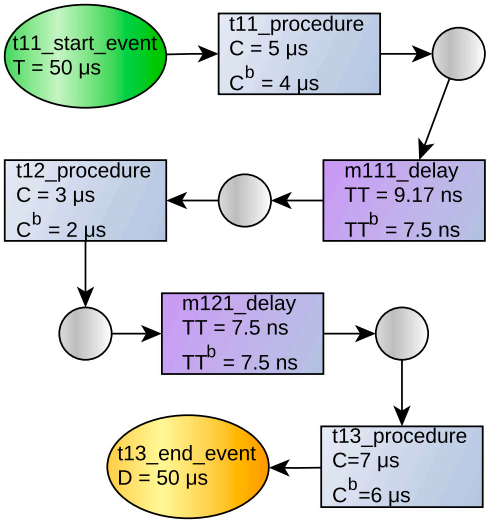

Fig. 16. MAST model for Γ_1.

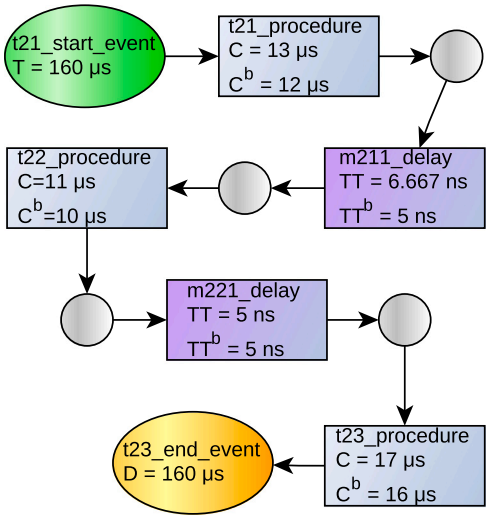

Fig. 17. MAST model for Γ_2.

two messages could use the link. In contrast, although the link between 1×1 and 1×2 is also used by two messages, its accumulated rate is 0.33 cycle^{-1} because both messages are generated from tasks in the same core and, in consequence, they cannot be produced concurrently. The maximum rate limitation is verified as every NoC link has an accumulated rate less than $\frac{1}{L_{R_w}} = 1\ \text{cycle}^{-1}$.

To convert a many-core system into a MAST model we will focus on one of the flows at a time. For each task of the *e2e* flow we perform the following loop in a sequential order (starting from the first task in the flow, following the flow order and finishing at the last task).

The conversion performed for each of the tasks is:

1. The task is transformed into a MAST step. We consider the message generation to be part of the execution time of the step. This step is executed at the processing resource representing the core the task is mapped to.
2. If there is a following task on the *e2e* flow, and that task is on another PR, a delay block must be created to model the message traversal time through the NoC. The best and worst-case traversal times assigned to the delay are calculated using Eq. (2) and Eq. (5) respectively.
3. If there is not a following task on the *e2e* flow this means that the *e2e* modeling has finished and no further action is needed for the *e2e* flow.

Once the MAST file of the system model is generated, any of the MAST offset-based analysis tools can be applied.

The *e2e* flow modeled for Γ_1 is shown in Fig. 16 and the model for Γ_2 is shown in Fig. 17. In both figures we can see the events and activities that model both flows. The internal events that connect each step or delay with the next are represented with big dots.

The first item of the modeled transactions are the events that will trigger the first step of each *e2e* flow and have a period determined for each flow.

The steps, which model the tasks, contain the measured or estimated worst-case execution time C and also the best-case execution time C^b for the task execution. These times include the time required to generate the message.

The best-case traversal time for the delays is calculated using Eq. (2). For example, in the case of message m_{111} that goes through three routers, this value is obtained as follows:

$$TT^b_{111} = 1.5 cycles \cdot 3 = 4.5 cycles = 7.5\ \text{ns}$$

Table 3
Results of the MAST analysis.

Γ_1 flow		
t1_event_msg	$R^b = 4,000$ ns	$R = 5,000$ ns
t2_event_msg	$R^b = 6,009$ ns	$R = 19,010$ ns
t3_event_2	$R^b = 12,017$ ns	$R = 26,019$ ns
Γ_2 flow		
t4_event_msg	$R^b = 12,000$ ns	$R = 13,000$ ns
t5_event_msg	$R^b = 22,005$ ns	$R = 27,007$ ns
t6_event_2	$R^b = 38,010$ ns	$R = 44,012$ ns

Eq. (11) is used to calculate the interference suffered by each packet of a message. For example, message m_{111} has one competitor message, m_{211}, that shares the south output link of router 0 × 1 with m_{111}. In consequence the interference term of m_{111} takes the value:

$$I_{111} = nb_{111_{01}} * L_{R_w} = 1 \cdot 1cycle = 1cycle = 1.67 \text{ ns}$$

According to Eq. (5), the worst-case traversal time of the delay used to model message m_{111} is calculated as:

$$TT_{111} = TT^b_{111} + I_{111} = 5.5cycles = 9.17 \text{ ns}$$

The rest of calculations are pretty similar or without any interference to take into account for the delay block.

Internal events concatenate activities until the last event is reached. The worst-case response time of the end event is compared with the deadline of the last step.

The execution of the offset-based MAST analysis for both flows is shown in Table 3. All the response times are measured from the event activated by the period. The last events of both flows have a worst-case response time lower than the deadline of the respective flows, meaning that the system is schedulable.

7. High-level communication mechanisms

The messages between the *Epiphany* cores explained in Section 5 are not synchronized messages. However, it is usual to require that the data shared among tasks is accessed in a mutually exclusive manner. In this section we present the modeling of two synchronization mechanisms, the sampling ports and the queuing ports, inspired in the namesake communication ports defined in the ARINC-653 standard [35]. Both mechanisms have been implemented in the M2OS-mc operating system for the Epiphany many-core [26,27].

A Sampling Port (SP) enables reading and writing a simple instance of a data element. A writing operation on an SP overwrites the existing data. A reading operation returns the last written data. There is a mechanism that allows the reader to know if the data item has been read for the very first time. SPs can be used when the periods of readers and writers are different, when having one producer and multiple consumers or when the applications are interested only in the most recent data.

A Queuing Port (QP) enables reading and writing a series of in-order data elements. A QP is based on a circular FIFO queue. QPs can be used when the periods of readers and writers are equal on average and when the application does not want to miss any data. They are not suitable when the periods of the writers are smaller than the periods of the reader as this will end up in the QP missing messages. When a task tries to write to a full QP, the operation finishes immediately with an error notification. When a task tries to read from an empty QP the reading task is blocked and will be awakened when a data item arrives in the QP.

The operations on an SP or a QP are implemented as critical sections protected by a multiprocessor mutual exclusion synchronization primitive similar to a spinlock.

A more detailed explanation of the SP and QP implementation in M2OS-mc can be found at [26,27].

7.1. Epiphany's mutex primitive

The implementation of the sampling and queuing ports requires the use of a mutual exclusion synchronization mechanism. For this purpose, Epiphany provides the "mutex" primitive. An *Epiphany* mutex is a kind of spinlock that allows synchronizing two or more cores: a core trying to lock an already locked mutex will remain stalled until the mutex is unlocked. A mutex is implemented using an integer number located in the local memory of one of the cores that is accessed using the mutex operations provided by the Epiphany Hardware Utility Library (eLib). The atomicity of the lock and unlock operations on a mutex is granted by the NoC control hardware.

