CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Partitioning and Analysis of the Network-on-Chip on a COTS Many-Core Platform

Matthias Becker

Borislav Nikolic

Dakshina Dasari

Benny Åkesson*

Vincent Nélis*

Moris Behnam

Thomas Nolte

*CISTER Research Centre

CISTER-TR-170302

2017/04/18

Partitioning and Analysis of the Network-on-Chip on a COTS Many-Core Platform

Matthias Becker, Borislav Nikolic, Dakshina Dasari, Benny Åkesson*, Vincent Nélis*, Moris Behnam, Thomas Nolte

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: matthias.becker@mdh.se, borni@isep.ipp.pt, dandi@isep.ipp.pt, kbake@isep.ipp.pt, nelis@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Many-core processors can provide the computational power required by future complex embedded systems. However, their adoption is not trivial, since several sources of interference on COTS many-core platforms have adverse effects on the resulting performance. One main source of performance degradation is the contention on the Network-on-Chip, which is used for communication among the compute cores via the offchip memory. Available analysis techniques for the traversal time of messages on the NoC do not consider many of the architectural features found on COTS platforms.

In this work, we target a state-of-the-art many-core processor, the Kalray MPPA. A novel partitioning strategy for reducing the contention on the NoC is proposed. Further, we present an analysis technique dedicated to the proposed partitioning strategy, which considers all architectural features of the COTS NoC. Additionally, it is shown how to configure the parameters for flow-regulation on the NoC, such that the Worst-Case Traversal Time (WCTT) is minimal and buffers never overflow. The benefits of our approach are evaluated based on extensive experiments that show that contention is significantly reduced compared to the unconstrained case, while the proposed analysis outperforms a state-of-the-art analysis for the same platform. An industrial case study shows the tightness of the proposed analysis.

Partitioning and Analysis of the Network-on-Chip on a COTS Many-Core Platform

Matthias Becker*, Borislav Nikolić§, Dakshina Dasari‡, Benny Akesson†
Vincent Nélis†, Moris Behnam*, Thomas Nolte*
*MRTC / Mälardalen University, Sweden {matthias.becker, moris.behnam, thomas.nolte}@mdh.se
§ Institute of Computer and Network Engineering, Technische Universität Braunschweig, Germany bnikolic@ida.ing.tu-bs.de
†CISTER/INESC-TEC, ISEP, Portugal {borni, kbake, nelis}@isep.ipp.pt
‡ Robert Bosch GmbH, Renningen, Germany dakshina.dasari@de.bosch.com

***Abstract*—Many-core processors can provide the computational power required by future complex embedded systems. However, their adoption is not trivial, since several sources of interference on COTS many-core platforms have adverse effects on the resulting performance. One main source of performance degradation is the contention on the Network-on-Chip (NoC), which is used for communication among the compute cores via the off-chip memory. Available analysis techniques for the traversal time of messages on the NoC do not consider many of the architectural features found on COTS platforms.**

In this work, we target a state-of-the-art many-core processor, the Kalray MPPA®. A novel partitioning strategy for reducing the contention on the NoC is proposed. Further, we present an analysis technique dedicated to the proposed partitioning strategy, which considers all architectural features of the COTS NoC. Additionally, it is shown how to configure the parameters for flow-regulation on the NoC, such that the Worst-Case Traversal Time (WCTT) is minimal and buffers never overflow. The benefits of our approach are evaluated based on extensive experiments that show that contention is significantly reduced compared to the unconstrained case, while the proposed analysis outperforms a state-of-the-art analysis for the same platform. An industrial case study shows the tightness of the proposed analysis.

I. Introduction

Many industrial domains face the challenge of proliferation in their systems. Traditional system designs are founded on a one application per controller principle. Powerful backbone networks are used to provide the required inter-application communication, as well as the communication with sensors and actuators. Due to the growing system complexity, a threshold was reached where additional controllers cannot be added to the system, either due to space limitations or cost restrictions. A prominent example is the automotive domain [1], [2].

Many-core processors are a promising platform in this domain [3], [4]. With tens or even hundreds of compute cores on a single processor, they provide enough computational performance to host several applications [5], while their low energy consumption makes them well-suited for deployment in embedded systems. The compute elements on a many-core processor are arranged in so called tiles, where each tile can host a number of cores. The tiles themselves connect to a Network-on-Chip (NoC) that is used to access shared resources, such as I/O or off-chip memory. However, the analysis of such systems is not trivial since several sources of interference that affect the execution of applications need to be considered [6], [7].

978-1-5090-5269-1/17/$31.00 ©2017 IEEE

This work assumes a mapping of one application to one tile. This is reasonable since a tile of the considered Commercial Off-The-Shelf (COTS) platform, the Kalray MPPA®[8], contains 16 compute cores. Additional benefits arise through the spatial isolation between tiles on the die. If all required resources are available within a tile, an application executing in one tile *cannot* affect another application on another tile.

Current state-of-the art analysis techniques for NoCs do not consider many of the crucial architectural elements that are found in real implementations. For instance, an important low-level detail with the Kalray MPPA® is that it provides flow regulation on source nodes [9] in order to avoid buffer overflows on the NoC router. Another source of pessimism may arise if the buffer in the NoC router is neglected. Traditional analysis techniques either do not factor-in the impact of all the involved elements, which may lead to optimistic results [10] in the presence of buffers, or they lead to over-approximations which in turn leads to an under-utilization of the platform [11].

This paper addresses the problem of contention in the NoC through the following five contributions:

1) A NoC organization based on symmetric partitioning, which reduces contention on the access to off-chip resources, such as memory or I/O, and additionally provides composability among clusters. We demonstrate that this reduced contention leads to faster traversal of messages in the path from the cluster to the off-chip DDR memory via the I/O cluster, thereby reducing effectively the *overall access latency*. This is because the benefits of the NoC management are also sustained at the DDR subsystem level.
2) An analysis of the contention-induced delays in a partition is presented, taking the flow regulation on source nodes, as well as the limited buffer size on NoC routers into account.
3) A method to configure the flow regulation on source nodes in a way that minimizes the Worst-Case Traversal Time (WCTT) of application messages is proposed.
4) Experimental evaluations show the benefits of the partitioned model and the presented analysis compared to existing approaches.
5) Finally, an industrial case study is presented to demonstrate the strength of the approach and the reduced pessimism of the proposed analysis.

The rest of the paper is organized as follows. Section II discusses related work, followed by a description of the system model in Section III. The proposed NoC partitioning strategy is introduced in Section IV. Section V presents the calculation of the WCTT for application messages, and Section VI introduces the configuration of the flow regulation. Experiments are presented in Section VII. Lastly, conclusions are drawn in Section VIII.

II. Related Work

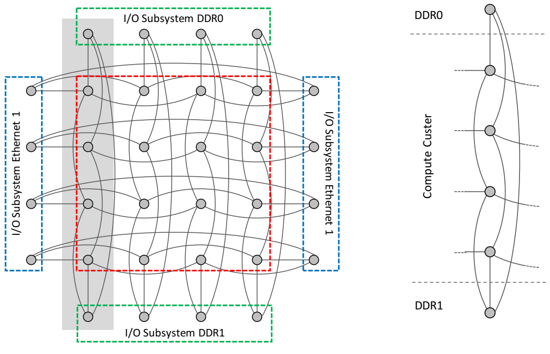
The design and analysis of NoC architectures targeting applications with strict timing constraints has received significant attention in recent years. Two main approaches can be observed: 1) The design of NoC architectures tailored to avoid contention by applying Time-Division Multiplexing (TDM) [12], [13], [14], [15]. This provides high predictability and allows for simple WCTT calculations, but may waste bandwidth on the NoC. 2) The second approach lies in the analysis of the worst-case behavior of a given NoC. This generally targets the NoC implemented on COTS many-core platforms (e.g. Kalray MPPA® 256 [8] or Tileras Tile Processor [16]). In this work, we focus on Round Robin (RR) arbitrated NoC architectures as this is the arbitration mechanism implemented on the Kalray MPPA® [8].

Ferrandiz et al. [17] propose the Recursive Calculus (RC) method for RR arbitrated NoCs. The RC takes the largest possible interference of contending flows on each link into account and calculates a safe WCTT for individual NoC packets. Dasari et al. [18], as well as Liu et al. [19], extended the RC method to account for periodic NoC packets, and Abdallah et al. [20] extended the original RC with additional properties to reduce pessimism. Ayed et al. [11] adapt the RC for a Kalray-like NoC. However, all these methods only compute WCTT for *individual NoC packets* and they *do not consider buffer effects* on the NoC.

The NoC of the Kalray MPPA® 256 is designed to provide for guaranteed service [8]. This is achieved by flow regulation on the source node [21] together with non-blocking routers (i.e., there is no link-level flow control implemented). The main intention of this design is to enable simple analysis using Network Calculus (NC) [9].

