Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Engenharia / Faculdade de Informática
Engenharia de Computação

EFFECTS OF NoC ARCHITECTURAL PARAMETERS IN MPSoC PERFORMANCE

Douglas Roberto Guarani da Silva
Bruno Scherer Oliveira

End of Term Work

Prof. Fernando Gehm Moraes, PhD.

Porto Alegre
2013

Douglas Roberto Guarani da Silva
Bruno Scherer Oliveira

EFFECTS OF NoC ARCHITECTURAL PARAMETERS IN MPSoC PERFORMANCE

End of Term work presented as part of the activities to obtain the degree of Computer Engineering at the Faculty of Engineering at the Pontifical Catholic University of Rio Grande do Sul.

Prof. Fernando Gehm Moraes, PhD.
ADVISOR

ABSTRACT

The goal of this end of term work is to evaluate the impact of the Network-on-Chip (NoC) parameters over the performance of applications on Multiprocessors Systems-on-Chip (MPSoCs). Nowadays, MPSoCs have so many constraints of performance that bus-based communications are not able to achieve the full potential of MPSoCs. Therefore, the adoption of networks-on-chip (NoCs) is a trend for the communication infrastructure in MPSoCs due to their performance compared to bus-based architectures and scalability compared to crossbar-based architectures. However, we were not able to find any reference in the state-of-the-art evaluating the impact of NoC parameters in the performance of applications running in MPSoCs. This work proposes a simulation-based monitoring method to evaluate network performance. Such monitoring collects performance results for different MPSoC applications scenarios, and for each scenario network parameters such as buffer size, routing algorithm and topology vary. The goal is to correlate NoC parameters with the MPSoC performance. This work adopts the HERMES NoC and the HeMPS MPSoC and tries to answer the following question: "*how does a given NoC parameter affect the performance of the MPSoC*?"

“Sometimes the questions are complicated and the answers are simple”

Dr. Seuss

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

DMA	Direct Memory Access
FIFO	First In, First Out
HeMPS	HERMES Multiprocessor System
MP	Master Processor
MPSoC	Multiprocessor System-on-Chip
NI	Network Interface
NoC	Network-on-Chip
PE	Processing Element
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
QoS	Quality of Service
SoC	System-on-Chip
SP	Slave Processor
UART	Universal Asynchronous Receiver/Transmitter
VHDL	Very High Speed Integrated Circuits Hardware Description Language

1 INTRODUCTION

"He who has begun has half done.
Dare to be wise; begin!"

Horace, *Epistles*

1.1 Introduction

In 1965, Gordon Moore foresaw that every 18 months the number of transistors in an integrated circuit would double while keeping the same price. Therefore, it became possible to develop complete systems inside one single chip, and so the Systems-on-Chips (SoC) were created. A SoC is an integrated circuit that implements the majority or all the functionality of a whole electronic device [JER05]. When the SoC contains multiple processors, it shall be called Multiprocessor System on Chip (MPSoC).

These MPSoCs may be designed as general purpose, being able to run a set of different applications, with performance constraints. To achieve such goal, they present capabilities for being adaptable to provide quality of service (QoS). Another feature of MPSoCs is that they can are able to load applications during execution time. For this reason, there are MPSoCs with built-in methods to define applications mapping for a better performance and power management [SCH10].

Nowadays, MPSoCs are present in many commercially available systems, with a broad use in areas like signal processing and telecommunications. Such MPSoCs are built with an architecture that, in its nature, contains heterogeneous resources, which may include a diversity of dedicated hardware, multiple general-purpose processors, memories and an interconnection architecture [WOL04].

The interconnection architecture used by these MPSoCs varies. Some Authors have foreseen that bus-based and point-to-point connections would not fulfill the communication requirements of today's integrated circuits. Consequently, starting from concepts originated from computer networks, the concept of intra-chip network (or Network-on-Chip, NoC) was created [BEN02]. This interconnection architecture brings many improvements over bus-based interconnections: efficient power usage and reliability [ZAM11], scalability of bandwidth (compared to buses) [KUM13][STE12] and reusability. The basic architecture of an intra-chip network is a set network of routers connected to certain elements. Communications work by sending data from a processing element through a router through intermediate channels until it reaches its destiny [BEN02].

The state-of-the-art of NoC research presents many works evaluating application specific NoC designs [MOD11]. However, according to our review, none has approached the problem of how general-purpose NoC designs affect applications running in MPSoCs connected using this infrastructure. The buffer size, routing algorithm and NoC topology are examples of parameters that should be explored to have a better understanding of how the NoC affects applications running in MPSoCs.

The goal of this work is to generate these missing metrics, and thus to evaluate how different NoC parameters influence the application level. Therefore, obtaining some understanding of how on-chip networks – designed for general purpose MPSoCs – and their parameters impact applications performance.

1.2 Motivation

While researching the state of the art of NoC-based MPSoCs, we were not able to find any study on the impact caused by the NoC architecture over a set of different applications using an MPSoC as infrastructure for running applications. For this reason, we will explore this path with the goal to generate such metrics, determining how NoC parameters impact applications running on MPSoCs.

1.3 Objectives of this Work

The objectives of this end of term work include:

- Get a better understanding of how NoC-based MPSoCs work, resulting in added understanding and comprehension of NoCs;
- Develop local and link level non-intrusive monitoring methodologies and systems, able to collect packets statistics transmitted through the NoC;
- Create methodologies and systems able to generate metrics using the collected data. Those metrics include:
  - Network latency;
  - Network jitter;
  - Network throughput;
  - Network estimated area and frequency.
- Develop a reference work for students, researchers and engineers that want to have a better understanding of how NoC parameters affect the execution of applications running in MPSoCs;
- Apply many of the concepts learned during the undergraduate course of Computer Engineering.

1.4 Document Structure

After the introduction and contextualization set forth by this first Chapter, the second Chapter provide a description of MPSoCs and NoCs, followed by a presentation of the HeMPS MPSoC. In the third Chapter we discuss the monitoring methodology and systems, together with the analysis methods. This is followed by the fourth Chapter, which presents the evaluated NoC parameters. The fifth Chapter presents the evaluated scenarios. The sixth and most important Chapter presents the end of term work results and finally the seventh Chapter presents our conclusions.

2 NOC AND MPSOC

"All we have to decide is what to do with the time that is given to us."

J.R.R. Tolkien, *The Fellowship of the Ring*

This Chapter presents basic concepts required to the development of this work. Section 2.1 presents concepts about NoCs while section 2.2 covers MPSoC features necessary for the understanding of this work.

2.1 NoC

Networks on-chip emerged as an alternative to bus architectures. The main characteristics of this architecture are the following [OST03][KUM13][STE12][ZAM11]: *(i)* enhanced reliability and efficiency of power management; *(ii)* larger scalability of the bandwidth with regard to bus based architectures; *(iii)* distributed routing decisions.

The NoC concept became a main trend in embedded systems. This is mostly related to the fact that NoCs have a more efficient power usage (compared to bus connections), once that the communication is made in a peer-to-peer mode, using smaller wires due to the smaller distance between those peers. In bus-based connections, the usage of longer wires results in bigger power usage.

Due to the multiplicity of possible paths in a NoC, it is possible to explore the communication parallelism between nodes, meaning that multiple nodes can engage in simultaneous transactions. Bus based architectures can make a single transaction at any given moment, limiting the bandwidth per node.

Scalability relates to the ability to connect additional nodes of hardware, without major performance loss. Actually, in NoCs the addition of modules in the network induce an augmentation of the number of communication channels, therefore potentially improving the performance [OST03]. In bus-based interconnections, this fact is not observed. What actually happens is that the additional nodes increase the bus length, reducing its performance.

2.1.1 Network Topologies

There are two categories of network topologies for multiprocessors, according to how routers and processing nodes are interconnected: direct networks and indirect networks [ZEF03] [MOR04]. In direct networks, as illustrated in Figure 1, each router has a processing node linked with it, and this pair can be seen as a unique element, typically referred as a node or tile [DUA01].

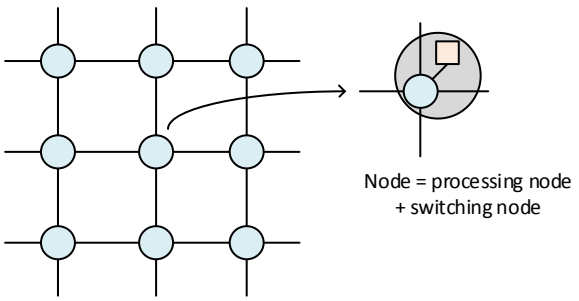


Figure 1 – Network Nodes [WOS07].

The most common direct orthogonal (intersected at right angles) networks are the n-dimensional arrays of nodes, such as: mesh (Figure 2-a), torus (Figure 2-b) and hypercube (Figure 2-c).

In indirect networks not all routers are connected to processing nodes. Only a few routers have connection to processing nodes, and these routers are the only ones that can be the source or target of messages. Examples of indirect topologies include tree, butterfly, banyan networks and clos networks [DUA01].

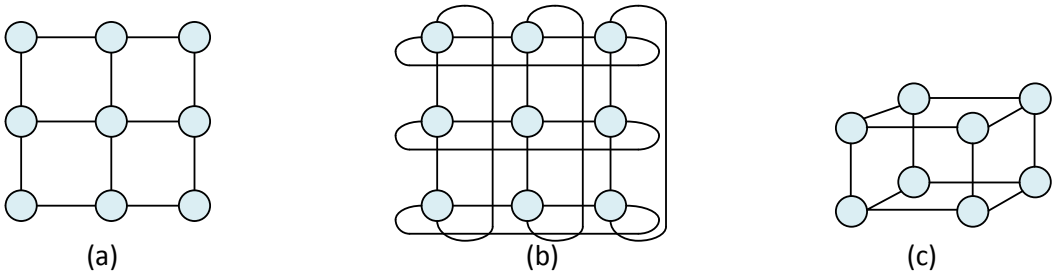


Figure 2 – Examples of direct network topologies (a) Mesh 2D 3x3; (b) Torus 3x3; (c) Hypercube 3D [WOS07].