In order to lock the mutex a core issues a special read request message that reads the value of the mutex and, in case its value is zero, it atomically changes its value to the core ID of the core executing the lock, consequently locking the mutex. If the value is not zero then the mutex was locked by some other eCore and the read operation blocks until the mutex is unlocked. The special read request is followed by a write-back operation that writes the current value of the mutex in the local memory of the calling core. When this returned value is obtained the application issuing the call knows that the mutex is owned by itself and can continue its execution. In summary, locking the mutex is a blocking operation. During the time the application is blocked the eCore is stalled and, therefore, this blocking time is modeled as execution time.

The unlock operation simply consists in writing a zero on the mutex. In consequence, it only requires a write message from the core unlocking the mutex to the local memory of the core holding the mutex.

7.2. Modeling sampling ports

In order to model the sampling ports, we need to identify the messages generated during the execution of its write and read operations. It is important to notice that those messages will only be generated if the SP is located in the local memory of a core different from the one performing the operation.

The identified messages will affect the model in the following three aspects: (a) calculate the accumulated rate of the links of the NoC, (b) increment the worst-case execution time of the tasks by the interference suffered by the read and write-back messages and by the blocking time due to the mutually exclusive use of the SP, and (c) include a delay block to model the traversal time of the last packet on the NoC.

The pseudocode of an SP write operation is shown in Listing 1, showing also the read (R) and write (W) messages generated for each instruction. As described in Section 7.1, locking a mutex involves a read message (along with its associated write-back message). The rest of the instructions in Listing 1 will generate write messages. Therefore, in order to consider the worst-case situation, the worst-case execution time measured in isolation of a task that writes on an SP located in the local memory of another core should be incremented due to the interference suffered by the read and write-back messages according to Eq. (6) (with the interference calculated using Eq. (10) and Eq. (11)).

Regarding the maximum rate limitation, the read message (only one packet) will have the rate calculated with Eq. (1). Among the write messages the one with the highest rate is the message generated to write the data, in case the data is longer than one packet (64 bits). In such situation, the compiler will use the `memcopy` function which leads to a packet generation rate of $\frac{1}{3}$ cycle^{-1}. In case the data fits in just one packet, we have measured a packet generation rate for the three write messages in Listing 1 of $\frac{1}{7}$ cycle^{-1}.

Listing 1: Pseudocode of the write operation on an SP.

```
Lock the mutex          (R)
Write the data          (W)
Set the data as new     (W)
Unlock the mutex        (W)
```

The pseudocode of an SP read operation is shown in Listing 2. Likewise for the write operation, the worst-case execution time of the calling task must be incremented by the interference time suffered by the read messages (and their corresponding write-back messages). The only difference in this case is that the number of read messages depends on the size of the data to read.

Regarding the maximum rate limitation, the rate of the read messages is calculated with Eq. (7). Inspecting the assembler code generated by the compiler we can deduce that the lowest number of instructions between two consecutive read messages (parameter c in Eq. (7)) is 25 cycles. The write message to unlock the mutex is one packet long so, its rate will be calculated using Eq. (1). The rate of the write-back messages generated as a response of the read requests is calculated using Eq. (8) with the same value of c than for the read messages (25 cycles).

Listing 2: Pseudocode of the read operation on an SP.

```
Lock the mutex                    (R)
Read the data                     (R)
Read if the data is new           (R)
Unlock the mutex                  (W)
```

Modeling the write and read operations on an SP also requires including a delay element with the best and worst-case traversal times for the write messages generated to unlock the mutex (last instructions in Listing 1 and Listing 2). The best and worst traversal times of this message will be calculated using Eq. (2) and Eq. (5), respectively.

Finally, it is necessary to model the blocking time due to the mutually exclusive use of the SP. In the linear *e2e* flows we are modeling, each non-terminal task in an *e2e* flow, τ_{ij}, reads from a port shared with the previous task in the flow, $\tau_{i(j-1)}$, and writes in a port shared with the following task, $\tau_{i(j+1)}$. In consequence, τ_{ij} will suffer a blocking time equal to the worst case duration of the write operation of the previous task, $B_{wi(j-1)}$, plus the worst case duration of the read operation of the following task, $B_{ri(j+1)}$. These blocking values can be measured in isolation and, in order to consider the worst-case situation, augmented with the interference times of all the generated read and write-back messages (calculated using Eq. (10) and Eq. (10)). Given that the processor is stalled during this blocking time, the worst-case execution times of the task (measured in isolation) should be augmented with the value of the blocking times to contemplate the worst-case scenario.

To obtain the final worst-case execution time of the task, the interference times of the read and write-back messages must also be added to the worst-case execution time (measured in isolation). The effective execution time C'_{ij} of a non-terminal task τ_{ij} will be estimated with Eq. (12) (notice that terms $B_{wi(j-1)}$ and $B_{ri(j+1)}$ should not be included for the first and last tasks in a flow, respectively).

$$C'_{ij} = C_{ij} + \sum_{\forall k : t_{ijk}=read} (I_{ijk} + I_{ij(k+1)}) + B_{wi(j-1)} + B_{ri(j-1)} \tag{12}$$

An example of modeling two tasks using an SP is presented in Section 8.1.

7.3. Modeling queuing ports

Modeling the write and read operations on a queuing port, whose pseudocodes are shown in Listing 3 and Listing 4 respectively, is very similar to the sampling port modeling performed in the previous section.

The worst-case execution time of the tasks using a QP (typically measured in isolation) must be augmented by the interference times of all the read and write-back messages according to Eq. (6) (with the interference calculated using Eq. (10) and Eq. (11)).

We also need to obtain the packet generation rates of the messages. Considering the write messages in the QP write operation (Listing 3), the highest rate corresponds to the message generated to write the data in case the data is longer than one packet. In such situation, the compiler will use the `memcopy` function which leads to a packet generation rate of $\frac{1}{3}$ cycles^{-1}. In case the data fits in just one packet, we have measured a global packet generation rate for the packets of the three write messages of $\frac{1}{8}$ cycles^{-1}. The maximum generation rate of the three read messages in Listing 3 can be obtained by applying Eq. (7) with a value of 25 cycles for the c parameter (the value of c has been obtained by inspecting the assembler code generated by the compiler and measuring the delay of the remote router while reading the data and generating the write-back message). Likewise we can calculate the maximum generation rate of the write-back messages using Eq. (8).

Regarding the QP read operation (Listing 4), we have measured a packet generation rate for the three write messages (one packet each) of $\frac{1}{4}$ cycles^{-1}. For their part, the maximum generation rate of the three read messages is calculated using Eq. (7) with a value for the c parameter of 25 cycles when the data to read is longer than one packet and of 44 cycles when it fits in one packet. Likewise, we can calculate the maximum generation rate of the write-back messages using Eq. (8).

Listing 3: Pseudocode of the write operation on a QP

```
Lock the mutex                    (R)
If QP is not full                 (R)
  Allocate the QP slot            (R&W)
  Write the data                  (W)
Else
  Set "full QP" error
End if
Unlock the mutex                  (W)
```

Listing 4: Pseudocode of the read operation on a QP

```
Lock the mutex                    (R)
If QP is not empty                (R)
  Get QP position                 (R)
  Read the data                   (R)
  Free the  QP slot               (W)
  Unlock the mutex                (W)
Else
  Unlock the mutex                (W)
  Block the task
End if
```

As for the SPs, a delay must be included to model the traversal time of the write message generated to unlock the mutex (last instruction in Listing 3 and Listing 4). The best and worst traversal times of this message will be calculated using Eq. (2) and Eq. (5), respectively.