The existing methods have several limitations. The RC-based methods do not consider buffers and hence the computed results may be optimistic (unsafe) if buffers are present, as shown in [10]. Also, these methods only consider individual NoC packets. However, realistic application messages consist of a *sequence of packets* released at the same time. Hence, an earlier NoC packet of the same application message may block a later one – While this is considered by the NC approaches, the computed results are pessimistic in most scenarios [11]. Puffitsch et al. [22] compare TDM and a dynamically scheduled NoC for the example of the Kalray NoC [8] and the Argo NoC [14]. The WCTT of the dynamically scheduled NoC were based on NC. Their experimental evaluation shows that the WCTT of the TDM NoC is generally tighter then the dynamically scheduled NoC using the NC analysis, but that the dynamically scheduled NoC achieves higher utilization.

For most many-core platforms, the on-chip memory is small compared to the application footprint (instructions plus data). This means that during the execution of the application access to off-chip memory is required. This generally introduces additional load on the system and may affect the execution time of applications.



(a) (b)

Fig. 1: NoC architecture of the Kalray MPPA® 256 with four I/O subsystem regions and the compute cluster region in the middle is shown in (a). The NoC links to the compute clusters and to the I/O nodes are omitted for better clarity. (b) One column of the NoC (the column with gray shaded background).

Giannopoulou et al. [23] propose a framework for mixed-criticality scheduling on one cluster of a many-core processor, such as the Kalray MPPA®. Time-triggered scheduling is used on the clusters while WCTT on the NoC is computed using real-time calculus and NC. They consider access to off-cluster memory in their analysis.

Perret et al. [7] analyze the potential sources of interference on clustered COTS many-core architectures, such as the Kalray MPPA®. They identify the NoC as well as the external DDR memory as main sources for interference. Consequently, they provide design guidelines for execution frameworks that adhere to the requirements found in safety-critical embedded systems. In [24], the same authors present an execution framework for time-critical applications on the Kalray MPPA®. To add predictability to the NoC, they implement TDM on top of the RR-arbitrated COTS NoC. This then fully decouples different applications.

Our method differs from the existing approaches by *taking the authentic NoC topology of the Kalray MPPA® into account*, which allows to partition the NoC such that contention is minimized. We then show how to compute tight WCTT for application messages, which are divided into several NoC packets, on such partitions. This *includes* the buffering effects on NoC router and the injection pattern imposed to the injection of NoC packets by the flow regulation on source nodes. It is further shown how to select the flow regulation parameters in a way that buffer overflows in NoC routers are avoided.

III. System Model

This section introduces the NoC architecture found on the Kalray MPPA® many-core processor, as well as the communication model used by the clusters.

A. NoC Architecture

The NoC is the interconnect of choice for today's many-core platforms [25]. The Kalray MPPA® consist of 16 compute clusters, arranged on a 4x4 grid. Each compute cluster hosts 16 compute cores, as well as local memory. A network interface is used to connect the cluster to the NoC router. In contrast to the traditional 2D-mesh based topology that can be found on Intel's Single Chip Cloud Computer [26] or on Tilera's Tile Processor [16], the NoC on the Kalray MPPA® is arranged in a torus topology, as shown in Fig. 1a. In addition to the 16 compute clusters, 16 nodes connect to 4 so called I/O subsystems. Each of the 4 I/O subsystems serves 4 independent connections to the NoC. On the north- and south-sides of the chip, the I/O subsystems connect to external memory, while the I/O subsystems located on the east- and west-side of the chip are intended for ethernet communication. In order to provide the required performance, two independent NoCs are implemented on the Kalray MPPA®, the Data-NoC (D-NoC) and the Control-NoC (C-NoC). Both have the same architecture and differ only in the buffer size within the router, and flow regulation implemented on the D-NoC nodes. The C-NoC is intended for control messages, hence their payload is small. In contrast, the D-NoC is designed for transferring large payloads.

B. Switching Mechanism on the NoC

Wormhole switching is the switching mechanism on the NoC [27]. A NoC packet is divided into flow control digits (flits), where a flit has a fixed size of f bytes. Each link can transmit one flit every D_C clock cycles, where D_C is the link latency. In addition to the packets payload, a header flit is appended. The header contains the necessary information to route the message from source to destination node, i.e. static routing is applied. The number of header flits in a message is denoted as h. During transmission, the header propagates on its static route through the network. Once the header proceeds from one router to the next, the remaining flits follow in a pipelined manner.

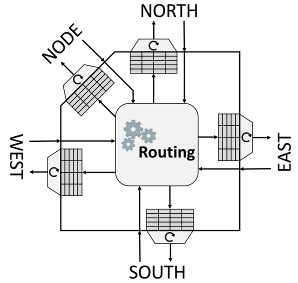


Fig. 2: Architecture of the NoC router on the Kalray MPPA® 256.

The architecture of the NoC router is shown in Fig. 2. Each output link has its own set of buffers. Each incoming link has its *own* dedicated buffer at *each* output link (i.e. the output link in direction *north* has separate buffers for the traffic from *west*, *south*, *east*, respectively, and from the connected *node*). When the header flit arrives at an input port, it experiences a switching delay of D_{SW} clock cycles before the flit is placed in the corresponding output buffer. However, this is a pipelined delay, so two flits that arrive D_C cycles apart at the input link will become available in the same inter-arrival time D_{SW} cycles later at the output buffer. Each of the buffers has a capacity of b_C flits on the C-NoC, and b_D flits on the D-NoC. In contrast to most NoC implementations, links on the Kalray-NoC *do not have a flow control mechanism* [9], [28]. This means a buffer can potentially overflow if more flits arrive than depart over a certain time period. The link arbitration between the different buffers at an output link is based on the *Round Robin* (RR) mechanism and works on packet level. This successfully (by design) prevents deadlocks in the system, i.e. the situation that packets on the NoC block each other in such a way that no packet can progress is eliminated due to the non-blocking links.

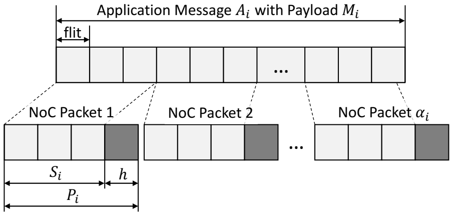


Fig. 3: Application message and the packetization into NoC packets.

C. Flow Regulation on the Source Node

To avoid buffer overflow and to guarantee service on the D-NoC, a flow regulation mechanism is implemented on each source node [8], [21]. A *packet shaper* and a *traffic limiter* work in tandem to guarantee the service on the NoC. This applies at connection level, i.e. messages of different NoC connections departing from the same node may have different flow regulation settings.

1) Packet Shaper: The packet shaper limits the size of NoC packets that are injected by a connection i to P_i flits. Since each packet has its own header, the effective payload is $S_i = P_i - h$ flits. An application message of size M_i flits is thus sent over the NoC via a series of $\alpha = \lceil M_i / S_i \rceil$ NoC packets. The relation between the application message and the NoC packets that are generated by the packet shaper before the packets are injected into the NoC is visualized in Fig. 3.

2) Traffic Limiter: The traffic limiter regulates the injected traffic. It is configured by two parameters, the *window size* T_W and the *bandwidth quota* β [8]. On each clock cycle the traffic limiter compares if the number of injected flits during the last T_W cycles *plus* the number of flits in the next pending packet is larger than β. If not, the complete packet is injected.

D. Application Model

An application A_i is mapped to a single compute tile of the many-core platform, and different tiles can possibly have different execution models. During execution, each application may *read* and *write* data to and from the off-chip memory. Such requests are performed in a sequential manner. During its execution, an application A_i can issue a set of read requests $\mathcal{R}_i$ and a set of write requests $\mathcal{W}_i$. Each request j has an associated payload size of M_j flits, which needs to be read or written from or to the I/O subsystem, respectively. In this work, application message refers to a message $\in \mathcal{R}_i$ or $\mathcal{W}_i$.

For a read request, a message is sent from the compute cluster to the I/O cluster on the C-NoC. Such a request message always has a fixed size of M_{CNoC} flits, which is not included in M_i. Once the I/O cluster fetches the requested data from off-chip memory, the data of size M_i flits is sent back to the compute cluster over the D-NoC. Write requests,

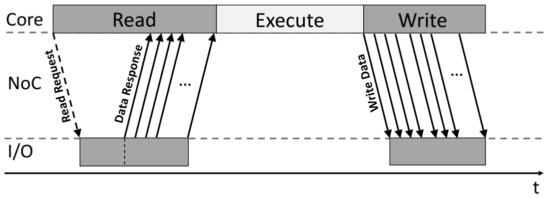


Fig. 4: Read-Execute-Write semantic and the resulting communication pattern.

on the other hand, do not need a C-NoC message, since the data can be directly sent over the D-NoC to the I/O subsystem. Both communication patterns are depicted in Fig. 4, where the dashed arrow represents communication on the C-NoC and the full arrow communication on the D-NoC.