2.1.2 Communication Mechanism

The communication mechanism specifies how a message evolves through the network. Two methods for transferring messages are circuit switching and packet switching [HWA92]. Circuit switching consists in establishing a connection before sending the data, which means it allocates a sequence of channels to guarantee the QoS. After connection establishment, any other attempt to use the allocated channels is denied, until the connection is released. For packet switching, packets are transmitted without any need for connection establishment procedures. For this reason, each packet is routed individually from its source until it reaches its target.

2.1.3 Routing Algorithm

A routing algorithm defines the path that a packet takes between its source and target routers. Depending where routing decisions are made, it is possible to classify routing in source or distributed routing [DUA01]. In source routing the path taken by packets is defined at the origin. In distributed routing, every router decides the next direction to send the packet. Routing algorithms can be classified as deterministic or adaptive, based on how the path is defined to transmit packets. A deterministic routing employs a path that is uniquely defined by the source and target addresses only [MOR04]. Adaptive routing may define the path also according to instantaneous network traffic [DUA01][MOR04]. Adaptive routing can still be divided into partially or fully adaptive. Partially adaptive routing uses only a subset of the available physical paths between source and target routers, by imposing a set of rules restricting the possible routes. Fully adaptive algorithms do not impose any restriction to these rules.

2.1.4 Switching Mode

The choice of a switching mode is required when using packet switching. This defines how packets traverse routers. Some of the modes are store-and-forward, virtual cut-through and wormhole [NI93]. The store-and-forward mode completely stores the packet in the router, and then forwards it to the next router. In virtual-cut-through mode, the packet is forwarded once the next router guarantees that the packet can be completely accepted. Therefore, the use of a buffer to store a complete packet is a necessity, just like in the store-and-forward mode, although with potentially lower communication latency. The wormhole method is a variation of the virtual cut-through mode, which avoids the need for large buffer space, because packets are sent between routers in units called flits (flow control digits). Typically, only the header flit has the routing information. Hence, all other flits that compose a packet follow the same path reserved by the header [MOR04].

2.1.5 HERMES NoC

We use in this work the HERMES NoC, developed in the GAPH research group [MOR04]. This NoC uses as communication mechanism packet switching, wormhole mechanism, and adopts a mesh topology, due to the easiness to develop the router algorithm, generate the layout and add new nodes [MEL05]. Figure 3 illustrates a 3x3 instance of the Hermes NoC.

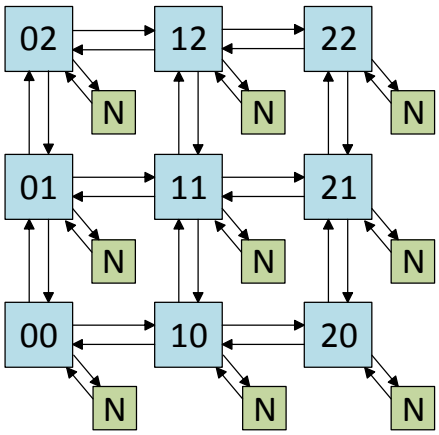


Figure 3 – An example of system based on the NoC HERMES. N represents a processing element and the routers are identified by their XY coordinates in the network [WOS07].

As presented in Figure 4, the following signals compose the interface between routers:

- Tx: indicates data availability;
- Data_out: data to be sent;
- Ack_tx: control signal indicating successful data reception;
- Rx: control signal indicating data availability
- Data_in: data to be received
- Ack_rx: control signal indicating successful data reception

The number of control flow, flit width, buffer depth and routing algorithm are parameterizable.

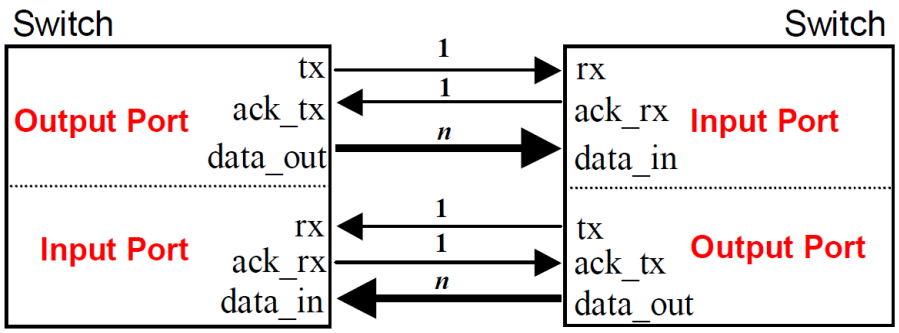


Figure 4 – HERMES interface between routers [MOR04]

Each router uses a centralized control and has up to 5 bidirectional ports: North, South, East, West and Local. Besides the Local port, each port connects the router to a neighbor router. The Local port connects the router to a processing element. Each link has two unidirectional channels (physical channels). Each physical channel can be multiplexed in m-virtual channels. Figure 5 shows a router with two virtual channels per physical channel.

Each input port has its own buffer for decreasing performance loss when flits are blocked. This performance loss happens because when a flit is blocked, subsequent flits can be blocked in the routers along the path from the source to the router with the blocked port. By using a buffer, the number of routers affected

by the flit blocking can be reduced. The buffer implemented in the HERMES Router works as a circular First In First Out (FIFO).

The input port of each router is responsible for receiving flits, and to store them in the buffer of the correct virtual channel. The lane_rx has **n** bits, where 2^n is the number of virtual channels. After buffer selection, the flit is stored and the number of credits (not to be taken as the *credit_in* signal) of the virtual channel (free spaces to storage) is decreased. When the output port transmits a flit, it is removed from the buffer and the number of credits is increased. The available credit information is transmitted to the neighbor router by the credit_o signal. It has 1 bit for each virtual channel, which is interpreted similarly to the lane_rx.

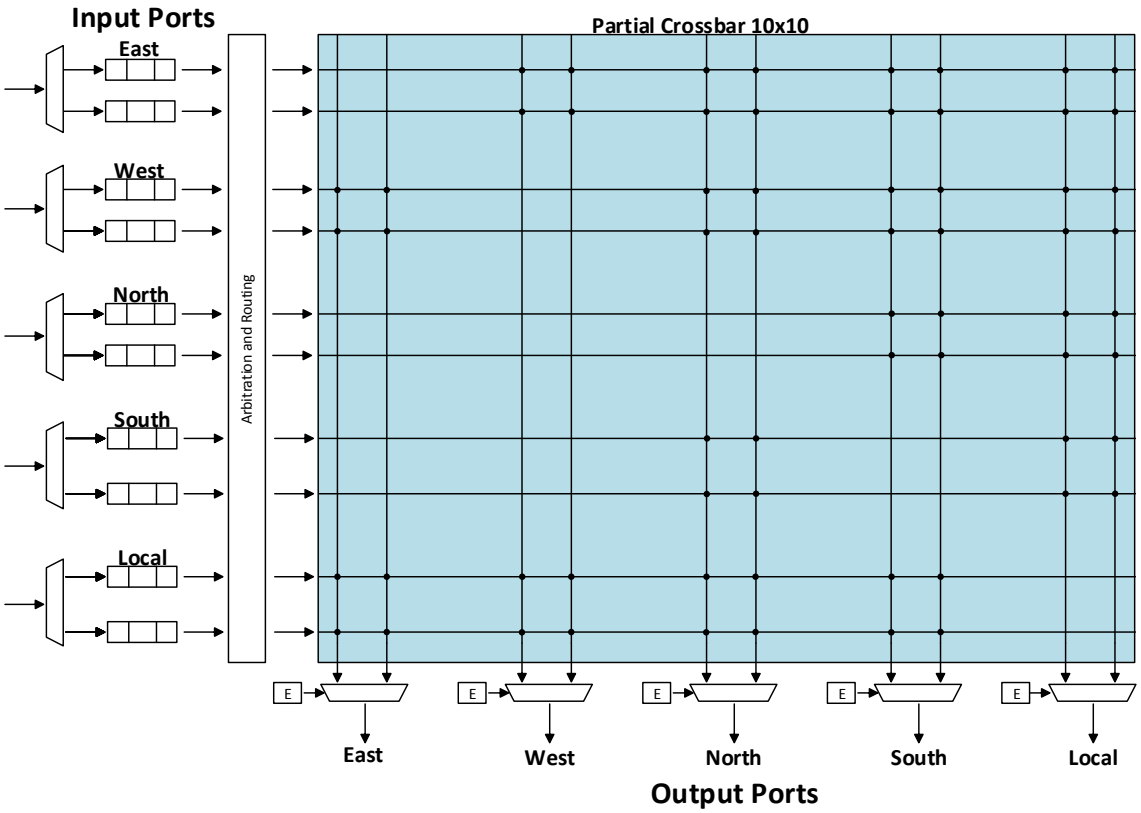


Figure 5 – HERMES router structure for two virtual channels. The E Module represents the circuit that schedules an input port to output port connection. This Crossbar is already optimized for the XY routing algorithm. Dots indicate a possible connection between an input and output port pair [WOS07].