We also have to model the blocking time due to the mutually exclusive use of the QP. As in the case of the SPs, the longest operation on a QP could be measured in isolation and should be augmented with the interference times of all the read and write-back messages generated (calculated using Eq. (10) and Eq. (11)). The worst-case execution times of the tasks that use the QP should be augmented with the value of the calculated blocking time to contemplate the worst-case scenario as shown in Eq. (12).

An example of modeling a complex system using QPs is presented in Section 8.2.

8. Modeling with ports

The algorithm used to obtain the MAST model of an application with one or more *e2e* flows using SPs or QPs to communicate the tasks is shown in Listings 5, 6 and 7. The input data to the algorithm are the *e2e* flows with their steps mapped to cores and with the port type (SP or QP) used to synchronize each pair of consecutive steps.

First of all, in Listing 5, the algorithm generates the messages corresponding to the ports used inside the *e2e* flow. After that, in

Listing 5: Part 1 of 3 of the pseudocode used to obtain the MAST model of an application

```
1  — Create messages
2  for each step τ_ij
3    — Every message has the same source and destination, so same TT^b
4    TT^b_ij = L_H · |τ_ij.PR.x − τ_i(j+1).PR.x| + |τ_ij.PR.y − τ_i(j+1).PR.y| + 1
5    if step τ_ij writes on an SP
6      — Create the messages identified in Section 6.2
7      — Message tuple: m_ijk = {ρ_ijk, μ_ijk, t_ijk, d_ijk}
8      m_ij1 = {1/(TT^b_ij+TT^b_ij+c), 1, read, τ_i(j+1).PR}
9      m_ij2 = {1/(TT^b_ij+TT^b_ij+c), 1, write—back, τ_i(j+1).PR}
10     if μ_ij > 1
11       ρ_ij3 = 1/3
12     else
13       ρ_ij3 = 1/8
14     end if
15     m_ij3 = {ρ_ij3, μ_ij, write, τ_ij(j+1).PR}
16     m_ij4 = {1/8, 1, write, τ_i(j+1).PR}
17     m_ij5 = {1/8, 1, write, τ_i(j+1).PR}
18   end if
19   if step τ_ij writes on a QP
20     — Create the messages identified in Section 6.3
21     — Message tuple: m_ijk = {ρ_ijk, μ_ijk, t_ijk, d_ijk}
22     m_ij1 = {1/(TT^b_ij+TT^b_ij+c), 1, read, τ_i(j+1).PR}
23     m_ij2 = {1/(TT^b_ij+TT^b_ij+c), 1, write—back, τ_i(j+1).PR}
24     m_ij3 = {1/8, 1, write, τ_i(j+1).PR}
25     m_ij4 = {1/(TT^b_ij+TT^b_ij+c), 1, read, τ_i(j+1).PR}
26     m_ij5 = {1/(TT^b_ij+TT^b_ij+c), 1, write—back, τ_i(j+1).PR}
27     m_ij6 = {1/(TT^b_ij+TT^b_ij+c), 1, read, τ_i(j+1).PR}
28     m_ij7 = {1/(TT^b_ij+TT^b_ij+c), 1, write—back, τ_i(j+1).PR}
29     if μ_ij > 1
30       ρ_ij8 = 1/3
31     else
32       ρ_ij8 = 1/8
33     end if
34     m_ij8 = {ρ_ij8, μ_ij, write, τ_i(j+1).PR}
35     m_ij9 = {1/8, 1, write, τ_i(j+1).PR}
36   end if
37 end for
```

Listing 6, the algorithm checks the link rates of the whole system. With that purpose the highest rate possible per link and orthogonal NoC is accumulated using Eq. (3) as it is shown in lines 5 to 13 in Listing 6. Once every rate has been obtained, we check the maximum rate restriction with the maximum rate of the system ($\frac{1}{L_R}$), which in the case or Epiphany has to be done for each of the two orthogonal NoCs: the cMesh accumulating the write and response messages with a maximum rate of $\frac{1}{L_{R_w}}$ and the rMesh accumulating the read messages with a maximum rate of $\frac{1}{L_{R_r}}$. This is shown in lines 23 to 30 in Listing 6. Only the message with the highest rate generated from a given PR is considered, as only one activity will be executing in the same PR at a given time. This is shown in lines 14 to 19 in Listing 6.

The last message of a communication mechanism operation is a write message unlocking the spin lock. The BCTT of that last message is obtained with Eq. (2) using $L_H = 1.5$ and H as in Eq. (9) as done in line 4 in Listing 5, while the network interference to a message is obtained with Eq. (4) that will be transformed into Eq. (11) for write messages as shown in lines 2 to 6 in Listing 7.

The network interference and the mutual exclusion access to the ports could impact the execution time of the activities, which should be reflected in our timing analysis. The impact of the interference on the execution of the read and write-back messages (lines 33 to 44 in Listing 6) and the accesses to the ports (lines 9 to 21 in Listing 7) is shown in Eq. (12).

In the following subsections we present two modeling examples to illustrate how the algorithm works.

8.1. Sampling port modeling example

We are going to show an example of a typical sampling port application where a task needs to read information coming from another core to perform some calculations. A graphical representation of the example application is shown in Fig. 18, while the parameters can be seen in Table 4.

Task τ_{11} updates the data formed by two packets stored in the sampling port located at core 0 × 1. Task τ_{12} will perform a read operation over that local sampling port with a given polling period

Listing 6: Part 2 of 3 of the pseudocode used to obtain the MAST model of an application

```
1  -- Calculate links ratio
2  for each PR
3    max_ρ_r (every_link) = 0
4    max_ρ_w (every_link) = 0
5    for each message m_ijk sent from PR_xy
6      for each link l_xy→x'y' traversed by m_ijk
7        if t_ijk = read
8          max_ρ_r (l_xy→x'y') = max(max_ρ_r(l_xy→x'y'), ρ_ijk)
9        else -- t_ijk = write or write-back
10         max_ρ_w (l_xy→x'y') = max(max_ρ_w(l_xy→x'y'), ρ_ijk)
11       end if
12     end for
13   end for
14   for each read link l_xy→x'y'
15     ρ_r (l_xy→x'y') += max_ρ_r (l_xy→x'y')
16   end for
17   for each write link l_xy→x'y'
18     ρ_w (l_xy→x'y') += max_ρ_w (l_xy→x'y')
19   end for
20 end for
21
22 -- Check maximum ratio restriction
23 for each link l_xy→x'y'
24   if ρ_r(l_xy→x'y') > 1/L_R_r
25     exit (not analyzable)
26   end if
27   if ρ_w(l_xy→x'y') > 1/L_R_w
28     exit (not analyzable)
29   end if
30 end for
31
32 -- Calculate message interferences
33 for each message m_ijk
34   nb_ijk = 0
35   for each router r_xy traversed by m_ijk
36     nb_ijkxy = number of buffers of r_xy sharing the output link with m_ijk
37     nb_ijk += nb_ijkxy
38   end for
39   if t_ijk = read
40     I_ijk = nb_ijk · L_R_r
41   else
42     I_ijk = nb_ijk · L_R_w
43   end if
44 end for
```

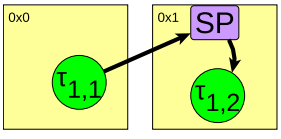

Fig. 18. Diagram of sampling port example.

Table 4
Parameters for the sampling port example.