No assumptions are made on the arrival times of read or write requests. This is the case for many industrial application models. For example, AUTOSAR [29] follow a coarse-grained memory access, divided into *read*, *execute*, and *write* phases [5], [30].

On the application side, the time that needs to be reserved for read or write messages is the time required for the complete transmission of the data over the NoC. This is to keep the memory coherent, since a C-NoC message can potentially overtake a D-NoC message. If the I/O subsystem is in the process of receiving data (i.e. a write request) and a C-NoC message requesting the same data arrives, it is possible that the partially written data plus old data is sent back, thereby compromising the data consistency. From the application perspective, the main objective is to minimize the communication times since this allows for a better utilization of the cluster.

IV. Contention-Aware NoC Partitioning

In an unconstrained NoC, a NoC packet can potentially suffer blocking on every router on its path from its source cluster to the I/O. This blocking on each router may be caused by contention from packets departing from the router to the same (direction) link from other clusters or the source cluster of the considered packet. Since there is a FIFO buffer at the output link for packets arriving from each direction, the considered packet incurs (intra-queue) blocking delays firstly, since it is queued up behind other packets in the corresponding output buffer. Secondly, further (inter-queue) delays are incurred due to the round robin arbitration between the different FIFO queues on the output link. The routing of NoC packets emerging from other compute clusters thus has large impact on the WCTT of all NoC packets, and hence on the off-chip memory access time. This also means that, even though compute clusters are independent entities, as soon as the NoC is accessed, other NoC traffic needs to be accounted for. While different methods to compute the WCTT are available for several NoC-types, all require a priori knowledge of NoC communication in order to derive the WCTT bounds. If routes are given, upper bounds can be computed for the Kalray MPPA® NoC, as shown in [8]. However, these bounds are, in most cases, overly pessimistic [11].

This section introduces a novel partitioning based on placement and routing rules for the NoC on the Kalray MPPA® with the objective to minimize the WCTT of messages over the NoC by significantly reducing the inter-cluster interference.

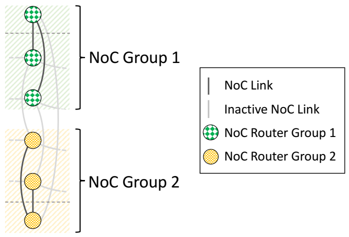


Fig. 5: Division of the NoC column into two independent groups in order to minimize the possible interference.

A. Effective NoC Sub-Topology

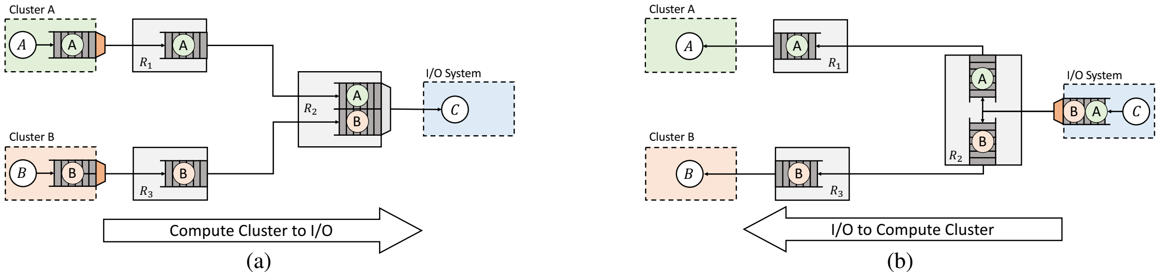
The application model described in Section III brings several advantages. The key idea of our approach is to *refrain from direct communication among applications residing in different clusters* and that all communication happens via the shared off-chip memory. With such a partitioning, there is *no horizontal* NoC traffic, i.e. the eastward and the westward links are not used for any communication. We later experimentally show in Section VII, that the benefits of contention-free communication outweigh and justify the intentional loss of these links on the NoC.

Given this setup, the proposed partitioning is based on the concept of *NoC Columns and NoC groups*. The entire NoC is logically divided vertically into four NoC columns, as shown in Fig. 1b. Each NoC column comprises one router of each DDR I/O subsystem and 4 routers connected to the 4 compute clusters in that column. Each NoC column is further divided into two *NoC Groups*, as shown in Fig. 5, each consisting of two compute clusters. We mandate that the two compute clusters in NoC group 1 in every NoC column access the off-chip memory through the I/O subsystem on the *north side* and the two compute clusters in NoC group 2 in every NoC column access the off-chip memory through the I/O subsystem on the *south side* of the chip, respectively[1]. Note that with this explicit well-defined spatial memory separation across the two NoC groups, there is no interference and direct communication between them. To guarantee the spatial isolation between the respective groups on the NoC, unused NoC links need to be disabled. This can be done on the Kalray MPPA® using *lockout bits*, which remove the connected links from the NoC topology until the system reset [31]. This is an important feature since it guarantees freedom of interference between NoC groups, even in case one group malfunctions.

The partitioning of the NoC into the proposed NoC groups has three benefits in terms of system design, contention minimization and predictable execution:

1) Each compute cluster has a *private* link to the router of the I/O subsystem, and shares only the final link with the second compute cluster of the NoC group. This also implicitly means no contention for the internal NoC links.
2) Each NoC group is *identical* in terms of NoC topology. This symmetry facilitates a uniform analysis applicable to all NoC groups.

[1] If all DMA controllers have access to both memories this is straightforward. Otherwise linker scripts can be used to place the code in the right memory banks to implicitly direct the memory access through the required DMA controller.



(a)

(b)

Fig. 6: All involved network elements of one NoC group for cluster to I/O messages are shown in (a) and for I/O to cluster messages are shown in (b).

3) Interference on the NoC is limited to the compute clusters within a group. This is the case since access to off-chip memory is predefined within the groups and no other communication takes place on the NoC. Thus, the introduced rules already limit the possible interference only to interference from a single compute cluster as opposed to 15 other clusters in the unconstrained system.

V. Computing the WCTT of NoC Messages

This section shows how to obtain the WCTT of application messages on the NoC. These calculations consider the architectural details of the COTS platform and are tailored to the partitioning previously described in Section IV. As seen in Fig. 6, the NoC elements that are shared among the two compute clusters are different for the two directions of communication. This mainly manifests itself in router R_2. The architecture in Fig. 6a performs round robin arbitration between the two buffers that contend for the same link, compared to Fig 6b where each buffer is located at a different output link. Thus, we analyze the two cases separately. Without loss of generality, we always analyze a message from cluster *A* that receives blocking by a message from cluster *B*.

First we present how to compute the WCTT of the last packet of the application message in isolation, defined as Basic Network Latency C_{BNL}. Given the pipelined transmission of packets on the NoC, and the fact that all packets of the application message are released at the same time, the total WCTT of the packet is then computed as C_{BNL} plus the blocking delays added by other packets from the same flow and blocking delays due to packets from other clusters. These calculations are presented for each type of message.

A. Basic Network Latency

The basic network latency constitutes the latency of *one* NoC packet when traveling in isolation, i.e. there are no other packets on the NoC. This latency is the same for all three types of messages that can be observed in the partitioned NoC, since the same number of links and routers are encountered in both directions (see Fig. 6). To calculate the basic network latency, the traversal time of the header flit must be computed. Additionally, the pipelined arrival of the remaining body flits needs to be considered.

Let P_{LP} be the size of the Last Packet of the application message A_i. P_{LP} can be computed as follows:

$$P_{LP} = M_{A,i} - ((\alpha_{A,i} - 1) \cdot S_A) + h \tag{1}$$

In order to transfer the complete application message payload $M_{A,i}$, we need $\alpha_{A,i}$ NoC packets. Thanks to the flow regulation, all but the last NoC packet have the complete packet payload S_A. Subtracting the payload of these complete packets from $M_{A,i}$ results in the payload of the last packet. A header is added to account for the total number of flits in the packet.

The header flit needs to traverse 3 links where it experiences 2 switching delays inside routers on the path, as shown in Fig. 6, before reaching its destination. The remaining flits then follow in a pipelined manner.

$$C_{BNL} = 2 \cdot D_{SW} + 3 \cdot D_C + (P_{LP} - 1) \cdot D_C \tag{2}$$

The traversal time of any packet can be computed in this way by replacing P_{LP} with the respective packet size.

B. Read from Memory Scenario

This subsection introduces the WCTT for the required messages, when a cluster reads data from the I/O subsystem.