The HERMES NoC uses a centralized control logic that implements the arbitration logic and the routing algorithm. By receiving a header flit, the arbitration is executed and if the requisition is granted, the routing algorithm is used to connect the flit of the input port to the chosen output port. Every router must have a unique network address. To simplify network routing, the addresses are expressed in XY coordinates, whereas X represents horizontal position and Y the vertical one.

When all flits are transmitted, the connection is closed. The connection can be closed in two different ways: by a trailer or by using a flit counter. When using a trailer two or more flits are used as packet terminator and an additional logic is used to detect the trailer. Therefore, to simplify the design, the HERMES router has a counter for each input channel. The counter of each channel is initialized when the second flit of a packet is received, which represents the number of flits that constitute the payload. The counter is decreased for each successfully transmitted packet. Therefore, when the counter value reaches zero, the position of the free vector of the output virtual channel goes to 1 (free=1), closing the connection.

2.2 HeMPS MPSoC (HERMES Multiprocessor System)

We use the HeMPS MPSoC [CAR09] as the reference platform for this work. The basic characteristics of the HeMPS MPSoC are: homogeneous processing; use of NoC; communication by message exchanging; and paged memory organization.

All processing elements have the same architecture, thus simplifying object code generation and task mapping and migration. The use of a NoC is justified to enhance scalability [PAS08].

Each processing element has its own private memory, responsible for the storage of data and instructions. This reduces data traffic during communication. Communication in HERMES is based on message exchange, which is more natural for distributed memory systems. By using a paged memory organization there is a simplification of the task mapping process as well as in task migration processes. Consequently, there is no need to use complex memory management mechanisms [CAS13].

The HeMPS MPSoC contains three main components: *(i)* Plasma-IPs, the HeMPS processing elements; *(ii)* an instance of the HERMES NoC used as communication infrastructure; *(iii)* an external memory called task repository. Figure 6 presents an instance of the HeMPS architecture with a 2x3 HERMES NoC connecting six Plasma-IPs.

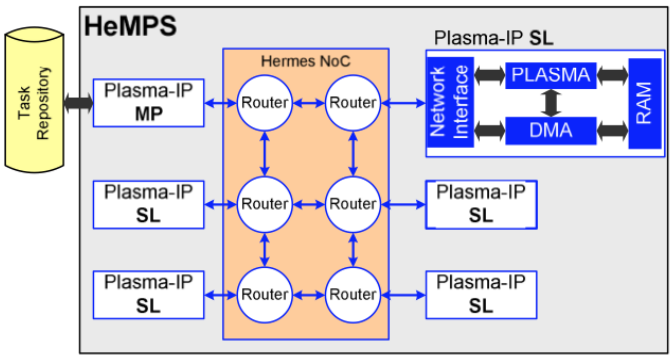


Figure 6 – An instance of the HeMPS MPSoC with six processing elements [CAR09].

The Plasma-IP is the processing element used by HeMPS. It consists in two types of instances: *master*, called Plasma-IP MP, responsible for the resource management of the system; and *slave*, named Plasma-IP SL. In the HeMPS MPSoC there is a single Plasma-IP MP which implements a centralized resource management mechanism. Plasma-IP is composed by the following components:

(i) A plasma processor [PLA11], a 32-bit RISC processor with an encompassing subset of the instructions from the MIPS2000 architecture. The Plasma used in the HeMPS is a modification of the version 2.0 of the Plasma processor, which includes new memory mapped registers, and the exclusion of the UART module;

(ii) A private memory, which contains the operating system, called microkernel, executed by the Plasma processor. In the case of the Plasma-IP SL, the memory is divided in pages with fixed size, and each page can house one task;

(iii) A direct Memory Access (DMA) module that enables the processor to execute tasks without controlling the message exchange with the network. Primary functions are to transfer the object code from arriving tasks from the network interface to the processor memory and send the master processor debugging messages;

(iv) A network interface, which connects the HERMES NoC to the Plasma processor and to the DMA, responsible to implement the communication protocol with the NoC.

The task repository is a external memory to the MPSoC that contains the object code for all tasks that possibly run on the system. The first positions of the repository contain descriptors for every task, with information such as a unique identifier, the initial position and the task size. In HeMPS, all applications are

modeled through task graphs, where vertices represent tasks, and edges stand for communications between tasks. Tasks without dependencies are called *initial tasks*. When the MPSoC is initialized with a pre-determined application, the mapping heuristics loads only the initial tasks to the MPSoC. The other tasks are dynamically mapped to processing elements at run time, depending on requested communication and available resources.

3 NOC MONITORING AND EVALUATION

"It's a dangerous business going out your front door."

J.R.R. Tolkien, *The Fellowship of the Ring*

In this Chapter we present monitoring concepts, along with the specification of the developed monitors and their functionalities.

3.1 Monitoring Concepts

Monitors can be either software or hardware components capable of analyzing the system resources in dynamic workloads [KON13].

There are multiple types of monitors, categorized according to their respective functionality, such as:

- **Monitors for performance:** Observe the behavior of a system, collecting, for example, throughput and latency-related statistics.
- **Monitors for Quality-of-Service:** Observe if latency, throughput and utilization follow pre-determined constraints.
- **Monitors for power, energy and temperature:** Have the role of finding thermal hotspots and large temperature gradients in the circuit, as well as power consumption.

Monitors can also be classified according to:

- **Implementation technique:** Monitoring can be mechanisms based on software, hardware or hybrid.
- **Sampling frequency:** According to this criteria monitors may be classified as continuous, periodic, on-demand or adaptive.
- **Intrusiveness:** Monitors may be Intrusive or non-intrusive, whether the monitor behavior can perturb the system execution or data flow.

All these definitions are based on the survey of Kornaros et. al. [KON13].

3.2 Non-intrusive Monitors

This work develops an implementation of monitors capable of extracting a significant amount of statistics from a NoC that are later processed by a set of tools. With these statistics, we can measure network utilization and performance metrics of an intra-chip network in multiple application scenarios. Two types of monitors are implemented: one type designed to obtain data link utilization and the other type to obtain the latency and throughput between sender/receiver pairs.

The monitors are components described in simulation-only VHDL. Multiple monitors are instantiated inside each router, with the objective of analyzing packets transmitted through the network. Just by adding these monitors, we are able to extract all data required to obtain metrics that this work employs. The applications running in the MPSoC do not need any modification, as well as the architecture of any other module of the MPSoC.

The HeMPS MPSoC has an RTL description, which is accurate at the clock cycle level. Therefore, monitors are expected to run also at the clock cycle level, taking advantages of VHDL features that are not synthesizable, such as support to file operations. While evaluating the MPSoC platform, monitors generate text files containing statistics about the network. These text files are used as input for additional tools that are able to extract network metrics.

Monitors have the objective of helping to analyze how modifying certain parameters of the intra-chip network affect the application performance in the MPSoC. They neither modify the packet s nor include any additional latency in the network; therefore, they are not intrusive with respect to platform execution. It is also important to highlight that such monitors evaluate the entire execution time of the applications (the time suring which a full set of application runs, or when the simulation stops) in the MPSoC.

3.2.1 Local Monitor

This monitor is placed at the local port of each router, as illustrated in Figure 7. Its objective is to extract statistics related to the communication between a given source and one or more targets. This monitor creates a text dump of all packets that traverse the ports where these monitors were instantiated.

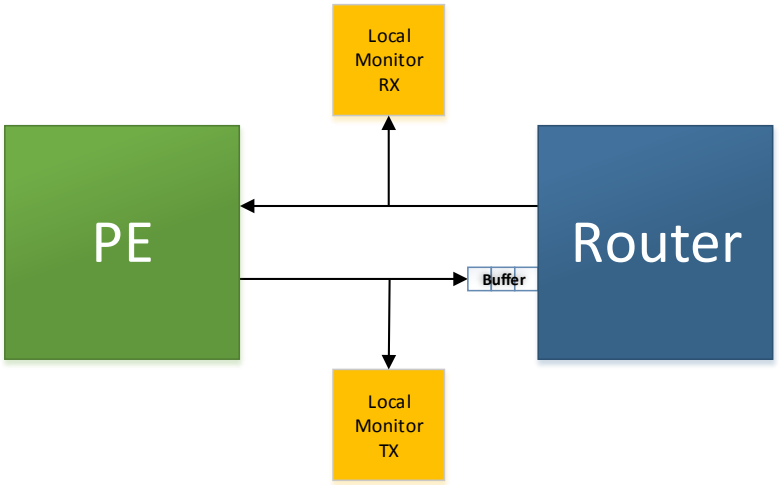


Figure 7 – Location of the local monitor.

Inside each router, two local monitors are instantiated between the network interface and the local buffer, as shown in Figure 7. In this location, monitors are able to ascertain the exact moment that each packet joins or leaves the network.

Since there are such monitors at every router, one for each local port link, each monitor generates two types of text dumps:

- All packets that a specific PE receives, obtained through the monit or located in the RX local port;
- All packets that a specific PE sends, obtained through the monitor located in the TX local port.

For each packet that a PE sends or receives, this monitor records the following information in a text file for each port:

- The first clock cycle that the monitor detected the transmission of a packet in a specific port;
- The clock cycle when the last flit of this packet passed through the port;
- The packet data.

A tool, developed in Python, capable of simultaneously interpreting the outputs of every local monitor network matches equivalent packets in both receiving and sending pairs. Thus, it is possible to extract the point-to-point network latency for each specific packet. The latency is computed using the following equation:

$$Latency = RxClkEnd - TxClkBegin$$

Where: *RxClkEnd* is the clock cycle when the last flit of a packet arrived in the PE, *TxClkBegin* is the clock cycle the sender PE injected the first flit of the packet in the network. Thus, the unit of the *Latency* is clock cycles.