Γ_1	($T_1 = 1$ ms $D_1 = 1$ ms)
τ_{11}	$C_{11} = 5$ µs $C^b_{11} = 4$ µs $C'_{11} = 5.43$ µs $Prio_{11} = 3$
m_{111}	$\mu_{111} = 1$ $t_{111} = read$ $\rho_{111} = 1.67$ µcycles^{-1}
m_{112}	$\mu_{112} = 1$ $t_{112} = write - back$ $\rho_{112} = 1.67$ µcycles^{-1}
m_{113}	$\mu_{113} = 2$ $t_{113} = write$ $\rho_{113} = 0.33$ cycles^{-1}
m_{114}	$\mu_{114} = 1$ $t_{114} = write$ $\rho_{114} = 0.14$ cycles^{-1}
m_{115}	$\mu_{115} = 1$ $t_{115} = write$ $\rho_{115} = 0.14$ cycles^{-1}
τ_{12}	$C_{12} = 3$ µs $C^b_{12} = 2$ µs $C'_{12} = 3.35$ µs $Prio_{11} = 3$

T_{poll} = 1.5 µs. If the data is new, τ_{12} will execute its code for an execution time of C_{12}.

The messages in Table 4 correspond to the ones identified for a write operation on an SP shown in Listing 1 and created by the algorithm shown in Listing 5. The read and write-back messages (m_{111} and m_{112} respectively) are one packet long and, in consequence, their packet generation rate is calculated using Eq. (1). The write message that writes the data, m_{113}, has the generation rate of the `memcopy` function ($\frac{1}{3}$ cycles^{-1}). For their part, the write messages to set the data as new and unlock the mutex are both one packet long and, taken as a group, they have a generation rate of $\frac{1}{8}$ cycles^{-1}. The rate of the NoC links is shown in Fig. 19, where we can clearly see that they all verify the maximum rate restriction.

Listing 7: Part 3 of 3 of the pseudocode used to obtain the MAST model of an application

```
1  — Create delays
2  for each message m_ijk that is last message of step τ_ij
3     TT_ijk = TT^b_ij + I_ijk
4     delay τ_ij = TT_ijk
5     Create delay (TT_ijk, TT^b_ijk) placing it after τ_ij
6  end for
7
8  — Increment step WCETs
9  for each τ_ij
10    for each m_ijk
11       if t_ijk = read or write—back
12          C_ij += I_ijk + I_ij(k+1)
13       end if
14    end for
15    if τ_ij is not the first step
16       C_ij += B_wi(j−1)
17    end if
18    if τ_ij is not the last step
19       C_ij += B_ri(j+1)
20    end if
21 end for
```

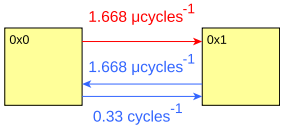

Fig. 19. Rates of the NoC links in the sampling port example.

Table 5
Results of the MAST model and the Epiphany executions for the Γ_1 flow of the sampling port example.

Step	MAST model results		Epiphany execution results		
	WCRT	BCRT	WCRT	Avg RT	BCRT
τ_{11}	5,430 ns	4,000 ns	5,208 ns	5,048 ns	4,968 ns
τ_{12}	8,985 ns	6,205 ns	8,703 ns	8,225 ns	7,795 ns

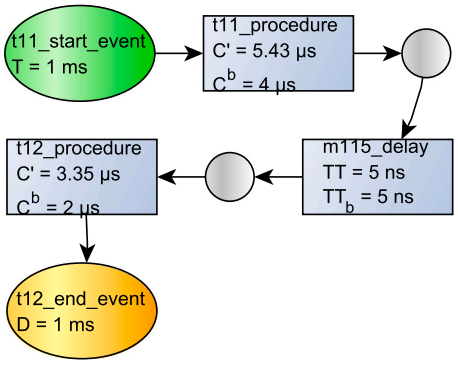

Fig. 20. MAST model for the sampling port example.

In this simple example, the read and write-back messages (m_{111} and m_{112}) do not suffer any interference from other packets so the worst-case execution time of τ_{11} does not have to be augmented by this reason. However, the measured execution times of both tasks must be augmented in the maximum blocking time due to the mutually exclusive use of the SP (lines 9 to 21 in Listing 7). In this case, both tasks perform a different operation: τ_{11} writes two packets to the SP, which is an operation that requires 346.67 ns, while τ_{12} reads two packets from the SP, which is an operation that requires 428.34 ns. In consequence, the measured times of τ_{11} and τ_{12} (C_{11} and C_{12}) have been incremented to the values $C'_{11} = 5.43$ µs and $C'_{12} = 3.35$ µs.

Besides, a delay element is added to model the traversal time through the NoC of the last packet of the last message (m_{115}), as shown in lines 2 to 6 in Listing 7. The best traversal time of this packet, TT^b_{115}, is calculated using Eq. (2) with $L_H = 1.5cycles = 2.5$ ns and $H_{115} = 2$ ns (see line 4 of Listing 5). Since in this simple example there is no interference, the worst-case traversal time, TT_{115}, is equal to TT^b_{115}.

Fig. 20 shows the resulting MAST elements, Table 5 shows the response times obtained using MAST and the response times of the *Epiphany* processor executing synthetic tasks with the same configuration. It can be seen that the measured execution is bounded by the MAST model worst-case results as they are lower and very close to the ones obtained by the analysis.

8.2. Queuing port example

Using queuing ports to implement *e2e* flows can be more appropriate in many cases since the blocking behavior of the read operations avoids the need for polling.

Fig. 21 shows a system with three *e2e* flows. The parameters of these flows are displayed in Tables 6 and 7. In order to show the modeling process we will focus on *e2e* flow Γ_2 and, in particular, on task τ_{22}. The messages m_{221} to m_{229} generated for τ_{22} correspond to the ones identified in Listing 3.

The read messages (m_{221}, m_{223} and m_{225}) go through three routers having a TT^b of 4.5 cycles (calculated using Eq. (2)) and, since they share a link with the read messages generated by τ_{11}, their TT is 12.5 cycles (according to Eq. (5), using Eq. (10) to obtain the interference term). The traversal times for the write-back messages (m_{222}, m_{224} and m_{226}) are calculated similarly (using Eq. (11), instead Eq. (10), to obtain the interference term), obtaining the values $TT^b = 4.5$ cycles and $TT = 5.5$ cycles. The traversal times for the write messages (m_{227}, m_{228} and m_{229}) are also calculated with Eq. (5) and Eq. (11), obtaining the values $TT^b = 4.5$ cycles and $TT = 5.5$ cycles. To serve as an example, the calculation of the traversal times of one packet of a read message (m_{221}), a write-back message (m_{222}) and a write message (m_{227}) are shown below:

$$TT^b_{221} = L_H \cdot H = 1.5 \cdot 3 = 4.5cycles$$

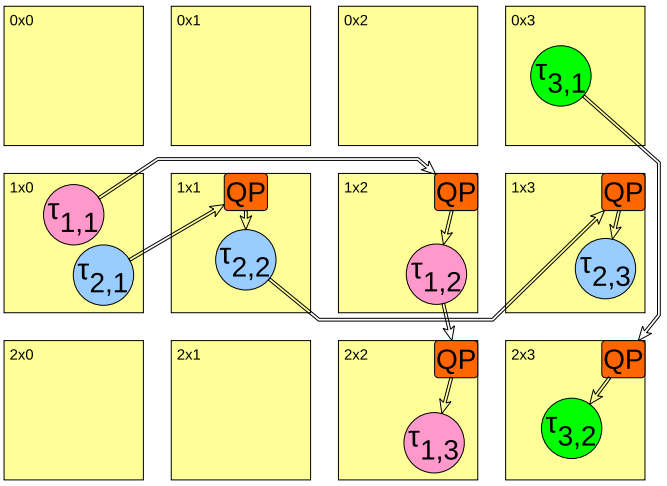

Fig. 21. System example, formed by three e2e flows.