1) Request Message on the C-NoC to Read Data from the I/O Subsystem: The request message from the compute cluster to the I/O subsystem travels on the C-NoC, which is intended for control messages. These messages have a fixed size of M_{CNOC} flits ($M_{CNOC} <= S_i$). Due to the NoC partitioning, there is at most one message of cluster B that can introduce blocking on the shared router (R_2). This means that the RC-based analysis for a round robin arbitrated NoC [11] can be applied here, where, in addition to the basic network latency, the blocking delay due to a request message of the other cluster is added. Since C-NoC messages can be sent in only one NoC packet, no more blocking is possible.

$$WCTT_{C-NoC} = M_{CNOC} \cdot D_C + C_{BNL} \tag{3}$$

2) Sending the Requested Data to the Compute Cluster: Communication from the I/O cluster to the compute cluster is shown in Fig. 6b. On the NoC, only the link between the I/O cluster and router R_2 is shared. However, both application messages are released from the same node, i.e. they can block each other before they are injected into the NoC, but they cannot block each other on the NoC itself. This guarantees that there can be no buffer overflow on the NoC routers, since packets travel without blocking. Hence, the I/O subsystem can safely inject packets without flow regulation.

This allows to divide the WCTT computation in two parts: 1) the release blocking B_{rel}, which is experienced due to contending messages inside the I/O cluster, and 2) the Network Latency (NL) of the message on the NoC, C_{NL}.

The network latency of an application message with payload $M_{A,i}$ can be easily computed.

$$C_{NL} = ((\alpha_{A_i} - 1) \cdot P_A \cdot D_C) + C_{BNL} \tag{4}$$

The first part of the equation denotes the sequential injection of $\alpha_{A_i} - 1$ full packets and their transmission over one

NoC link. This is equivalent to the blocking the last packet experiences before it is injected into the NoC. The second part denotes the traversal of the last NoC packet over the complete path.

The maximum blocking by a message of cluster B that can be suffered on the I/O cluster occurs if the largest application message to cluster B is scheduled right before the message under analysis. This means *all* NoC packets of the contending message need to be injected into the NoC before the packets of the message under analysis. Note, there can be only one active message per compute cluster. The largest blocking can then be computed assuming transmission of the largest message $M_{B,j}$ to cluster *B* over a single NoC link. Once all flits are injected into the NoC, they do not have any possibility to affect the message under analysis.

$$B_{rel} = ((\alpha_{B,j} \cdot h) + M_{B,j}) \cdot D_C \tag{5}$$

The $WCTT_{IO \to CC}$ can then be expressed as the summation:

$$WCTT_{IO \to CC} = C_{NL} + B_{rel} \tag{6}$$

C. Write to Memory Scenario

When sending data from the compute cluster to the I/O subsystem (see Fig. 6a), the traversal time of an application message is affected by blocking in the shared router, but also by the flow regulation on the source node. Since packets might be blocked in the shared router, and hence accumulate, flow regulation is required to guarantee that the buffer on the router does not overflow.

The objective is to determine the WCTT of an application message of payload $M_{A,i}$ from the cluster *A* to the I/O subsystem *C*, while cluster *B* introduces the maximum blocking, i.e. there is *always* a packet of *B* contending for the shared link. Further, it is assumed that the inter-arrival time of consecutive messages of cluster *A* are sufficiently apart, such that the buffer in R_2 is empty when a new transmission starts, i.e. the preceding message arrived at its destination. This can be a pessimistic assumption, but it improves the composability because it allows to compute the WCTT of messages of cluster A without knowledge of the properties of messages of cluster B or earlier messages sent by cluster A.

1) Desired Traffic Regulation: Different flow regulation settings impact the resulting WCTT of messages and also the accumulation of flits in the buffer of R_2 during the transmission. Fig. 7 presents the largest traversal time of a message of size 10000 flits, that is sent from cluster A to the I/O subsystem, when maximum contention is present on the router R_2. This is shown for varying flow regulation settings β. The results of Fig. 7 are based on a cycle accurate simulator of the NoC group [2]. It can be observed, that the WCTT decreases with increasing β, up to a point when the WCTT value does not change anymore, i.e. further increasing β does not have an impact on the WCTT. The smallest setting for β that results in this minimum WCTT yields a lower bound, which is defined as β_{min}. Additionally, the same figure depicts the largest observed buffer occupation in R_2 during the transmission of the message. If the buffer requirement of a message under a certain setting of β is larger than the available buffer, a buffer overflow is caused. Thus, the intersection of the largest buffer occupation and the actual buffer capacity (in the figure shown as 3600 flits), shows β_{max}, the upper bound of β. The non linear behavior in both curves is caused by the flow regulation that operates on packet granularity, while β is expressed as the number of flits.

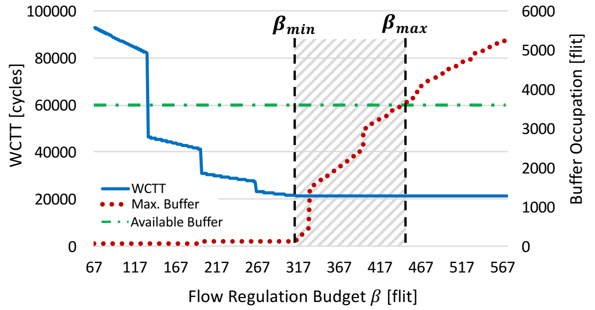


Fig. 7: WCTT and max. buffer occupation for the same message sent under different flow regulation settings. β_{min} and β_{max} are additionally shown.

Several conditions must be fulfilled in order for β to be in the desired range between β_{min} and β_{max}. First the conditions to achieve the minimum WCTT are presented, when $\beta \geq \beta_{min}$. In [20], it is shown that the worst-case scenario for the traversal time of packets on a round robin arbitrated NoC occurs if the header of contending packets arrive simultaneously at the shared router and the respective other packet wins the round robin arbitration. Due to the flow regulation on source node it must also hold:

Lemma 1: In order to obtain the $WCTT_{CC \to IO}$ of an application message $M_{A,i}$, β must be configured such that the buffer in R_2 always holds at least the header flit of a NoC packet of $M_{A,i}$ in the cycle after a contending packet of cluster *B* finished its departure, for all packets of $M_{A,i}$.

Proof 1: This directly follows from the property in [20]. If a packet of cluster *B* finished its transmission and in the next cycle there is no packet of $M_{A,i}$ ready (i.e. at least its header flit arrived in R_2), then there is no data transfer in this cycle. As shown in [20], the worst case then occurs when the next packets of cluster *B* and *A*, respectively, arrive simultaneously, where arbitration is given to cluster *B*. Thus, the additional empty cycle increases the WCTT (see Fig. 7 for $\beta < \beta_{min}$). □

Given Lemma 1, in Section VI-A we will show how to compute the value of β_{min}. Additionally, the limited buffer in the router requires an extra condition for the maximum value of β.

Lemma 2: In order to obtain the $WCTT_{CC \to IO}$ of an application message $M_{A,i}$, β must be configured such that the number of flits that accumulate in the buffer on the shared router R_2 (due to the RR arbitration with packets of cluster *B* at the output link) must always be smaller or equal to the buffer capacity b_D. Otherwise, packets can be dropped.

Proof 2: This directly follows from the hardware architecture. Since no link-level flow control is implemented, a flit is lost if it arrives while the buffer is already full (see Fig. 7 for $\beta > \beta_{max}$). □

Given Lemma 2, in Section VI-B we will show how to

[2] See Section VII-E for more details on the simulator used.

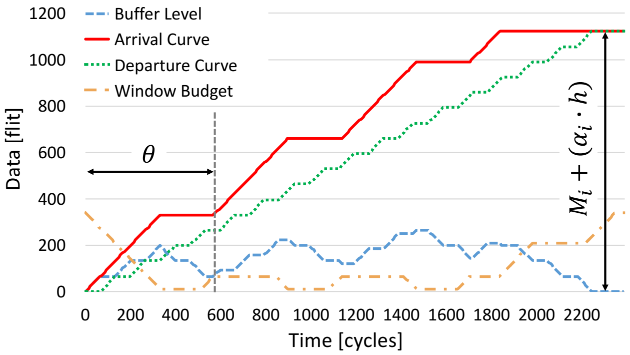


Fig. 8: Arrival and departure curve as seen from the buffer in R_2. Additionally the window budget and the buffer level are shown.

compute the value for β_{max}.

Now, assuming that β is in the desired range (between β_{min} and β_{max}), in the remainder of this section we present how to compute the $WCTT_{CC\to IO}$.

2) Arrival and Departure Patterns under Traffic Shaping: As discussed, many involved elements do not contribute to the blocking experienced by the sequence of NoC packets (such as the links or router R_1). They just introduce an offset, but do not change the pattern of the arrival and departure curves on the respective NoC elements. Thus, in this subsection only the arrival and departure patterns of the buffer in R_2 are considered. Fig. 8 depicts an example scenario, based on analysis, where the arrival curve (as produced by the traffic shaper in node A) and the departure curve (as a result of the blocking with packets of cluster B) are shown. Additionally, the *buffer level* is shown (computed by subtracting the departure curve from the arrival curve), and the *window budget* which is defined as β minus the number of flits that were injected during the last T_W cycles. Note that vertical axis in the graph represents buffer space and window budget, whereas horizontal axis represents time.