After obtaining the latency for each packet, following metrics are computed for each sending-receiving pair of PEs, and for the entire network:

- **Average packet latency:** Many parallel applications require a low average communication latency between processing elements to fully take advantage of multiprocessing. This value is calculated using the equation, for each *source-destination* pair:

$$AvgLatency = \frac{SumLatencies}{QtyPackets}$$

  Where: *AvgLatency* is the average latency, *SumLatencies* is the sum of all the packet latencies and *QtyPackets* is the amount of transferred packets.
- **Jitter**: Jitter is the deviation from the average latency. It affects the ability of the applications to adapt to this difference, thus, this value affects directly the Quality-of-Service. The jitter is the standard deviation of the latencies and can be calculate by following this procedure
  - For each packet exchanged between a *source-destination* pair, compute the latency variance:

$$Variance = (PacketLatency - AvgLatency)^2$$

  - The square root of the mean of the square of differences is the jitter.

$$Jitter = \sqrt{\frac{SumVariance}{QtyPackets}}$$

- **The minimum packet latency:** Minimum packet latency between each *source-destination* pair.
- **The maximum packet latency:** Maximum packet latency between each *source-destination* pair.
- **The sum of the latencies:** Sum of all extracted latencies between each *source-destination* pair.
- **The number of packets and flits:** Transmitted between each source-destination pair

3.2.2 Data Link Monitor

This monitor type has the objective of extracting network usage. This monitor is instantiated in the sending link of every router, as shown in Figure 8. These monitors count the number of packets and flits that each one of these ports transmits in a specific amount time, named *time window*.

Similar to the local monitor, the data link monitor records in a text file the collected data. However, instead of storing every packet, the data link monitor stores how many packets and flits each port transmitted, separating this information by time windows. Using these text files generated by the data link monitor, an intermediate tool generates a database containing an optimized copy of the extracted statistics. The user can then employ a graphical tool, also developed in Python, to open this database and analyze the network usage.

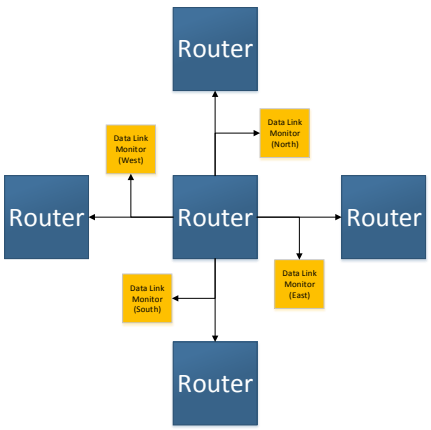


Figure 8 – Location of the Data Link Monitor.

Figure 9 and Figure 10 show the interface of these graphical tools displaying the statistics acquired from a 3x3 network. Figure 9 shows the network utilization according to the parameters selected in the window shown in Figure 10.

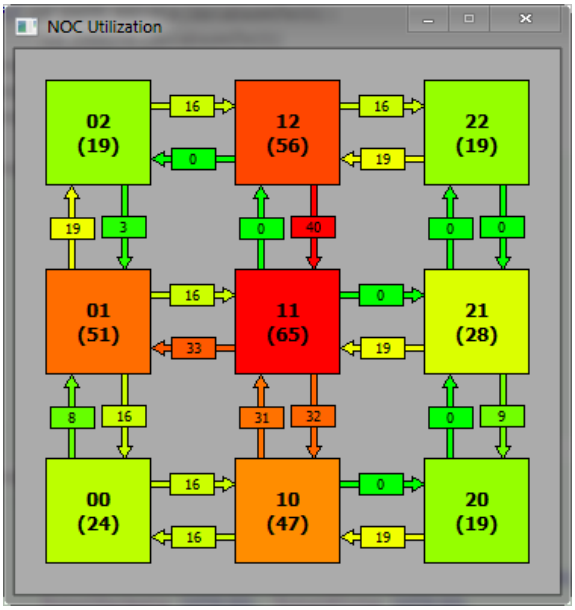


Figure 9 – Visualization of the network utilization.

Each box in Figure 9 represents a router. Every router has two numbers inside its representation. The upper number is the router address, and the lower number between parentheses is the quantity of transmitted units (packets or flits) sent by every port.

Arrows represent a channel between two routers. The arrow direction identifies the specific channel direction and the number inside the arrow represents how many units this specific link transmitted.

Colors represent utilization relative to the maximum computed usage. Green represents an element that is not used. Red denotes the most used element. Shades between green and red represent linearly growing utilization.

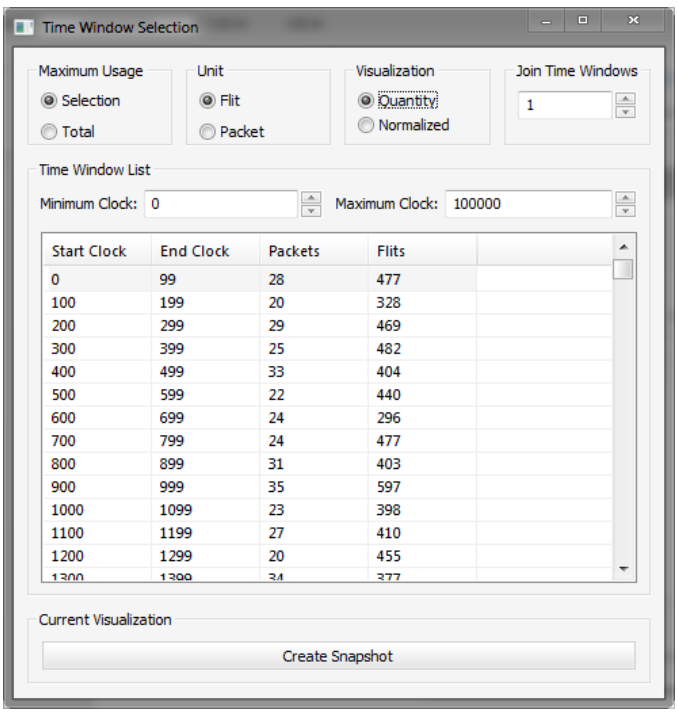


Figure 10 – Parameters used to generate the visualization.

Figure 10 shows the following parameters:

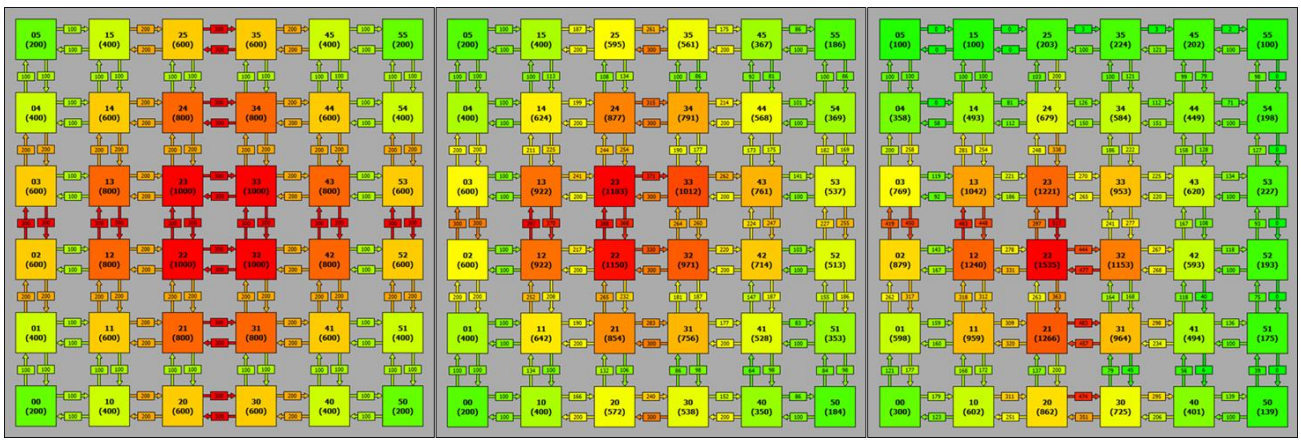
- **Maximum Usage:** Defines the convention used in the element coloring and normalized values.
  - **Selection:** The utilization is relative to the most utilized router and data channel in the selected time window.
  - **Total:** The utilization is relative to the total utilization of each specific element, considering all time windows combined. For example, if 10 packets were transmitted in a specific link during all the evaluation time, and in the selected time window five packets were transmitted in this link, this link utilization will be considered as 50% in the selection.
- **Unit:** Defines the data unit used in the visualization.
  - **Flit:** The values represent the number of transmitted flits.
  - **Packet:** The values represent the number of transmitted packets.
- **Visualization:** Defines the method used to display the values.
  - **Quantity:** The number used is the exact quantity of units transmitted.
  - **Normalized:** Normalizes the value, as the ratio between each element and the maximum usage.
- **Join Time Windows:** Allows the user to join the specified number of time windows. They are shown as a single item in the list and act as if the user had selected these items simultaneously.

The user can use the **Time Window List** to select the time window displayed in the visualization. This list can be filtered by changing the **Minimum Clock** and **Maximum Clock**. It is also possible to select multiple time windows by selecting more than one item. In this case, the value used in each element will be the sum of this element value in every selected time window.

Finally, the user can take a snapshot of the current visualization by clicking on the button **Create Snapshot.** When the user clicks this button, a new window is open with a copy of the current visualization. When the time window is changed, the visualization is updated accordingly, but this snapshot remains unchanged.