$$TT_{221} = TT^b_{221} + I_{221} = 4.5 + 8 = 12.5 cycles$$

$$TT^b_{222} = L_H \cdot H = 1.5 \cdot 3 = 4.5 cycles$$

$$TT_{222} = TT^b_{222} + I_{222} = 4.5 + 1 = 5.5 cycles$$

$$TT^b_{227} = L_H \cdot H = 1.5 \cdot 3 = 4.5 cycles$$

$$TT_{227} = TT^b_{227} + I_{227} = 4.5 + 1 = 5.5 cycles$$

Once we have calculated the message traversal times we can get the packet generation rates. The read messages and the write-back messages have their packet generation rate calculated using Eq. (7) with a value of 25 cycles for the c parameter (as was described in Section 7.3) and the traversal times obtained in the previous paragraph. The calculation of the packet generation rates of one read message (m_{221}) and a write-back message (m_{222}) are shown below:

$$\rho_{221} = \frac{1}{TT^b_{221} + TT^b_{222} + c} = \frac{1}{4.5 + 4.5 + 25} = \frac{1}{34} cycles^{-1}$$

$$\rho_{222} = \frac{1}{TT^b_{222} + TT^b_{221} + c} = \frac{1}{4.5 + 4.5 + 25} = \frac{1}{34} cycles^{-1}$$

Regarding the packet generation rate of the write messages, the most restrictive is m_{113} (the one that writes the data in the QP) that has the generation rate of the `memcopy` function ($\frac{1}{3}$ cycles^{-1}). For their part, the write messages to set the data as new and unlock the mutex are both one packet long and, taken as a group, they have a generation rate of $\frac{1}{8}$ cycles^{-1}.

The rates of the NoC links are shown in Fig. 22 which includes the packet generation rate of all the messages in the system. As it can be seen, all the links verify the maximum rate restriction, as the links of the *cMesh* (blue wires) could have up to rate one and the links of the *rMesh* (red wires) could have up to rate $\frac{1}{8}$.

The worst-case execution time of τ_{22} must be augmented in the maximum blocking time due to the use of QPs shared with τ_{21} and τ_{23}. The execution time of the read operation performed by τ_{23} on its local QP requires a time of $B_{23} = 1602$ ns. For its part, the write operation performed by τ_{21} on the QP on the core 1×1 requires a time of $B_{21} = 780$ ns. In order to consider the worst case situation, B_{21} must be augmented with the interference times of all the read and write-back packets, but in this scenario the messages going from τ_{21} to τ_{22} will not be interfered by any other packet in the NoC:

$$B'_{21} = B_{21} + I_{211} + I_{212} + I_{213} + I_{214} + I_{215} + I_{216}$$

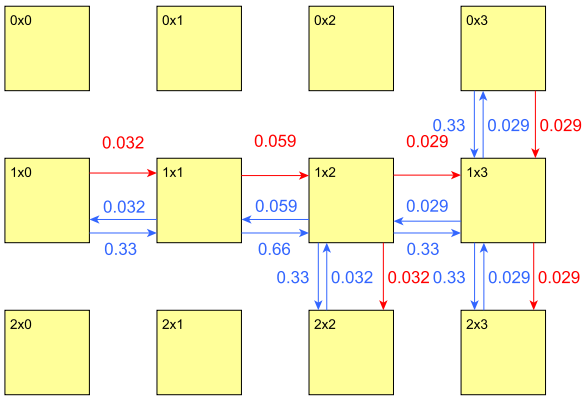

Fig. 22. Rates of the links of the NoC for the queuing port example (read messages in red and write and write-back messages in blue). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

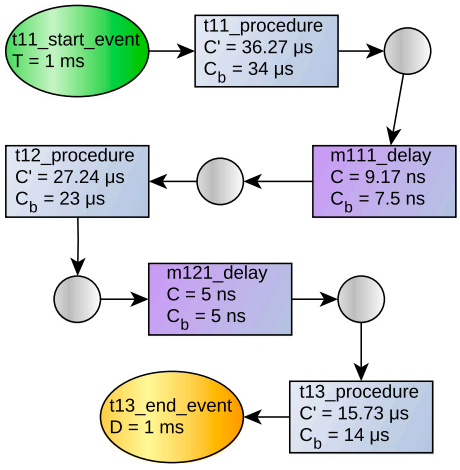

Fig. 23. First *e2e* flow modeled in MAST.

$$B'_{21} = 780 \text{ ns} + 0 \text{ ns} = 780 \text{ ns}$$

Besides, in order to consider the worst case situation, the worst-case execution time of τ_{22} must also be augmented with the interference times of all the read and write-back packets as shown below:

$$C'_{22} = C_{22} + B_{23} + B'_{21} + I_{221} + I_{222} + I_{223} + I_{224} + I_{225} + I_{226}$$

$$C'_{22} = 25 \text{ µs} + 1602 \text{ ns} + 780 \text{ ns} + (25.33 + 2.5) \cdot 3 \text{ ns}$$

$$C'_{22} = 27.47 \text{ µs}$$

The same process is followed by the rest of messages and tasks resulting in the values displayed in Tables 6 and 7.

Figs. 23–25 show the resulting MAST elements, Table 8 shows the response times obtained using MAST as well as the response times of the Epiphany processor executing synthetic tasks with the same configuration. It can be seen that the measured execution is bounded by the MAST model worst-case results as they are lower and very close to the ones obtained by the analysis.

9. Conclusion

The modeling and analysis technique presented in this paper allows us to reduce the complexity of the analysis of inter-message interference thanks to the introduction of a limitation on the maximum link utilization rate.

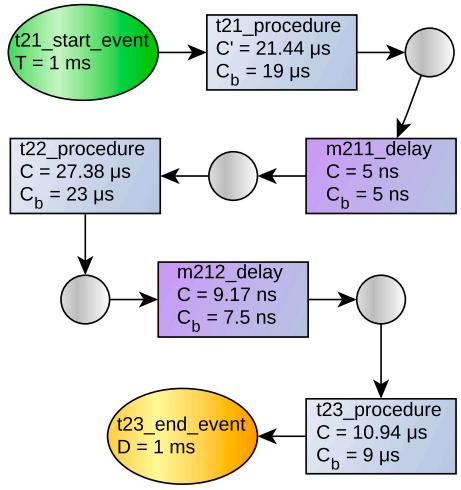

Fig. 24. Second *e2e* flow modeled in MAST.

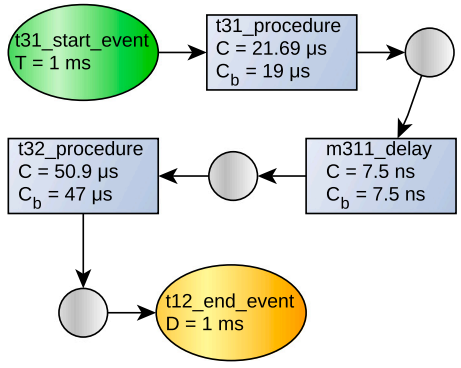

Fig. 25. Third *e2e* flow modeled in MAST.

Our approach includes the modeling of high-level communication mechanisms provided by the operating system, namely sampling ports and queuing ports.

Unlike most of the related work, our modeling and analysis technique has been applied and tested on a real platform: the Epiphany many-core processor which has 16 cores connected by a 4 × 4 2D mesh using a real-time operating system designed for many-core processors called M2OS-mc.