3) Computing the $WCTT_{CC\to IO}$: This section shows how to compute the WCTT of the sequence of packets resulting in the transmission of the application message of size $M_{A,i}$ under flow regulation settings $\beta_{min} \leq \beta \leq \beta_{max}$.

$\alpha_{A,i}$ packets are needed to fully transmit $M_{A,i}$ payload flits. The $WCTT_{CC\to IO}$ of A_i then constitutes the basic transmission time of the last packet, C_{BNL}, the blocking it receives in R_2, as well as the blocking experienced due to the traffic shaping before it is injected into the NoC.

Lemma 3: If the traffic shaping is configured such that $\beta_{min} \leq \beta \leq \beta_{max}$, then the blocking experienced by the $\alpha_{A,i}$ packets is independent of the flow regulation and solely depends on the departure curve of R_2.

Proof 3: Due to Lemma 1 it is known that during the transmission of the message, there is always a data transfer on the output link of R_2. The flow regulation injects packets at least with the same rate as they depart from R_2. Thus, the transmission is constrained by the departure of packets from R_2.□

Based on Lemma 3, the $WCTT_{CC\to IO}$ can then be expressed by:

$$WCTT_{CC\to IO} = (P_B + P_A) \cdot (\alpha_{A,i} - 1) + P_B + C_{BNL} \tag{7}$$

The first part of the equation constitutes the time to transmit $\alpha_{A,i} - 1$ complete packets. The additional blocking P_B which is experienced by the last packet is added, as well as the basic network latency of the last packet in isolation, C_{BNL}.

VI. Selecting the Flow Regulation Parameters

The flow regulation on source nodes is one of the key components of the architecture. Correct configuration guarantees that no buffer overflow will occur despite the missing flow control on NoC links [9]. The same flow regulation settings are used for all messages sent by a compute cluster. Without loss of generality, the selection of the flow regulation parameters is shown for compute cluster A. Two parameters are used to configure the flow regulation, the window size T_W in clock cycles and the bandwidth quota β in flits[3]. Only the second parameter is configurable.

Lemma 1 and 2 give us the two necessary conditions for the selection of β.

1) Prevent buffer under-utilization (underrun): During the transmission of the application payload, the buffer level is never 0 for more than one consecutive cycle.

2) Prevent buffer overflow: During the transmission of the application payload, the utilized buffer level is never greater than its capacity b.

In general, β can be selected from the interval $[P_A, T_W + P_A]$. If $\beta = P_A$ only one packet is injected every transmission burst and if $\beta = T_W + P_A$ an unlimited number of packets can be injected, i.e. no flow regulation is applied. However, both extremes may violate Lemmas 1 and 2. Hence in the remainder of this section, it is shown how to find valid bounds, β_{min} and β_{max}, such that Lemmas 1 and 2 are satisfied.

A. Determining β_{min}

The injection of one application message of size $M_{A,i}$ into the NoC can be divided into inter-arrival times of bursts θ (as illustrated in Fig. 8), where θ is the time between two consecutive bursts. θ constitutes the injection of a complete burst of NoC packets plus the waiting period enforced by the traffic limiter before the next burst can be injected into the NoC. Note that this holds for all but the last segment that only has a transfer time but no waiting time.

The number of NoC packets that can be injected in one burst can be computed as $\alpha_{burst} = \lfloor \beta / P_A \rfloor$. The floor brackets need to be applied since the traffic limiter injects a new packet only if the window budget is larger than or equal to the packet size.

Due to the flow regulation, at the end of the interval θ, the window budget is equal to the packet size. Hence, the next departure interval can start. The length of θ is dependent on the packet size of the sending cluster and the flow regulation parameters.

$$\theta = T_W + P_A - (\beta - \alpha_{burst} \cdot P_A) \tag{8}$$

The first part of the equation reflects the ideal injection interval. The second part accounts for the packetisation effects that prevent the injection of partial packets once the window budget is not large enough for a complete packet.

[3] To simplify the presentation in this section it is assumed that $D_C = 1$ (as it is on the Kalray MPPA®), i.e. one flit can be transmitted each clock cycle. For a general case, all T_W can be substituted with $\lfloor T_W / D_C \rfloor$.

The minimum valid setting for β, β_{min}, is achieved once the buffer level reaches 0 exactly at the end of θ. In one burst, $\alpha_{burst,A} \cdot P_A$ flits are injected over θ cycles. Due to the RR arbitration with packets of cluster B, it is also known that every $P_A + P_B$ cycles P_A flits departed from the buffer. Hence, to have a buffer level of 0, the value of β must be selected in a way such that the following equation holds.

$$\frac{\theta}{P_A + P_B} \cdot P_A = \alpha_{burst} \cdot P_A \tag{9}$$

The left side of the equation represents the number of flits departing from the buffer in θ, and the right side represents the number of flits that arrive in θ. Note that here θ is always a multiple of $P_A + P_B$, since we assume the buffer reaches 0 exactly at the end of one departure segment. By substituting θ and $\alpha_{burst,A}$ the following equation can be derived:

$$T_W + P_A = \left\lfloor \frac{\beta}{P_A} \right\rfloor \cdot P_B + \beta \tag{10}$$

Due to the floor function there are several values for the packet size P_A, where no exact solution exists. Since a larger θ would violate Lemma 1, a smaller value for θ is chosen. This, however, results in the buffer not being empty at the end of each θ, thus the buffer occupancy grows monotonically at these points. The updated equation is shown below:

$$T_W + P_A \leq \left\lfloor \frac{\beta}{P_A} \right\rfloor \cdot P_B + \beta \tag{11}$$

The objective now is to find the minimum value for β that satisfies the equation. This can for example be done using a binary search in the small search space of $[P_A, T_W + P_A]$.

B. Determining β_{max}

To provide an upper bound on the possible values for β, it must be guaranteed that the buffer level never exceeds the buffer capacity b_D (see Lemma 2). The buffer occupancy at time t is defined by the difference between the flits that arrived and the flits that departed up to this time. First the function dep(t) is introduced, that computes the number of flits that departed from the buffer in R_2 up to time t, and later arr(t), that computes the number of flits that the flow regulation injected into the NoC up to time t. Both functions assume that $\beta \geq \beta_{min}$.

1) Departure Curve from R_2: The function dep(t) can be computed in several steps. If there is always interference on the router, each packet of cluster A takes $(P_A + P_B) \cdot D_C$ cycles to transmit. One such transmission can then be divided into the delay encountered, *delay interval*, and the transmission time, *send interval*. The number of complete send intervals s_{dep}, before time t, can be computed by $\left\lfloor \frac{t}{P_A+P_B} \right\rfloor$. Similarly, the number of complete delay intervals d_{dep} can be computed by $\left\lceil \frac{t-P_B}{P_A+P_B} \right\rceil$. The function then consists of two parts:

$$\text{dep}(t) = s_{dep} \cdot P_A + (s_{dep} - d_{dep}) \cdot \Delta t_{dep} \tag{12}$$

The first part of the equation represents the complete send intervals before time t, where in each send interval one NoC packet of size P_A is transmitted. The second part accounts for a send interval that might be in the process of transmission, hence not necessarily the complete packet left the buffer. Since this is only required if t overlaps with a send interval the value is multiplied with $s - d$. This evaluates either to 0 (in case t overlaps with the delay interval), or to 1 (in the case t overlaps with the send interval). Δt_{dep} can easily be computed by subtracting the time spent for the complete send and idle intervals from time t.

$$\Delta t_{dep} = t - (d_{dep} \cdot P_B + s_{dep} \cdot P_i) \tag{13}$$

2) Arrival Curve in R_2: In a similar way, it is possible to describe the arrival function. During each θ, the sending node first injects a complete burst of α_{burst} packets, before the flow regulation stops further packets from being injected into the NoC. Thus, we define the injection time of the complete burst as *send interval*, and the following idle time as *delay interval*. The number of send intervals from the compute cluster s_{arr} can be described as $\left\lceil \frac{t-(\alpha_{brust} \cdot P_A)}{\theta} \right\rceil$. The number of delay intervals d_{arr} can be described as $\left\lfloor \frac{t}{\theta} \right\rfloor$. With these values it is now possible to compute Δt_{arr}, and $arr(t)$:

$$\Delta t_{arr} = t - (s_{arr} \cdot \alpha_{burst} \cdot P_A + d_{arr} \cdot (T_W + P_A + N_{max})) \tag{14}$$

$$arr(t) = s_{arr} \cdot \alpha_{burst} \cdot P_A - (s_{arr} - (d_{arr} + 1)) \cdot \Delta t_{arr} \tag{15}$$

The above calculations are independent of the actual size of the injected message. To include the actual size of a message A_i an additional step can be added:

$$arr(t, A_i) = \max(arr(t), M_{A,i} + \alpha_{A,i} \cdot h) \tag{16}$$

3) Buffer Occupancy: The maximum buffer occupancy is reached during the transmission:

$$b_{max}(A_i) = \max_{t \in [1, WCTT_{A_i}]} (dep(t) - arr(t, A_i)) \tag{17}$$

4) Computing the Maximum β: The largest message that is sent from the compute cluster results in the largest buffer occupancy during its transmission. This is because the buffer is empty at the start of the transmission, and the flow regulation parameters are selected at the cluster level, i.e. all messages are sent using the same settings. $b_{max}(A_{i,max})$ then provides the largest buffer occupation for any of the messages sent by the compute cluster, where $b_{max} = \max(M_{A,i} \in \mathcal{W}_i)$. Binary search can then be used to efficiently determine the maximum valid setting for β_{max}, since the search space is reduced to $\beta_{max} \in [\beta_{min}, T_W + P_A]$.