This graphical tool provides an easy way of finding network portions that are over-utilized, which can lead to thermal hotspots, faster aging and latency increase in the network. This tool can also analyze the routing algorithm.

Figure 11 is an example of the data extracted from a 6x6 network, using a complement traffic scenario with 100 packets and different routing algorithms. *Figure 11(a)* uses the XY routing algorithm, stressing the center of the network. *Figure 11(b)* uses the West-First routing algorithm, pushing the traffic to the left. Finally, *Figure 11(c)* uses the Negative-First routing algorithm, pushing the traffic to the bottom-left.



(a) XY (b) West-First (c) Negative-First

Figure 11 – Effect of traffic loads with different routing algorithm in a NoC.

4 NOC ARCHITECTURAL PARAMETER EVALUATION

"It is when I struggle to be brief that
I become obscure."

Horace, *Epistles*

In this Chapter, the evaluated NoC parameters are presented, as well as the adopted test scenario. These parameters include, buffer depth, routing algorithm and topology. To conduct a fair evaluation, a real application (MPEG decoder) was mapped in the HeMPS MPSoC, with a set of communication flows disturbing the MPEG traffic.

4.1 NoC Parameters

4.1.1 Buffer Depth

Buffers are an integral part of the network router. In most NoC architectures, the buffers are responsible for the largest portion of router silicon area and dissipated power. Therefore, it would be an advantage if we could reduce this, thus decreasing the overall NoC area overhead, without significant performance reductions. In this work, we evaluate four different buffer depths: 4, 8, 16 and 32 flits. The original HeMPS uses a 16-flit buffer depth.

4.1.2 Routing Algorithms

As the routing algorithm plays a major role in the NoC architectures, the present work evaluates the performance of 5 routing algorithms: XY, West-First, Negative-First, Odd-Even and the Torus Routing Algorithm for Network on Chip (TRANC).

4.1.2.1 XY Algorithm

The XY routing algorithm is deterministic, meaning that packets follow always the same path, even when there are congestions. The XY algorithm compares the target packet address to the current router address. When they are the same, the algorithm redirects the packet to the local port. If not, the packet is first sent in the horizontal direction until it reaches the target router column. Then, the packet is sent in the vertical direction, until it reaches its destination. Figure 12 shows an example of different paths used by the XY routing algorithm.

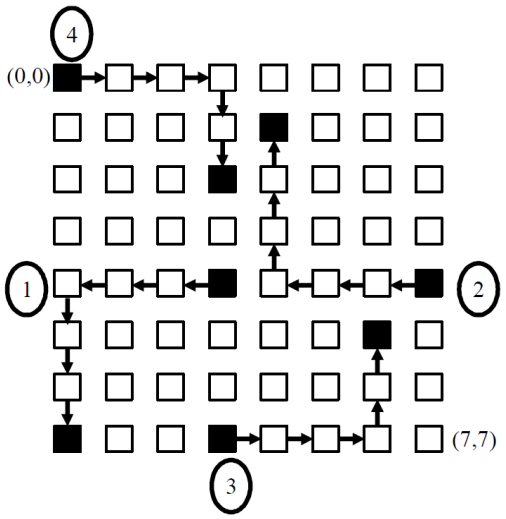


Figure 12 – Example of valid paths in the XY routing algorithm [MEL04].

4.1.2.2 West-First Algorithm

The West-First algorithm is partially adaptive. When the target router address is in the same column of the packet or the West of the current router, packets are routed deterministically, similarly to the XY algorithm (paths 1 and 2 of Figure 13). However, when the target router is located to the East of the source router, routing proceeds adaptively. Therefore, when the packet is blocked for some reason, the packet can be sent through East, North or South directions, avoiding the blocking condition (bars in paths 3 and 4 of Figure 13). The algorithm forbids curves to West when routing adaptively to avoid deadlock conditions. Blocking situations are represented in this Figure as red bars between routers.

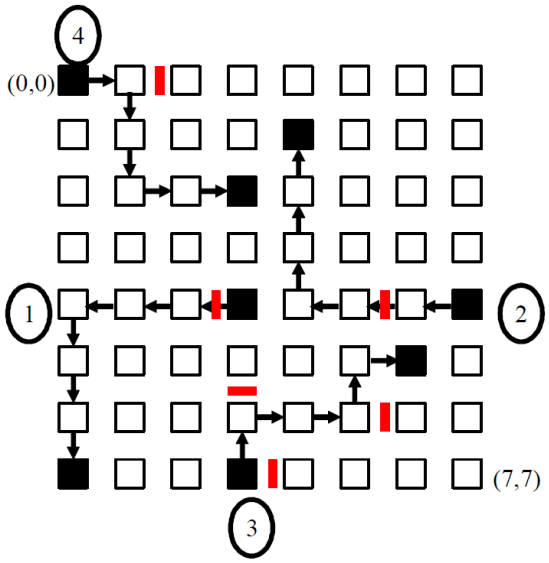


Figure 13 – Example of valid paths in the West-First routing algorithm [MEL04].

4.1.2.3 Negative-First

Being similar to the West-First algorithm, this algorithm is also partially adaptive. It considers North and West directions as "negative" directions and South and East as the "positive" ones. The algorithm forbids that packets take curves from positive directions to a negative one. However, the packets can take any other turns to take advantage of the adaptive routing. Figure 14 shows an example of the Negative-First routing algorithm.

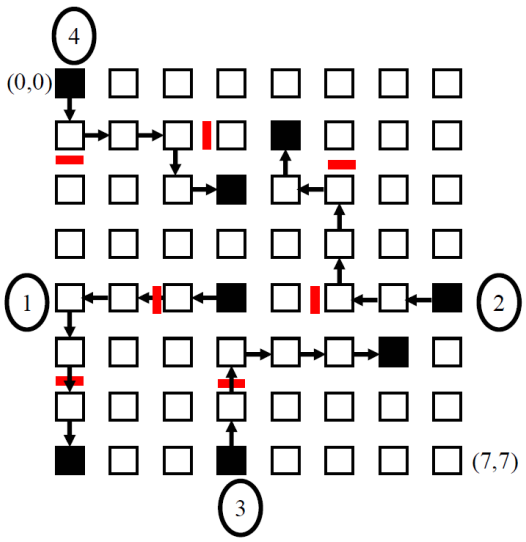


Figure 14 – Example of valid paths in the Negative-First routing algorithm [MEL04].

4.1.2.4 Odd-Even Algorithm

The Odd-Even is an adaptive routing algorithm [DAL93]. Differently from the partially adaptive algorithms discussed above, Odd-Even do not impose a limitation to packet directions. It allows packets to route adaptively in every direction. To avoid deadlocks, two specific rules limit the turns in specific columns. It interchangeably calls a column as *odd* or *even*, according to the coordinates of the specific column. The turning model follows the following rules:

i) A packet cannot execute a turn from East to North when in an *even* column and it cannot perform a turn form North to West when in an *odd* column.
ii) A packet cannot execute a turn from East to South when in an *even* column and it cannot perform a turn from South to West in an *odd* column.

Applying these two rules, packets can take any other turns to avoid congestions.

4.1.2.5 Torus Routing Algorithm for Network on Chip (TRANC)

The Torus Routing Algorithm for Network on Chip (TRANC) is an algorithm designed for Torus networks [DAR12]. It is a deterministic routing algorithm, similar to the XY algorithm for mesh topology. When we used it to evaluate its performance, it is compared with other available routing algorithm for Torus, which is the adapted West-First algorithm.

4.1.3 Arbitration

In a network on-chip, packets arrive in a given input port, and are redirect to an output port, according to the packet target address. In certain cases, multiple packets arriving in different input ports may require the same output port. Therefore, it is necessary to schedule the output port aiming to provide a fair usage of the output channel by the input channels. This scheduling must be done ensuring that no packet waits indefinitely to be directed to its target. The act of scheduling the usage of the output port is the responsibility of router arbiters. This work evaluates two implementations of the arbitration: centralized and distributed round robin.

4.1.3.1 Centralized

The centralized arbitration consists in a single module managing the usage of the output ports. In the *HERMES NoC*, the Switch Control module does this management. The arbiter knows the state of every output port, allowing the implementation of adaptive routing algorithms. However, this implementation may have a small performance, because the arbiter cannot schedule more than one packet simultaneously.

4.1.3.2 Distributed

In the distributed arbitration, each router port controls independently the routing and the arbitration. Each input port has a routing module associated to it, and each output port has an arbiter. The routing module requests a specific output port and the arbiter selects one specific input port if it has received multiple requisitions. This implementation is faster, because multiples packets can be allocated to their output port simultaneously, however, it occupies more area, since it necessary to replicate the routing control and the arbiter for each port. Adaptive routing algorithms may be also used with distributed control. The mechanism to allow adaptive routing in a distributed architecture is based in the status (free/used) of each output port. The routing module of each input port reads the status of the output ports to select the one to transmit the packet.

4.1.4 Topology

The present work evaluates the 2D-mesh and the 2D-Torus topologies. Both are known for their advantages, as scalability for the mesh and the smaller diameter for the Torus. When the topologies are

compared, both use the same routing algorithm, with the purpose to quantify the impact of the topology in the performance.

4.1.4.1 Mesh Architecture

The mesh architecture is based on an *m x n* mesh. Except the routers at the edges, all other routers have four neighboring routers, and a processing element connected to the local port. Its major advantage is scalability, being possible to connect a large amount of PEs in a regular-shape structure.

4.1.4.2 Torus

The Torus topology is also based on an *m x n* mesh, but it is characterized by connecting all edges to their equivalent on the opposite side. The Folded Torus topology is a topology where the physical placement of the routers are made in such a way that the wires always have the same length, thus, minimizing the problem of the links connecting edge routers.