We have verified that the synchronized messages originated by an application running on the M2OSmc RTOS in the Epiphany many core processor have a bounded time behavior and are correctly bounded by the developed analysis.

The synchronized messages are fully integrated in the M2OS-mc project and the experiments performed for this paper are easily reproducible in a Parallella board.

As future work we leave the study of the behavior of real-time systems on different platforms. Using other processors with a 2D mesh-based NoC will be another test on the capacity of our model. We can also study how to export this model to processors with different types of NoCs.

Besides, to implement real industrial systems we need to develop a synchronized way to use the I/O of the Epiphany processor and to access the external memory. Once these features are implemented we would need to model and analyze their timing behavior.

The design of a task allocation algorithm is proposed as further research looking for a better response-time performance of the whole system.

Table 6
Parameters for the queuing port example with three e2e flows (1/2).

Γ_1	$(T_1 = 1\ \text{ms}\quad D_1 = 1\ \text{ms})$
τ_{11}	$C_{11} = 35\ \mu\text{s}\ \ C^b_{11} = 34\ \mu\text{s}\ \ C'_{11} = 36.35\ \mu\text{s}$
m_{111}	$\mu_{111} = 1\ \ t_{111} = read\ \ \rho_{111} = 83.33\ \text{mcycles}^{-1}$
m_{112}	$\mu_{112} = 1\ \ t_{112} = write-back\ \ \rho_{112} = 83.34\ \text{mcycles}^{-1}$
m_{113}	$\mu_{113} = 1\ \ t_{113} = read\ \ \rho_{113} = 83.33\ \text{mcycles}^{-1}$
m_{114}	$\mu_{114} = 1\ \ t_{114} = write-back\ \ \rho_{114} = 83.34\ \text{mcycles}^{-1}$
m_{115}	$\mu_{115} = 1\ \ t_{115} = read\ \ \rho_{115} = 83.33\ \text{mcycles}^{-1}$
m_{116}	$\mu_{116} = 1\ \ t_{116} = write-back\ \ \rho_{116} = 83.34\ \text{mcycles}^{-1}$
m_{117}	$\mu_{117} = 1\ \ t_{117} = write\ \ \rho_{117} = 58.82\ \text{mcycles}^{-1}$
m_{118}	$\mu_{118} = 3\ \ t_{118} = write\ \ \rho_{118} = 0.33\ \text{cycle}^{-1}$
m_{119}	$\mu_{119} = 1\ \ t_{119} = write\ \ \rho_{119} = 58.82\ \text{mcycles}^{-1}$
τ_{12}	$C_{12} = 23\ \mu\text{s}\ \ C^b_{12} = 25\ \mu\text{s}\ \ C'_{12} = 27.24\ \mu\text{s}$
m_{121}	$\mu_{121} = 1\ \ t_{121} = read\ \ \rho_{121} = 0.11\ \text{cycles}^{-1}$
m_{122}	$\mu_{122} = 1\ \ t_{122} = write-back\ \ \rho_{122} = 0.11\ \text{cycles}^{-1}$
m_{123}	$\mu_{123} = 1\ \ t_{123} = read\ \ \rho_{123} = 0.11\ \text{cycles}^{-1}$
m_{124}	$\mu_{124} = 1\ \ t_{124} = write-back\ \ \rho_{124} = 0.11\ \text{cycles}^{-1}$
m_{125}	$\mu_{125} = 1\ \ t_{125} = read\ \ \rho_{125} = 0.11\ \text{cycles}^{-1}$
m_{126}	$\mu_{126} = 1\ \ t_{126} = write-back\ \ \rho_{126} = 0.11\ \text{cycles}^{-1}$
m_{127}	$\mu_{127} = 1\ \ t_{127} = write\ \ \rho_{127} = 71.43\ \text{mcycles}^{-1}$
m_{128}	$\mu_{128} = 5\ \ t_{128} = write\ \ \rho_{128} = 0.33\ \text{cycle}^{-1}$
m_{129}	$\mu_{129} = 1\ \ t_{129} = write\ \ \rho_{129} = 71.43\ \text{mcycles}^{-1}$
τ_{13}	$C_{13} = 15\ \mu\text{s}\ \ C^b_{13} = 14\ \mu\text{s}\ \ C'_{13} = 15.73\ \mu\text{s}$
Γ_2	$(T_2 = 1\ \text{ms}\quad D_2 = 1\ \text{ms})$
τ_{21}	$C_{21} = 20\ \mu\text{s}\ \ C^b_{21} = 19\ \mu\text{s}\ \ C'_{21} = 21.44\ \mu\text{s}$
m_{211}	$\mu_{211} = 1\ \ t_{211} = read\ \ \rho_{211} = 0.11\ \text{cycles}^{-1}$
m_{212}	$\mu_{212} = 1\ \ t_{212} = write-back\ \ \rho_{212} = 0.11\ \text{cycles}^{-1}$
m_{213}	$\mu_{213} = 1\ \ t_{213} = read\ \ \rho_{213} = 0.11\ \text{cycles}^{-1}$
m_{214}	$\mu_{214} = 1\ \ t_{214} = write-back\ \ \rho_{214} = 0.11\ \text{cycles}^{-1}$
m_{215}	$\mu_{215} = 1\ \ t_{215} = read\ \ \rho_{215} = 0.11\ \text{cycles}^{-1}$
m_{216}	$\mu_{216} = 1\ \ t_{216} = write-back\ \ \rho_{216} = 0.11\ \text{cycles}^{-1}$
m_{217}	$\mu_{217} = 1\ \ t_{217} = write\ \ \rho_{217} = 71.43\ \text{mcycles}^{-1}$
m_{218}	$\mu_{218} = 7\ \ t_{218} = write\ \ \rho_{218} = 0.33\ \text{cycles}^{-1}$
m_{219}	$\mu_{219} = 1\ \ t_{219} = write\ \ \rho_{219} = 71.43\ \text{mcycles}^{-1}$
τ_{22}	$C_{22} = 25\ \mu\text{s}\ \ C^b_{22} = 23\ \mu\text{s}\ \ C'_{22} = 27.47\ \mu\text{s}$
m_{221}	$\mu_{221} = 1\ \ t_{221} = read\ \ \rho_{221} = 83.33\ \text{mcycles}^{-1}$
m_{222}	$\mu_{212} = 1\ \ t_{222} = write-back\ \ \rho_{222} = 83.34\ \text{mcycles}^{-1}$
m_{223}	$\mu_{223} = 1\ \ t_{223} = read\ \ \rho_{223} = 83.33\ \text{mcycles}^{-1}$
m_{224}	$\mu_{224} = 1\ \ t_{224} = write-back\ \ \rho_{224} = 83.34\ \text{mcycles}^{-1}$
m_{225}	$\mu_{225} = 1\ \ t_{225} = read\ \ \rho_{225} = 83.33\ \text{mcycles}^{-1}$
m_{226}	$\mu_{226} = 1\ \ t_{226} = write-back\ \ \rho_{226} = 83.34\ \text{mcycles}^{-1}$
m_{227}	$\mu_{227} = 1\ \ t_{227} = write\ \ \rho_{227} = 58.82\ \text{mcycles}^{-1}$
m_{228}	$\mu_{228} = 11\ \ t_{228} = write\ \ \rho_{228} = 0.33\ \text{cycles}^{-1}$
m_{229}	$\mu_{229} = 1\ \ t_{229} = write\ \ \rho_{229} = 58.82\ \text{mcycles}^{-1}$
τ_{23}	$C_{23} = 10\ \mu\text{s}\ \ C^b_{23} = 9\ \mu\text{s}\ \ C'_{23} = 10.94\ \mu\text{s}$

Table 7
Parameters for the queuing port example with three e2e flows (2/2).