VII. Evaluation

In this section, we evaluate the proposed NoC partitioning as well as its tailored analysis based on synthetic experiments and experiments on the Kalray MPPA® platform. In addition, the low pessimism of the analysis is demonstrated through a case-study on a cycle-accurate simulator of one NoC partition.

A. Experiment Setup

For all synthetic experiments and the experiments based on the simulator, we select the hardware parameters according to the parameters of the target COTS platform, the Kalray MPPA®. The buffer size for the NoC router is set to 401 flits [22], the packet payload for D-NoC messages is set to $S_A = S_B = 62$ flits, and the header has a size of $h = 4$ flits. It is further assumed that $D_C = 1$ cycle and $D_{SW} = 1$ cycle [8]. The window size T_W is considered to be constant at 512 cycles [22]. Messages on the C-NoC have a fixed payload of 2 flits [32].

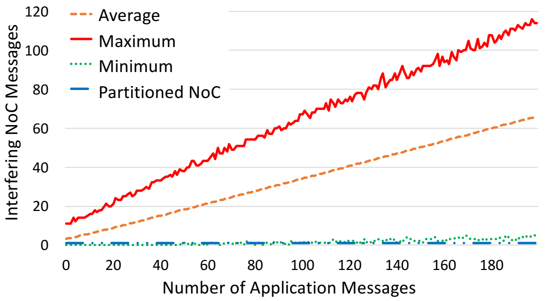


Fig. 9: Number of interfering messages on the NoC for a varying number of application messages and one message per application to the off-chip memory.

B. Interfering Messages on the NoC

In an unpartitioned NoC, messages may share part of their route with other messages and thereby face contention delays. The number of interfering messages, the payload size of the messages, the routing, as well as the arbitration protocol implemented by the NoC routers affect the blocking delay that a message may incur [17].

This experiment uses the number of interfering messages as performance metric, where a message can interfere only when it shares at least one link with the message under analysis. One application is randomly mapped to each compute cluster of the complete NoC topology (see Fig. 1a). Each application is randomly assigned to one of the 8 nodes connecting to the off-chip memory (on the I/O subsystem on the North- and South-side). In addition, applications may communicate directly with each other. On the selected hardware platform, both communication messages and messages to access off-chip memory travel on the D-NoC (in addition read requests are sent on the C-NoC). For such communication, unique source and destination node pairs are randomly selected out of the compute clusters. Routing on the torus-based NoC in Kalray MPPA® is not trivial since the implementation details of the exact routing are not revealed publicly by the vendors. However, NoC packets follow the shortest path and use source routing [8], i.e. the path is calculated at design time and encoded in the header flit. For the experiment, routes were generated using Dijkstras shortest path algorithm [33]. The generated routes are then analyzed to determine the total number of interfering messages that each message may encounter given the concrete mapping. For each data point, 1000 mappings were generated.

Fig. 9 presents the resulting average, minimum, and maximum number of messages on the NoC contending with a single NoC message, as the number of application to application messages is varied. In addition, the number of interfering messages on the partitioned NoC is shown. For the case without messages sent between applications (0 on the x-axis), the interfering messages in unconstrained mappings are larger, on average 3.17 interfering messages, than for the partitioned NoC, where only one interfering message is possible. The benefits of partitioning becomes further apparent as the number of application messages increases. For 200 application messages, the number increases to an average of 66 interfering messages. This experiment clearly shows that the proposed partitioning significantly reduces the interference on the NoC.

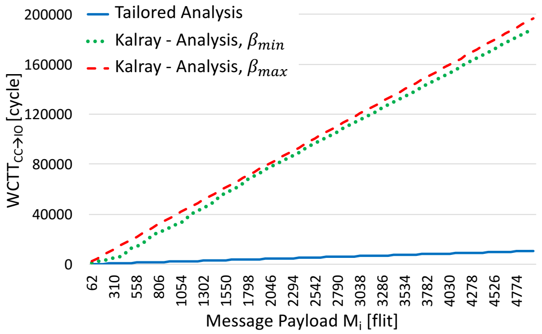


Fig. 10: WCTT of messages with varying payload sent from compute cluster to the I/O subsystem. Results are shown for our proposed analysis and the Kalray analysis with both, β_{min} and β_{max}.

C. Latency Analysis on the D-NoC

This experiment compares the $WCTT_{CC \rightarrow IO}$ computed by the proposed tailored timing analysis and the State-of–the-Art timing analysis of Kalray [8], [9]. For both analysis methods, the partitioned setting is assumed, i.e. messages travel 3 hops from the compute cluster to the I/O subsystem. Fig. 10 presents the results for varying application message payloads M_i, for our proposed timing analysis and also the analysis by Kalray for the MPPA® [8], [9]. Since the Kalray analysis requires the flow regulation parameters, one curve is shown for β_{min} and β_{max}, respectively. A large difference between the two methods can be observed. This is mostly because the Kalray analysis does not take the stream as a whole into account, but rather computes delays on packet basis. This shows the strength of the proposed timing analysis. Reducing the pessimism in the analysis allows for a higher utilization of the hardware platform.

D. Total Memory Read Latency on the MPPA®

This experiment compares the memory latency experienced on a compute cluster when reading data from external memory, for the proposed partitioned model (2 clusters) against unpartitioned settings (4, 8, and 16 clusters). The experiment is performed on the Kalray MPPA® platform. In this experiment, each of the compute clusters is constantly reading memory from the I/O subsystem in a range of $[1, 16]$ KB. In the unpartitioned scenario, different settings are shown. The naive case, where all 16 clusters read the memory via the same core in the I/O subsystem, the case where only the 8 clusters on the north side of the processor read memory via the same core of the I/O subsystem, and the case where only 4 clusters read memory via the same core of the I/O subsystem. This naturally leads to varying contention on the NoC, as well as on the access path to the memory. In the partitioned scenario only one partition is active (cluster 0 and cluster 4). Hence, there is no interference from other compute clusters on the NoC. While in a fully partitioned scenario, the clusters and the I/O subsystem do not share any resources, the off-chip memory is shared amongst them. This may lead to additional delays depending on the memory organization [7].

The access to external memory is implemented using the channel communication paradigm of the Kalray over the NoC [8], [32]. The compute clusters initiate the data transfer by sending a request message over the C-NoC. A dedicated listener task on the core of the I/O subsystem handles the

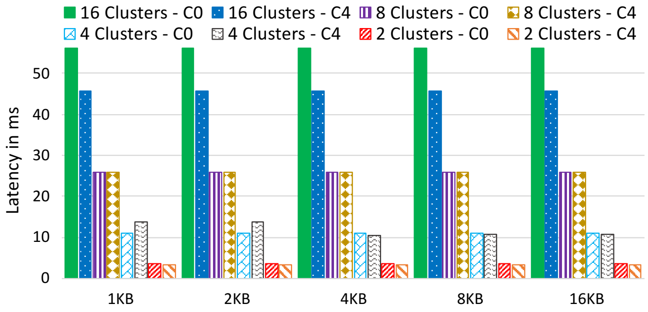


Fig. 11: Comparison of max. observed memory access latency in the partitioned platform (2 clusters), and unpartitioned platform (4, 8, and 16 clusters).

incoming request and sends the requested data back to the compute cluster. The presented memory access latencies are measured on the compute clusters, starting from the C-NoC message being issued until the complete data arrived over the D-NoC. For each data point, 10000 samples were collected.

Fig. 11 presents the experimental results. The maximum observed read latency is reported for the two compute clusters (0 and 4) of the NoC partition (in all scenarios). While the maximum observed read latencies increase only slightly with increasing data size, the main impact on latency is caused by interference from other compute clusters. Note that this not only includes the latencies on the NoC, but also scheduling latencies on the core of the I/O subsystem, as well as latencies on the memory access path. The benefit of the partitioned NoC compared to the unpartitioned NoC is clearly visible.