4.2 Experimental Setup

To evaluate the NoC parameters, we chose a scenario corresponding to a real use of the MPSoC. The chosen scenario uses an *mpeg* decoder as main application, with a set of *producer/consumer* applications generating disturbing traffic. The *mpeg* application contains 5 tasks: *start, ivlc, idct, iquant, print*. The *start* task sends frames to the *ivlc* task. Tasks *ivlc, idct, iquant* are responsible to execute the decoding algorithm. The last task, *print*, receives the decoded frames. A 4x4 HeMPS instance is used, with the *PLASMA* processor as PE. The entire environment was generated using the HeMPS Generator [CAR09] tool.

Figure 15 presents the task mapping. This task mapping benefits the *Torus* topology and adaptive routing, since a deterministic algorithm would be affected by the traffic generated by the disturbing traffic. Thus, this mapping scenario can highlight the advantages of these techniques.

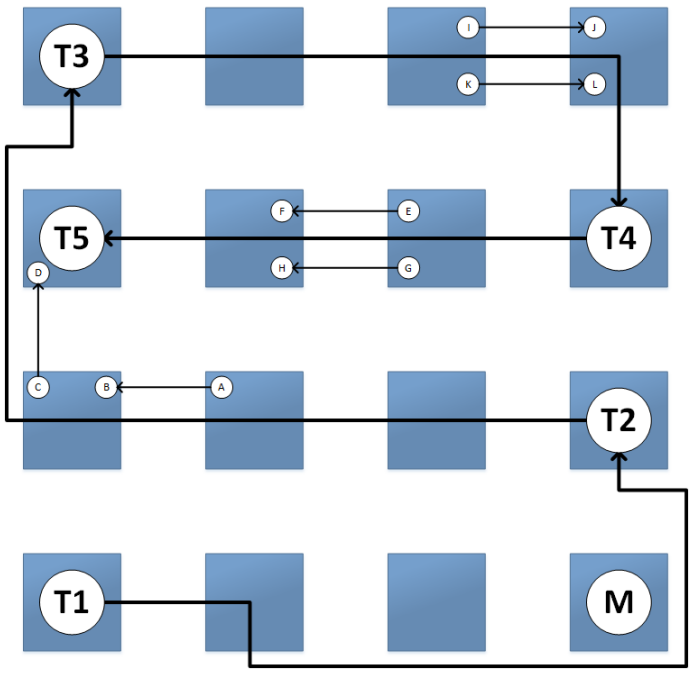


Figure 15 – Evaluated Scenery with all disturbing and the main application (T1: start.c; T2: ivlc.c; T3: idct.c; T4: iquant.c, T5: print.c). M is the manager processor.

The manager PE (*PLASMA-MP*) is mapped in the router 30. The *PLASMA-MP* sends the task object code to the PEs, according to the chosen mapping.

The *producer/consumer* (PC) application contains 2 tasks, generating traffic at the maximum processor rate (on average 15% of the available link bandwidth). The producer generates dummy packets, with a payload containing 200 flits, addressed to the consumer task. The goal of this application is to disturb the application under evaluation.

Based on the task mapping shown in the Figure 15, multiple sub-scenarios were created, varying the amount of disturbing tasks. The simulated sub-scenarios comprise (each disturbing application is represented by two letters, e.g., *AB*):

(i) No Disturbing
(ii) Disturbing AB
(iii) Disturbing AB CD
(iv) Disturbing AB CD EF
(v) Disturbing AB CD EF GH
(vi) Disturbing AB CD EF GH IJ
(vii) Disturbing AB CD EF GH IJ KL
(viii) Disturbing CD
(ix) Disturbing FE
(x) Disturbing FE GH
(xi) Disturbing IJ
(xii) Disturbing IJ KL

A second scenario was evaluated, with a synthetic application replacing the *mpeg* one. It is also based on 5 different tasks communicating sequentially with each other. However, the first task generates packets at the maximum processor rate (on average 13% of the available bandwidth). These packets have a payload size of 200 flits. Then, the subsequent tasks will just redirect this packet to the next task, without executing any computation on the packet. The goal of this scenario is to obtain the highest injection rate that PEs can generate.

Each sub-scenarios is simulated twice:

(i) **Fixed simulation time:** the number of frames or packets of the application under evaluation is very high, and the simulation is executed during 100 ms. The goal is to measure the impact of the evaluated parameters in the latency, jitter and throughput.
(ii) **Application bound**: the simulation runs until the end of the application under evaluation (the MPEG application decodes 128 frames and the synthetic one transmit 1500 packets). The goal is to measure the impact of the evaluated parameters in the execution time.

The obtained results are derived from the entire simulation time. This includes the beginning of the simulation where the tasks object code are transmitted to the slave processors. The warm-up time is not considered, due to the long simulation time.

5 RESULTS

"You see, but you do not observe."

Sir Arthur Conan Doyle, *(Scherlock Holmes) A Scandal in Bohemia, 1982*

In this Chapter we present the results related to the parameters being evaluated, with the metrics generated by the monitors. The experimental setup was presented in the previous Chapter.

5.1 MPEG Application Results

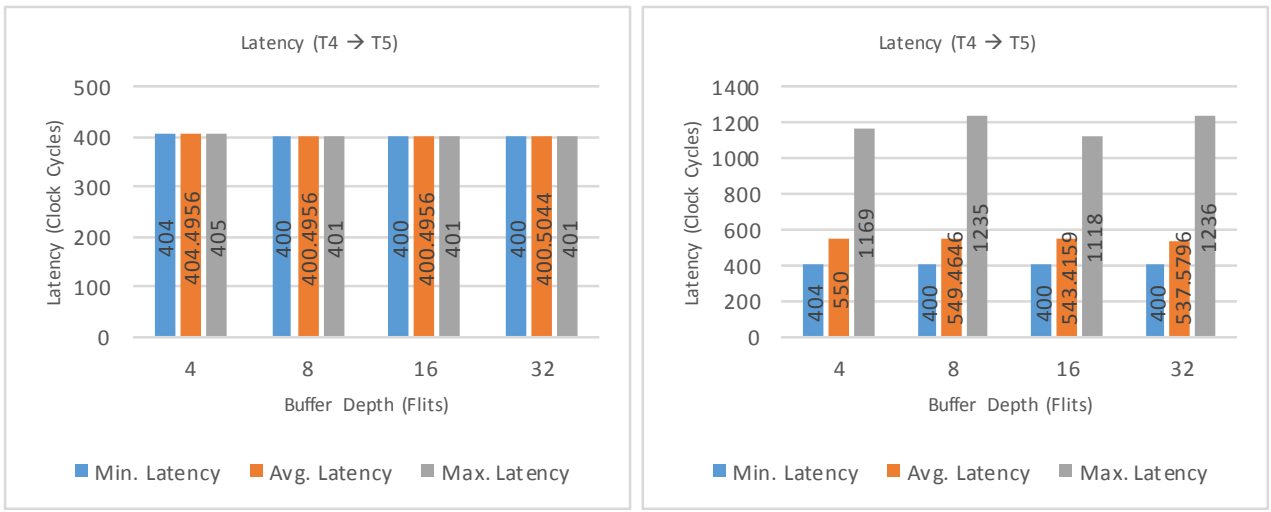
The results presented in this section correspond to the MPEG application executing with a fixed simulation time (100 ms). The extensive set of data and charts can be accessed on our Supplementary Material (http://www.inf.pucrs.br/moraes/my_pubs/tcc/tcc_bruno_douglas_sup_material.pdf) with all the Tables and Charts used to build the analysis of this End of Term Work. It was not included in the text due to its size.

5.1.1 Buffer Depth

This section evaluates the effects of different buffer depths in the network latency, jitter and execution time. Four different buffers depths were evaluated: 4, 8, 16 and 32 flits. Results are extracted from a network with a *mesh* topology, *XY* routing algorithm, and centralized arbitration.

5.1.1.1 Latency

Results show that the buffer depth did not influence the latency in an overall perspective. For example, when no disturbing traffic is added, the average latency is practically equivalent for different buffers configurations - Figure 16(a).



(a) No disturbing traffic

(b) Maximum disturbing traffic

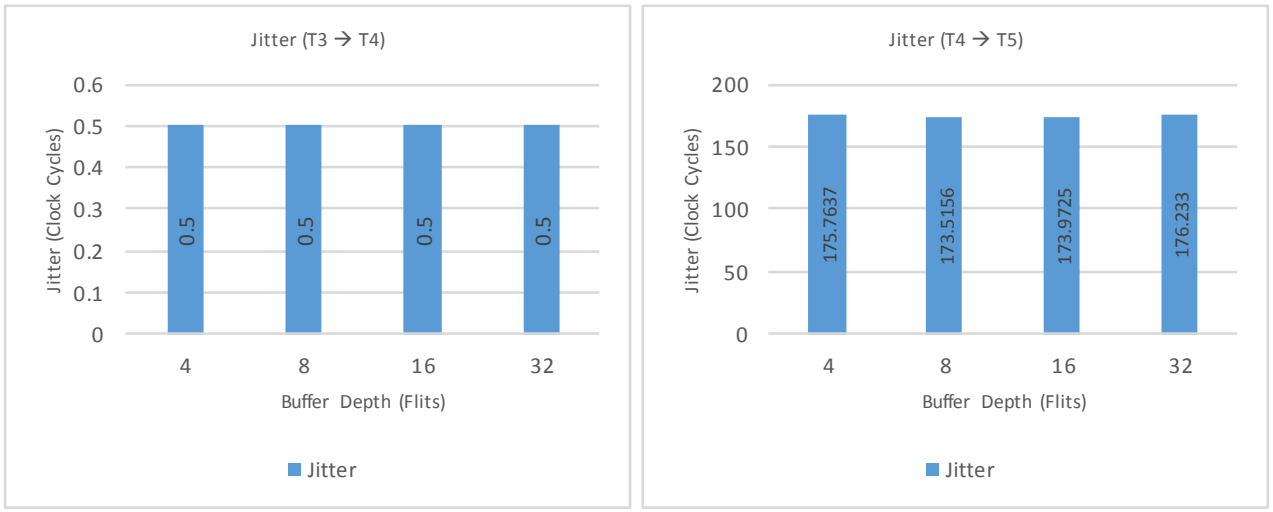
Figure 16 – Average Latency of the flow T4 → T5 for different buffer depths (similar results for the other flows).