Γ_3	$T_3 = 1\ \text{ms}\quad D_3 = 1\ \text{ms}$
τ_{31}	$C_{31} = 20\ \mu\text{s}\ \ C^b_{31} = 19\ \mu\text{s}\ \ C'_{31} = 21.69\ \mu\text{s}$
m_{311}	$\mu_{311} = 1\ \ t_{311} = read\ \ \rho_{311} = 83.33\ \text{mcycles}^{-1}$
m_{312}	$\mu_{312} = 1\ \ t_{312} = write-back\ \ \rho_{312} = 83.34\ \text{mcycles}^{-1}$
m_{313}	$\mu_{313} = 1\ \ t_{313} = read\ \ \rho_{313} = 83.33\ \text{mcycles}^{-1}$
m_{314}	$\mu_{314} = 1\ \ t_{314} = write-back\ \ \rho_{314} = 83.34\ \text{mcycles}^{-1}$
m_{315}	$\mu_{315} = 1\ \ t_{315} = read\ \ \rho_{315} = 83.33\ \text{mcycles}^{-1}$
m_{316}	$\mu_{316} = 1\ \ t_{316} = write-back\ \ \rho_{316} = 83.34\ \text{mcycles}^{-1}$
m_{317}	$\mu_{317} = 1\ \ t_{317} = write\ \ \rho_{317} = 58.82\ \text{mcycles}^{-1}$
m_{318}	$\mu_{318} = 13\ \ t_{318} = write\ \ \rho_{318} = 0.33\ \text{cycles}^{-1}$
m_{319}	$\mu_{319} = 1\ \ t_{319} = write\ \ \rho_{319} = 58.82\ \text{mcycles}^{-1}$
τ_{32}	$C_{32} = 50\ \mu\text{s}\ \ C^b_{32} = 47\ \mu\text{s}\ \ C'_{32} = 50.9\ \mu\text{s}$

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Table 8
Results of the MAST model and the Epiphany executions for the queuing port example.

Γ_1 flow					
Step	MAST model results		Epiphany execution results		
	WCRT	BCRT	Max RT	Avg RT	Min RT
τ_{11}	57,779 ns	34,000 ns	35,305 ns	35,247 ns	35,218 ns
τ_{12}	85,039 ns	57,008 ns	71,015 ns	63,343 ns	59,458 ns
τ_{13}	100,774 ns	71,013 ns	84,990 ns	77,308 ns	73,410 ns
Γ_2 flow					
Step	MAST model results		Epiphany execution results		
	WCRT	BCRT	Max RT	Avg RT	Min RT
τ_{21}	57,790 ns	19,000 ns	20,172 ns	20,170 ns	20,170 ns
τ_{22}	85,235 ns	42,005 ns	55,698 ns	47,943 ns	43,967 ns
τ_{23}	96,184 ns	51,013 ns	64,113 ns	56,350 ns	52,377 ns
Γ_3 flow					
Step	MAST model results		Epiphany execution results		
τ_{31}	21,690 ns	19,000 ns	20,007 ns	20,003 ns	20,003 ns
τ_{32}	72,598 ns	66,008 ns	67,868 ns	67,865 ns	67,862 ns

References

[1] L. Benini, G. De Micheli, Networks on chips: a new SoC paradigm, Computer 35 (1) (2002) 70–78, http://dx.doi.org/10.1109/2.976921.

[2] M. Zhang, J. Shi, T. Zhang, Y. Hu, Hard real-time communication over multi-hop switched ethernet, in: 2008 International Conference on Networking, Architecture, and Storage, IEEE, 2008, pp. 121–128, http://dx.doi.org/10.1109/NAS.2008.31.

[3] A. Yiming, T. Eisaka, A switched ethernet protocol for hard real-time embedded system applications, J. Interconnect. Netw. 6 (3) (2005) 345–360, http://dx.doi.org/10.1142/S0219265905001460.

[4] S. Lee, K.C. Lee, H.H. Kim, Maximum communication delay of a real-time industrial switched ethernet with multiple switching hubs, in: 30th Annual Conference of IEEE Industrial Electronics Society, 2004. IECON 2004, Vol. 3, IEEE, 2004, pp. 2327–2332, http://dx.doi.org/10.1109/IECON.2004.1432163.

[5] Z. Shi, A. Burns, Real-time communication analysis for on-chip networks with wormhole switching, in: Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip, NOCS '08, IEEE Computer Society, USA, 2008, pp. 161–170, http://dx.doi.org/10.5555/1397757.1397996, URL https://dl.acm.org/doi/10.5555/1397757.1397996.

[6] L.S. Indrusiak, End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration, J. Syst. Archit. 60 (7) (2014) 553–561, http://dx.doi.org/10.1016/j.sysarc.2014.05.002, URL http://www.sciencedirect.com/science/article/pii/S1383762114000800.

[7] M. Becker, B. Nikolic, D. Dasari, B. Akesson, V. Nélis, M. Behnam, T. Nolte, Partitioning and analysis of the network-on-chip on a COTS many-core platform, in: 2017 IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS, 2017, pp. 101–112, http://dx.doi.org/10.1109/RTAS.2017.32.

[8] M. Becker, S. Mubeen, D. Dasari, M. Behnam, T. Nolte, Scheduling multi-rate real-time applications on clustered many-core architectures with memory constraints, in: 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), 2018, pp. 560–567, http://dx.doi.org/10.1109/ASPDAC.2018.8297382.

[9] L. Indrusiak, A. Burns, B. Nikolic, Analysis of Buffering Effects on Hard Real-Time Priority-Preemptive Wormhole Networks, 2016, http://dx.doi.org/10.48550/ARXIV.1606.02942, arXiv. URL https://arxiv.org/abs/1606.02942.

[10] Q. Xiong, Z. Lu, F. Wu, C. Xie, Real-time analysis for wormhole NoC: Revisited and revised, in: Proceedings of the 26th Edition on Great Lakes Symposium on VLSI, GLSVLSI '16, Association for Computing Machinery, 2016, pp. 75–80, http://dx.doi.org/10.1145/2902961.2903023.

[11] M. Boyer, A. Graillat, B. Dupont de Dinechin, J. Migge, Bounding the delays of the MPPA network-on-chip with network calculus: Models and benchmarks, Perform. Eval. 143 (2020) 102124, http://dx.doi.org/10.1016/j.peva.2020.102124, URL https://www.sciencedirect.com/science/article/pii/S0166531620300444.

[12] S. Tobuschat, R. Ernst, Real-time communication analysis for networks-on-chip with backpressure, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2017, 2017, pp. 590–595, http://dx.doi.org/10.23919/DATE.2017.7927055.

[13] M. Dridi, F. Singhoff, S. Rubini, J.-P. Diguet, ECTM: A network-on-chip communication model to combine task and message schedulability analysis, J. Syst. Archit. 114 (2020) 101931, http://dx.doi.org/10.1016/j.sysarc.2020.101931, URL http://www.sciencedirect.com/science/article/pii/S1383762120301909.

[14] S. Gopalakrishnan, L. Sha, M. Caccamo, Hard Real-Time Communication in Bus-Based Networks, IEEE, 2004, pp. 405–414, http://dx.doi.org/10.1109/REAL.2004.24, URL https://ieeexplore.ieee.org/abstract/document/1381326.