E. Case Study

The case study is based on an Engine Management System (EMS), originally presented in [34]. The application consists of 15 runnables (the elementary unit of execution in AUTOSAR), M1 to M15, which are triggered periodically at a frequency of $5\,\mathrm{ms}$, $10\,\mathrm{ms}$, $20\,\mathrm{ms}$ and $100\,\mathrm{ms}$, respectively.

[34] further reports the footprint of the runnables code and all its private variables in memory, which is in the range of $[7076, 17424]$ bytes. Table I presents the parameters of interest for all the 15 runnables. Within each compute cluster, we use the read-execute-write communication semantic paradigm of [5], wherein all runnables' code and all their private variables are prefetched from off-chip memory before each periodic execution. After the execution, the code including all its private variables are written back to the off-chip memory. Both phases are scheduled in a time-triggered manner, which makes their release independent of the runnables' execution time on the compute cluster. Due to the independence of NoC-groups, only one group is considered. In this group, each compute cluster schedules one instance of this case study.

Hardware parameters are chosen in line with the Kalray MPPA®-256. The compute cores, as well as the NoC, are clocked with a frequency of $400\,\mathrm{MHz}$. The flow regulation value β is set to $\beta_{min} = 314$ flits, as computed by the approach presented in Section VI. This leads to an empty buffer after each departure segment.

1) Memory Request Handling within the I/O Subsystem: The delay between the C-NoC request message and the corresponding D-NoC message response depends on how messages are handled within the I/O subsystem. On the Kalray MPPA®, access to DDR3 SDRAM memory on the I/O subsystem is managed by a memory request arbiter [7]. The arbiter consist of a Multi Port Front End (MPFE) that in turn connects to a Reorder Core (RC). The different memory masters, such as the four RM cores, connect to the MPFE, where each master has an assigned priority. The MPFE forwards the requests of the memory masters based on their priority to the RC, where the requests are stored in a queue. A Starvation Counter (SC) is implemented to boost the priority of memory masters if a pending request is not forwarded within a certain time window. To issue requests, the RC selects the next request from within its queue, based on a set of rules (as described in [7]), before issuing it to the DDR3 SDRAM controller.

TABLE I: Engine Management System – Case Study.

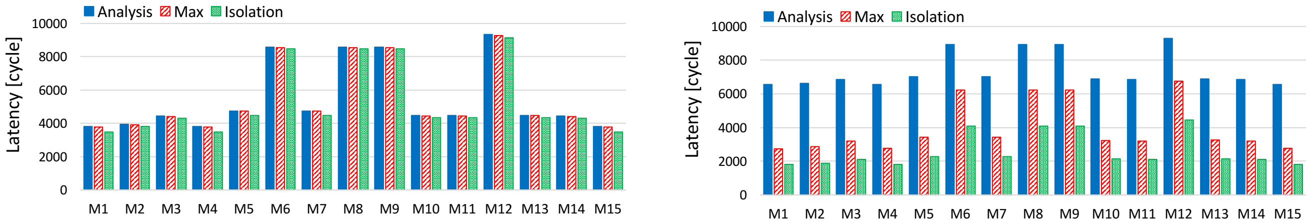
Short	Name	Size [byte]	Period [ms]
M1	MassAirFlowSWCEntity	7076	5
M2	ThrottleSensSWCEntity	7352	5
M3	APedSensor	8286	5
M4	APedVoterSWCEntity	7104	10
M5	ThrottleCtrlEntity	8868	10
M6	ThrottleActuatorEntity	16058	10
M7	BaseFuelMassEntity	8868	10
M8	ThrottleChangeSWCEntity	16058	10
M9	TransFuelMassSWCEntity	16058	10
M10	IgnitionSCWEntity	8348	10
M11	TotalFuelMassSWCEntity	8308	10
M12	OperatingModeSWCEntity	17424	20
M13	IdleSpeedCtrlSWCEntity	8372	20
M14	APedSensorDiag	8286	100
M15	InjBattVoltCorrSWC	7116	100

As discussed in [7], [24], interference between memory accesses of different compute clusters can be minimized by mapping the cluster data to exclusive memory banks of the DDR3 SDRAM (i.e. each of the 16 compute clusters has a dedicated memory bank for its data). Further, all memory masters are assigned the same priority and the SC is disabled. This configuration results in consecutive requests to the same memory bank being preferred, hence, the total access to the DDR3 SDRAM is performed in sequence, before requests of other compute clusters are served. This arbitration process incurs a constant latency of 22 cycles [23].

Requests of one compute cluster, hence, arrive sequentially at the DDR3-SDRAM, where the first request suffers a page miss, followed by consecutive page hits. After the page size of 2 KB is reached, an additional page miss is encountered. The latencies for page hits and page misses are chosen for a DDR3-1600G SDRAM, where a page miss has 40 cycles latency for 64 bytes fetched, and a page miss 4 cycles respectively.

2) Means of Observation: In addition to the proposed timing analysis, experiments are performed on a cycle-accurate simulator[4] of the NoC in one NoC group. The simulation further models the memory access as it results from the

[4] The source code of the simulator is available at http://www.idt.mdh.se/personal/mbr04/RTAS17/Simulation.zip



(a) NoC latencies for writing data to off-chip memory ($WCTT_{CC \to IO}$).

(b) Reading data from off-chip memory. ($WCTT_{C-NoC} + WCTT_{IO \to CC}$).

Fig. 12: Case study results for NoC delays in off-chip memory access from compute cluster within one NoC group.

memory request handling, as described in Section VII-E1. Simulation results are recorded for 600 ms, i.e. 6 hyperperiods of the EMS, which is sufficient given the time-triggered nature of the traffic towards the I/O subsystem. The simulation results capture the maximum observed WCTT, as well as the maximum observed WCTT in isolation, where messages are only sent from one cluster and hence do not encounter any interference on the NoC.

3) Analysis Time: During the experiments, the analysis time was measured. The analysis was performed 1000 times to collect representative values. This experiment was performed on a system containing an Intel i7 CPU (4 cores at 2.8 GHz), and 16 GB of RAM. The average measured analysis time for $WCTT_{CC \to IO}$ is 333 ns. Similarly, the average measured analysis time for $WCTT_{C-NoC} + WCTT_{CC \to IO}$ is 895 ns. All presented equations have constant complexity. This makes the WCTT calculation fast and applicable for large design-space explorations.

4) Write to Memory Scenario: Fig. 12a depicts the results of the experiment for NoC messages sent during write access to off-chip memory ($WCTT_{CC \to IO}$). The largest observed difference between the analysis and the maximum observed value of the simulation is 46 cycles (message M12), while the minimum difference is 15 cycles (M5-9). In contrast, flows in isolation perform similarly, with a maximum difference of 330 cycles (M1), and a minimum difference of 81 cycles (M6). This is the case, since β is configured to its minimum value β_{min}, yielding an empty buffer after each departure segment. Such a configuration has the benefit that messages experience only minimal jitter. The experiments also show that the computed analysis bounds are conservative and rather tight.

5) Read from Memory Scenario: Fig. 12b depicts the results for NoC messages sent during a read access to off-chip memory ($WCTT_{C-NoC} + WCTT_{CC \to IO}$). The results show a larger gap between analysis results and actually observed values during the simulation. This is the case, since the worst-case results if the largest application message of the respective other cluster is encountered in the FIFO buffer of the I/O subsystem just before the message is injected into the NoC. This can clearly be seen in the observed values for messages in isolation. The large maximum payload size of 17424 bytes (M12) leads to a maximum blocking of 4644 clock cycles in the FIFO buffer of the I/O subsystem. The minimum difference between the maximum observed value and the analysis result is 2540 cycles (M12). The results show that the analysis bounds for the messages involved in reading data from the I/O subsystem are conservative and tight, i.e. system resources are not over-provisioned.

VIII. Conclusions

The shared NoC is one of the main sources of interference on COTS many-core platforms. This paper presents a novel NoC partitioning strategy for identical NoC groups that reduces the possible interference a NoC message may encounter. NoC groups do not share resources, thereby facilitating independent analysis. A dedicated analysis is proposed for reading and writing data to off-chip memory from within a NoC group. Additionally, it is shown how to select the traffic regulation parameters for each source node in a way that the buffer on the NoC router never overflows. While the methods presented in this paper are tailored to the MPPA® many-core processor, the same principles to reduce the contention on the NoC can be transferred to other platforms.

We experimentally evaluate the proposed NoC partitioning and show that contention is significantly lower in partitioned systems, and that the pessimism in the analysis is heavily reduced compared to state of the art techniques. Finally, a case-study is presented to show the applicability to industrial applications as well as the low pessimism of the analysis compared to the simulation results.

Future work will focus on scheduling of memory accesses within the I/O subsystems. While methods, such as explicit tasks that handle requests, are simple to implement, they introduce additional latencies which ultimatively impact the memory access latencies. More elaborate approaches, using the provided hardware support of the MPPA®, such as micro-code engines for asynchronous data transfer over the D-NoC, may reduce the latencies in the I/O subsystem.