Even when adding all disturbing traffic, the average latency is almost the same for different buffers configuration, as shown in Figure 16(b). In this specific example, a 2.3% latency reduction between the buffer 4 and the buffer 32 configurations was observed. However, this difference is not a rule when increasing the buffer size, showing even some increases in the latency as the buffer size increases in some

other paths. It is important to mention that the HeMPS MPSoC stalls in some situations using a buffer depth equal to 4. It was out of the scope of this work to investigate this issue.

5.1.1.2 Jitter

Similarly to the latency evaluation, the buffer depth did not affect the jitter. With no disturbing traffic, Figure 17(a), as expected the jitter is almost 0, corresponding to a constant latency. Increasing the disturbing traffic, the jitter increases with all buffer depths, as shown in Figure 17(b).



(a) No disturbing traffic

(b) Maximum disturbing traffic

Figure 17 – Jitter of the flow T4 → T5 for different buffer depths.

5.1.1.3 Execution Time

The impact on the overall execution time when changing the buffer depth was minimal. Table 1 presents the execution time in milliseconds varying the buffer depth.

Table 1 – Execution time (ms) for different buffer depths.

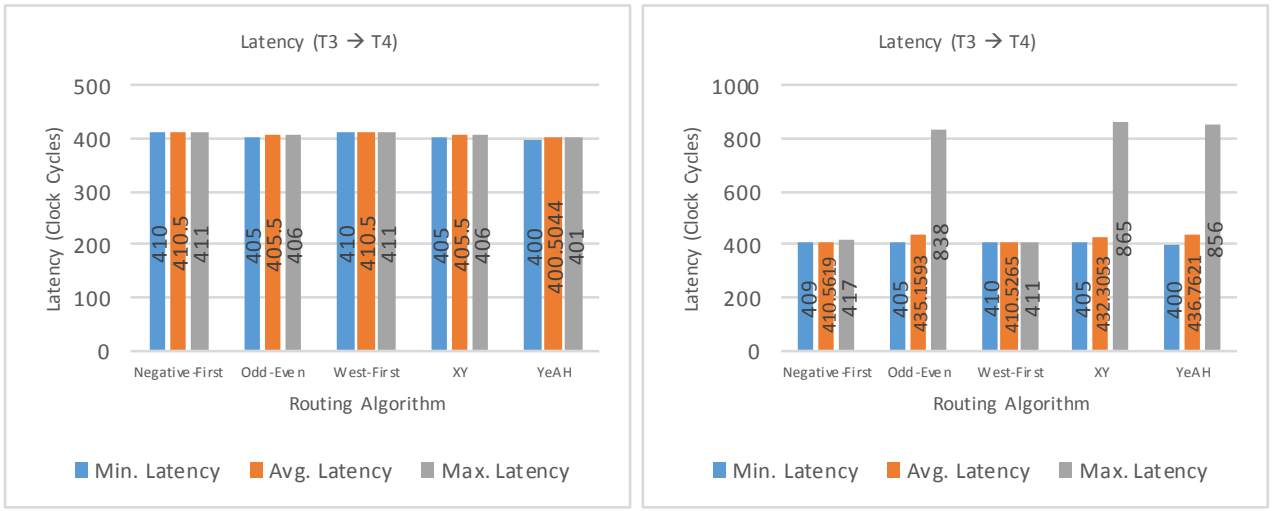
	Execution time (ms) without disturbing traffic				Execution time (ms) with disturbing traffic			
	4	8	16	32	4	8	16	32
Start	55.5	55.5	55.5	55.5	56.3	56.3	56.3	56.2
Ivlc	56.0	56.0	56.0	55.9	56.7	56.7	56.7	56.7
Idct	56.1	56.1	56.1	56.0	56.8	56.8	56.8	56.7
Iquant	56.1	56.1	56.1	56.1	56.9	56.8	56.9	56.8
Print	56.8	56.8	56.8	56.8	57.6	57.5	57.6	57.5

The buffer depth is the first NoC parameter evaluated. The NoC buffer is used to compensate the amount of time required by arbitration and routing. In an MPSoC environment, injection rates are small (bellow to 10%), not requiring in this way a buffer to amortize the performance penalty induced by the router neither to store packets due to congestion. Therefore, this work proposes the use of small buffers, 4 or 8 flits depth, to reduce the NoC area and power.

5.1.2 Mesh Routing Algorithms

5.1.2.1 Latency

Figure 18 present latency values for 4 different routing algorithms. The YeAH NoC uses distributed XY, with distributed arbitration. With no disturbing traffic (Figure 18 (a)), latency values are similar, presenting a small variation of 2.2%. A different scenario is observed when disturbing traffic is added (the flow T3→T4 favors adaptive routing from East to South, avoiding the disturbing I-J and K-L.). Even if the average latency presents similar results, the routing adaption enabled to reduce the maximum latency. As can be observed in Figure 18 (b), negative-first and west-first present the smaller latency variation.



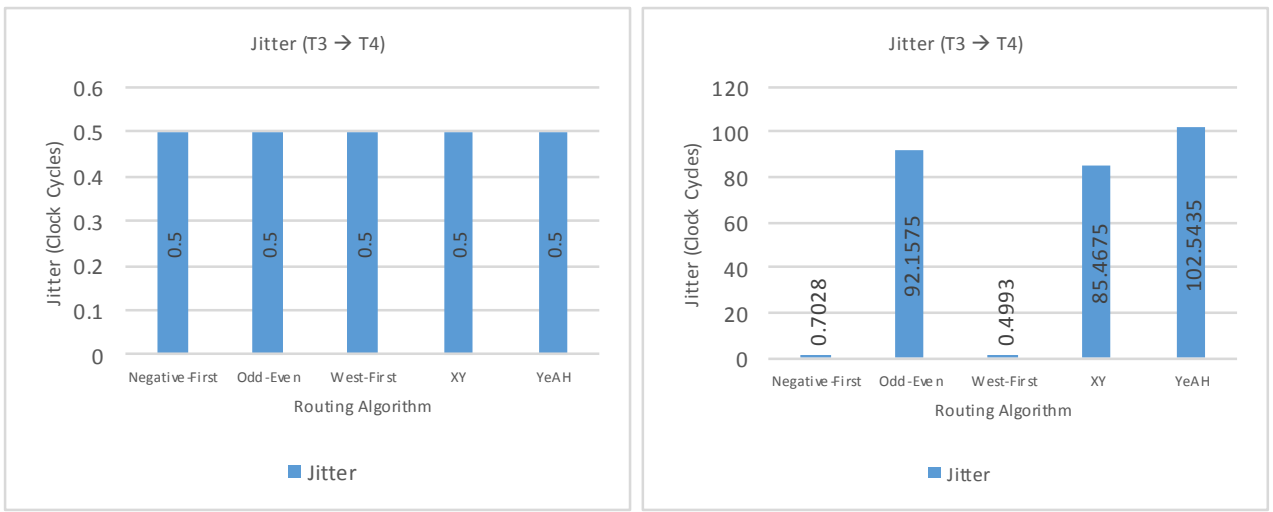
(a) No disturbing traffic

(b) Maximum disturbing traffic

Figure 18 – Average Latency of the flow T3 → T4 for different mesh routing algorithms.

5.1.2.2 Jitter

Again, with no disturbing traffic an almost zero jitter is observed (Figure 19(a)). In Figure 19(b) the jitter of the negative-first and west-first are very small, since adaptation enabled to avoid congested areas (flows I-J and K-L). In this specific case, these partially adaptive algorithms performed much better.



(a) No disturbing traffic

(b) Maximum disturbing traffic

Figure 19 – Jitter of the flow T3 → T4 for different mesh routing algorithms.

5.1.2.3 Execution Time

The impact on the overall execution time when changing the routing algorithm was also minimal. Table 2 presents the execution time varying the buffer depth.

Table 2 – Execution time (ms) for different mesh routing algorithms.