[15] S.K. Roy, R. Devaraj, A. Sarkar, K. Maji, S. Sinha, Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems, J. Syst. Archit. 105 (2020) 101706, http://dx.doi.org/10.1016/j.sysarc.2019.101706, URL https://www.sciencedirect.com/science/article/pii/S1383762119305132.

[16] B. Nikolić, S. Tobuschat, L. Soares Indrusiak, R. Ernst, A. Burns, Real-time analysis of priority-preemptive NoCs with arbitrary buffer sizes and router delays, Real-Time Syst. 55 (1) (2019) 63–105, http://dx.doi.org/10.1007/s11241-018-9312-0.

[17] D. Dasari, B. Nikoli'c, V. N'elis, S.M. Petters, Noc contention analysis using a branch-and-prune algorithm, ACM Trans. Embed. Comput. Syst. 13 (3s) (2014) http://dx.doi.org/10.1145/2567937.

[18] W. Puffitsch, E. Noulard, C. Pagetti, Off-line mapping of multi-rate dependent task sets to many-core platforms, Real-Time Syst. 51 (2015) 526–565, http://dx.doi.org/10.1007/s11241-015-9232-1.

[19] S. Metzlaff, J. Mische, T. Ungerer, A real-time capable many-core model, in: Proceedings of 32nd IEEE Real-Time Systems Symposium: Work-in-Progress Session, 2011, pp. 21–24.

[20] C. Benchehida, M.K. Benhaoua, H.-E. Zahaf, G. Lipari, Task and communication allocation for real-time tasks to networks-on-chip multiprocessors, in: 2020 Second International Conference on Embedded Distributed Systems, EDiS, 2020, pp. 9–14, http://dx.doi.org/10.1109/EDiS49545.2020.9296446, URL https://ieeexplore.ieee.org/document/9296446.

[21] M. Harbour, J.J. Gutiérrez, J. Palencia, J. Moyano, MAST: Modeling and analysis suite for real time applications, in: Proceedings - Euromicro Conference on Real-Time Systems, 2001, pp. 125–134, http://dx.doi.org/10.1109/EMRTS.2001.934015.

[22] J. Palencia, M. Harbour, Offset-based response time analysis of distributed systems scheduled under EDF, in: 15th Euromicro Conference on Real-Time Systems, 2003. Proceedings, 2003, pp. 3–12, http://dx.doi.org/10.1109/EMRTS.2003.1212721.

[23] M. Becker, D. Dasari, S. Mubeen, M. Behnam, T. Nolte, End-to-end timing analysis of cause-effect chains in automotive embedded systems, J. Syst. Archit. 80 (C) (2017) 104–113, http://dx.doi.org/10.1016/j.sysarc.2017.09.004.

[24] A. Girault, C. Prévot, S. Quinton, R. Henia, N. Sordon, Improving and estimating the precision of bounds on the worst-case latency of task chains, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (11) (2018) 2578–2589, http://dx.doi.org/10.1109/TCAD.2018.2861016.

[25] M. Günzel, K.-H. Chen, N. Ueter, G.v.d. Brüggen, M. Dürr, J.-J. Chen, Timing analysis of asynchronized distributed cause-effect chains, in: 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium, RTAS, 2021, pp. 40–52, http://dx.doi.org/10.1109/RTAS52030.2021.00012.

[26] D.G. Villaescusa, M.A. Rivas, M.G. Harbour, M2OS-Mc: An RTOS for Many-Core Processors, in: M. Bertogna, F. Terraneo (Eds.), Second Workshop on Next Generation Real-Time Embedded Systems, NG-RES 2021, in: OpenAccess Series in Informatics (OASIcs), vol. 87, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021, pp. 5:1–5:13, http://dx.doi.org/10.4230/OASIcs.NG-RES.2021.5, URL https://drops.dagstuhl.de/opus/volltexte/2021/13481.

[27] D. García Villaescusa, M. Aldea Rivas, M. González Harbour, Queuing ports for mesh based many-core processors, Ada Lett. 41 (2) (2022) 66–70, http://dx.doi.org/10.1145/3530801.3530804.

[28] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, Y.-C. Liu, Knights landing: Second-generation Intel Xeon Phi product, IEEE Micro 36 (2) (2016) 34–46, http://dx.doi.org/10.1109/MM.2016.25.

[29] Corporation, Tile processor architecture overview for the tile-GX series, 2012, URL https://cdn.manesht.ir/17871__210769647-UG130-ArchOverview-TILE-Gx.pdf.

[30] B.D. de Dinechin, R. Ayrignac, P.-E. Beaucamps, P. Couvert, B. Ganne, P.G. de Massas, F. Jacquet, S. Jones, N.M. Chaisemartin, F. Riss, T. Strudel, A clustered manycore processor architecture for embedded and accelerated applications, in: 2013 IEEE High Performance Extreme Computing Conference, HPEC, 2013, pp. 1–6, http://dx.doi.org/10.1109/HPEC.2013.6670342.

[31] E.A. Rambo, R. Ernst, Worst-case communication time analysis of networks-on-chip with shared virtual channels, in: 2015 Design, Automation Test in Europe Conference Exhibition, DATE, 2015, pp. 537–542, http://dx.doi.org/10.7873/DATE.2015.0023.

[32] A. Olofsson, Epiphany Architecture Reference, Technical report, Adapteva, 2013, URL https://www.adapteva.com/docs/epiphany_arch_ref.pdf.

[33] A. Olofsson, Epiphany-v: A 1024 processor 64-bit risc system-on-chip, 2016, arXiv preprint arXiv:1610.01832. http://dx.doi.org/10.48550/ARXIV.1610.01832.

[34] A. Olofsson, Parallella Reference Manual, Technical report, Adapteva, URL https://www.parallella.org/docs/parallella_manual.pdf.

[35] J.G. Balaguer, J.R.Z. Flores, J.A. de la Puente Alfaro, ARINC-653 inter-partition communications and the ravenscar profile, in: ACM SIGAda Ada Letters, Vol. 35. No. 1, April 2015, Ada Letters (2015) 38–45, http://dx.doi.org/10.1145/2870544.2870550, URL http://oa.upm.es/42418/.

David García Villaescusa is a Security Researcher at Ikerlan S. Coop. He previously obtained the PhD at the University of Cantabria. Before that he worked for Equipos Nucleares S.A. His more recent research focuses on the usage of mesh-based many-cores for real-time porpuses. He has developed the free software RTOS Marte OS for Raspberry Pi 1 (https://marte.unican.es/) and M2OS for Epiphany III (https://m2os.unican.es/) both with GPLv3+ licence.

Mario Aldea Rivas is currently an Associate Professor with the Department of Electronics and Computers, University of Cantabria, Spain. His research interests include real-time systems, with special focus on flexible scheduling, real-time operating systems, and real-time languages. He has been involved in several research and industrial projects related with real-time and embedded technologies. He is also the main developer of MaRTE OS, an operating system that has served as platform to provide support for advanced real-time services.

Michael González Harbour is a Professor in the Department of Computer Science and Electronics at the University of Cantabria. He works in software engineering for real-time systems, and particularly in modelling and schedulability analysis of distributed real-time systems, real-time operating systems, and real-time languages. He is a co-author of “A Practitioner’s Handbook on Real-Time Analysis”. He has been involved in several industrial projects using Ada to build real-time controllers for robots. Michael has participated in the real-time working group of the POSIX standard for portable operating system interfaces. He is one of the principal authors of the MAST suite for modelling and analysing real-time systems.