Acknowledgement

The work in this paper is partially supported by the Swedish Knowledge Foundation within the projects PREMISE and DPAC, as well as by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); also by the EuropeanUnion under the Seventh Framework Programme (FP7/2007-2013), grant agreement nr. 611016 (P-SOCRATES).

References

[1] Freescale, *Future Advances in Body Electronics*, 2013. Last access October 2016, available at http://cache.freescale.com/files/automotive/doc/white_paper/BODYDELECTRWP.pdf.

[2] Roland Berger Strategy Consultants, *Need for Consolidation in Vehicle Electronics*, 2015. Last access October 2016, available at http://www.greencarcongress.com/2015/07/20150729-berger.html.

[3] P. Gai and M. Violante, "Automotive embedded software architecture in the multi-core age," in *21th IEEE European Test Symposium (ETS)*, 2016, pp. 1–8.

[4] Kalray Inc., "MPPA processors for autonomous driving," Tech. Rep., 2016.

[5] M. Becker, D. Dasari, B. Nicolic, B. Åkesson, V. Nélis, and T. Nolte, "Contention-free execution of automotive applications on a clustered many-core platform," in *28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 14–24.

[6] D. Dasari, B. Akesson, V. Nelis, M. Awan, and S. Petters, "Identifying the sources of unpredictability in COTS-based multicore systems," in *8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2013, pp. 39–48.

[7] Q. Perret, P. Maurere, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet, "Predictable Composition of Memory Accesses on Manycore Processors," in *European Congress on Embedded Real-Time Software (ERTS)*, 2016.

[8] B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *Conference on Design, Automation & Test in Europe (DATE)*, 2014, pp. 97:1–97:6.

[9] B. D. de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti, "Guaranteed services of the NoC of a manycore processor," in *the International Workshop on Network on Chip Architectures (NoCArc)*, 2014, pp. 11–16.

[10] M. Liu, M. Becker, M. Behnam, and T. Nolte, "Buffer-aware analysis for worst-case traversal time of real-time traffic over RRA-based NoCs," in *25th Euromicro Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 2017.

[11] H. Ayed, J. Ermont, J.-L. Scharbarg, and C. Fraboul, "Towards a unified approach for worst-case analysis of Tilera-like and Kalray-like NoC architectures," in *12th IEEE World Conference on Factory Communication Systems (WFCS)*, 2016.

[12] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: concepts, architectures, and implementations," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.

[13] A. Hansson, M. Subburaman, and K. Goossens, "Aelite: A flit-synchronous network on chip with composable and predictable services," in *Conference on Design, Automation Test in Europe (DATE)*, 2009, pp. 250–255.

[14] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. Müller, K. Goossens, and J. Sparsø, "Argo: A real-time network-on-chip architecture with an efficient GALS implementation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 2, pp. 479–492, 2016.

[15] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip," in *Conference on Design, Automation and Test in Europe (DATE)*, 2004, pp. 20 890–.

[16] *Tile Processor Architecture Overview for the TILEPro Series*, last access October 2015, available at http://www.tilera.com/scm/docs/UG120-Architecture-Overview-TILEPro.pdf.

[17] T. Ferrandiz, F. Frances, and C. Fraboul, "A method of computation for worst-case delay analysis on spacewire networks," in *4th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2009, pp. 19–27.

[18] D. Dasari, B. Nikolić, V. Nélis, and S. M. Petters, "NoC contention analysis using a branch-and-prune algorithm," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 3s, pp. 113:1–113:26, 2014.

[19] M. Liu, M. Becker, M. Behnam, and T. Nolte, "A tighter recursive calculus to compute the worst-case traversal time of real-time traffic over NoCs," in *22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017.

[20] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul, "Wormhole networks properties and their use for optimizing worst case delay analysis of many-cores," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2015, pp. 1–10.

[21] Z. Lu, M. Millberg, A. Jantsch, A. Bruce, P. van der Wolf, and T. Henriksson, "Flow regulation for on-chip communication," in *Conference on Design, Automation Test in Europe (DATE)*, April 2009, pp. 578–581.

[22] W. Puffitsch, R. B. Sørensen, and M. Schoeberl, "Time-division multiplexing vs network calculus: A comparison," in *23rd International Conference on Real Time and Networks Systems (RTNS)*, 2015, pp. 289–296.

[23] G. Giannopoulou, N. Stoimenov, P. Huang, L. Thiele, and B. D. de Dinechin, "Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources," *Real-Time Systems*, vol. 52, no. 4, pp. 399–449, 2016.

[24] Q. Perret, P. Maurere, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet, "Temporal isolation of hard real-time applications on many-core processors," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, pp. 1–11.

[25] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[26] *Intel Single Chip Cloud Computer*, last access October 2015, available at www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-article.pdf.

[27] L. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer Journal*, vol. 26, no. 2, pp. 62–76, 1993.

[28] S. Chattopadhyay, L. K. Chong, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk, "A unified WCET analysis framework for multicore platforms," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4s, pp. 124:1–124:29, 2014.

[29] *AUTOSAR*, last access October 2016, available at www.autosar.org.

[30] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[31] Benoît Dupont de Dinechin, "Guaranteed Services of the NoC and DDR Memory of the MPPA Processor," in *Keynote at the 14th International Workshop on Real-Time Networks (RTN)*, 2015, available at http://ecrts.eit.uni-kl.de/fileadmin/wwwadmin/workshop_layout/rtn2016/RTN2016_fichiers/kalray_rtn2016.pdf.

[32] B. D. de Dinechin, P. G. de Massas, G. Lager, C. Lger, B. Orgogozo, J. Reybert, and T. Strudel, "A distributed run-time environment for the kalray mppa-256 integrated manycore processor," *Procedia Computer Science*, vol. 18, pp. 1654 – 1663, 2013.

[33] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[34] P. Dziurzanski, A. K. Sing, L. Indrusiak, and B. Saballus, "Benchmarking, system design and case-studies for multi-core based embedded automotive systems," in *2nd Int. Workshop on Dynamic Resource Allocation and Management in Embedded, High Performance and Cloud Computing (DREAMCloud)*, 2016.

Appendix

A. Notation Summary

Variable	**Definition**
A_i	Application i
$\mathcal{R}_i$	Set of data read operations by A_i
$\mathcal{W}_i$	Set of data write operations by A_i
M_j	Application message $\in \mathcal{R}_i \cup \mathcal{W}_i$
f	Size of one flit in bytes
D_C	NoC link delay in clock cycles
D_{SW}	Switching delay of one NoC router in clock cycles
b_D	Buffer size on the D-NoC in flits
b_C	Buffer size on the C-NoC in flits
h	Header size for each NoC packet in flits
M_{CNOC}	Size of one C-NoC message in flits
S_i	Payload for one NoC packet of cluster i in flits
P_i	Total size of one NoC packet of cluster i in flits
α_{A_i}	Number of NoC packets required for application message A_i
P_{LP}	Size of the last packet in flits
T_W	Window size of the traffic limiter in clock cycles
β	Bandwidth quota of the traffic limiter in flits
β_{min}	Lower bound for the bandwidth quota in flits
β_{max}	Upper bound for the bandwidth quota in flits
C_{BNL}	Basic network latency of one packet in clock cycles
$WCTT_{C-NoC}$	WCTT of a request message on the C-NoC in clock cycles
$WCTT_{IO \rightarrow CC}$	WCTT of a data response from I/O subsystem to the requesting cluster on the D-NoC in clock cycles
C_{NL}	Network latency of the application message in isolation over the D-NoC from I/O subsystem to the compute cluster in clock cycles
B_{rel}	Blocking in the I/O subsystem, before sending the message to the compute cluster in clock cycles
R_2	NoC router that is shared by both compute clusters, when sending to the I/O subsystem
$WCTT_{CC \rightarrow IO}$	WCTT of a message to send data from the compute cluster to the I/O subsystem in clock cycles
θ	Duration between two consecutive full bursts sent by the compute cluster under flow regulation in clock cycles
$dep(t)$	Number of flits that departed from the buffer of the shared router up to time t
$arr(t)$	Number of flits that arrived in the buffer of the shared router up to time t
$arr(t, M_i)$	Number of flits that arrived in the buffer of the shared router up to time t when M_i flits are sent
$b_{max}(M_i)$	Max. buffer occupancy while sending M_i payload flits from compute cluster to I/O subsystem in flits
s_{dep}	Number of packets which left the shared router up to time t
d_{dep}	Number of delay segments imposed on the departure of packets on the shared router up to time t
Δt_{dep}	Number of flits of a currently departing packet on the shared router already sent up to time t
s_{arr}	Number of packets which arrived at the shared router up to time t
d_{arr}	Number of delay segments imposed on the arriving packets on the shared router up to time t
Δt_{arr}	Number of flits of a currently arriving packet on the shared router already arrived up to time t
t	Current algorithm time in clock cycles