	Execution time (ms) without disturbing traffic					Execution time (ms) with disturbing traffic				
	NF	OE	WF	XY	YeAH	NF	OE	WF	XY	YeAH
Start	55.5	55.5	55.5	55.5	55.6	56.3	56.2	56.3	56.3	56.1
Ivlc	56.0	56.0	56.0	56.0	56.0	56.7	56.7	56.7	56.7	56.5
Idct	56.1	56.1	56.1	56.1	56.1	56.8	56.7	56.8	56.8	56.6
Iquant	56.1	56.1	56.1	56.1	56.2	56.8	56.8	56.8	56.9	56.7
Print	56.8	56.8	56.8	56.8	56.9	57.6	57.5	57.6	57.6	57.4

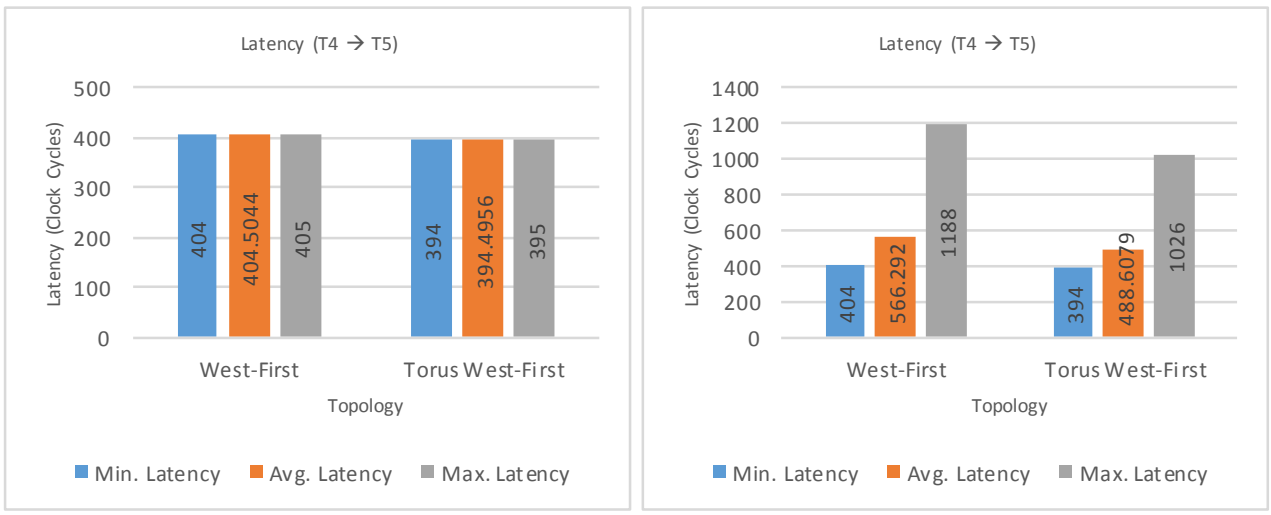
The routing algorithm in mesh networks is the second NoC parameter evaluated. The net effect of the routing algorithm seems to be small. However, it may strongly reduce the jitter, an important parameter for applications with QoS. Therefore, the Authors recommend to adopt MPSoCs with adaptive routing algorithms, since they potentially may benefit applications.

5.1.3 Topology Comparison

The topology comparison consists in finding out what is the major performance difference between these architectures. The major weakness of the torus topology is the wraparound wires, which may be mitigated using a folded torus topology, increasing the wire length twice compared to the mesh topology. The torus architecture should present a smaller latency and jitter when packets travel from one edge to the other of the mesh. To highlight the advantage of the torus topology it was chosen as observed flow the communication T4→T5 (see Figure 15). To evaluate just the effect of the topology we used the same routing algorithm for both simulations, the West-First routing algorithm.

5.1.3.1 Latency

Flow T4→T5 presents 1 hop in the torus topology and 3 hops in the mesh topology. In the scenario with no disturbing traffic (Figure 20 (a)) the latency is constant, being 404 and 394 clock cycles for the mesh and torus topologies respectively.



(a) No disturbing traffic

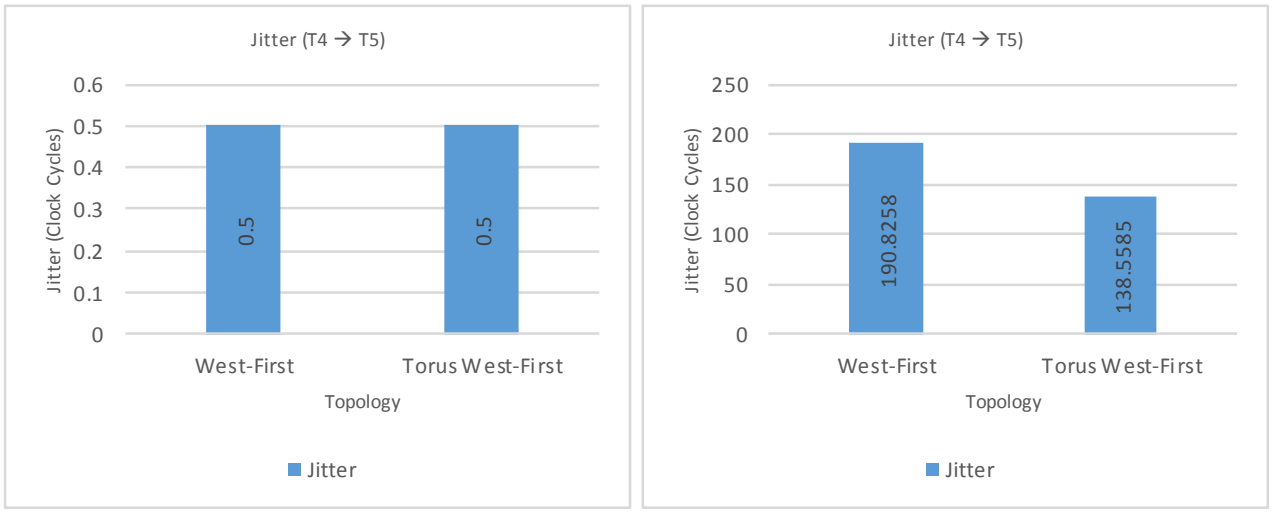
(b) Maximum disturbing traffic

Figure 20 – Minimal, Average and Maximal Latencies of the flow T4 → T5 for different topologies.

The mesh topology, in this scenario, has disturbing flows E-F and H-G in the path, presenting higher latency variation. The torus topology presents an average latency 16.8% smaller, with a small variation. The observed variation comes from the PE sharing, between tasks *print* (T5) and D.

5.1.3.2 Jitter

As expected, with no disturbing traffic the jitter is almost zero (Figure 21 (a)). The jitter with disturbing traffic (Figure 21 (b)) is 37.7% smaller with the torus topology.



(a) No disturbing traffic

(b) Maximum disturbing traffic

Figure 21 – Jitter of the flow T4 → T5 for different topologies.

5.1.3.3 Execution Time

The impact on the overall execution time when changing the routing algorithm was also minimal. Table 3 presents the execution time varying the buffer depth.

Table 3 – Execution time (ms) for different topologies.

	Execution time (ms) without disturbing traffic		Execution time (ms) with disturbing traffic	
	Mesh Topology	Torus Topology	Mesh Topology	Torus Topology
Start	55.5	55.6	56.3	56.1
Ivlc	56.0	56.0	56.7	56.5
Idct	56.1	56.1	56.8	56.6
Iquant	56.1	56.2	56.8	56.7
Print	56.8	56.9	57.6	57.4

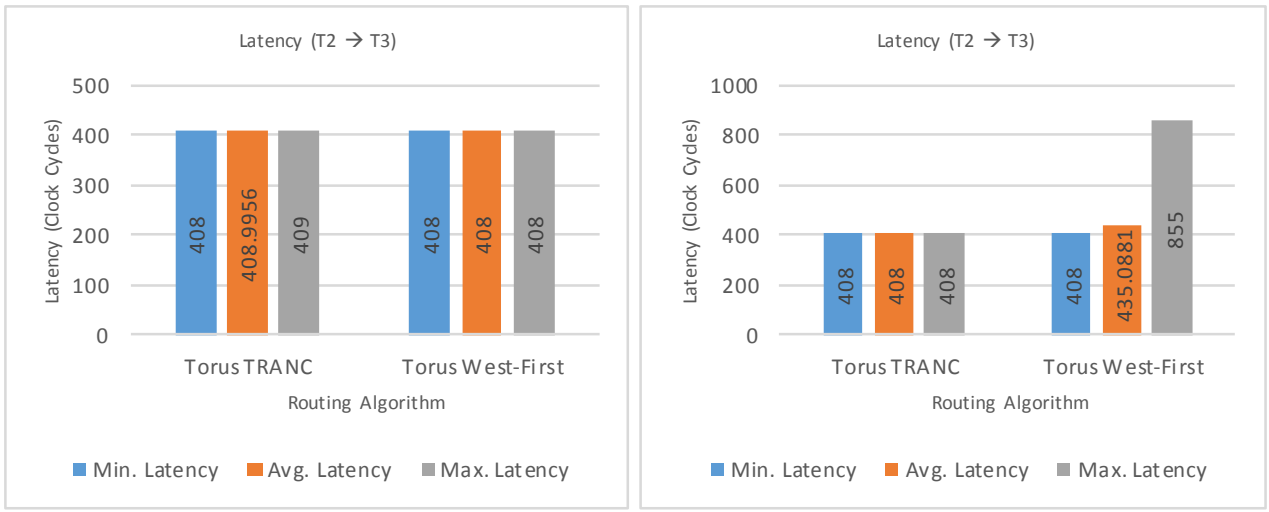
The topology is the third NoC parameter evaluated. In a homogeneous MPSoC, applications hardly could benefit from a torus topology, since mapping heuristics have as cost function to put communicating tasks as close as possible (obviously it is possible to create dedicated mapping heuristics for torus topologies). However, the torus is a choice if IPs have fixed mapping at the edges of the mesh, as shared memory controllers. In such case, the access time for such memories would benefit from a torus topology.

5.1.4 Torus Routing Algorithms

This section evaluates how the routing algorithm affects the network latency and jitter in torus networks. Two routing algorithms are evaluated, being the TRANC algorithm and the West-First algorithm. The results are extracted from a network with 16-flit buffers depth.

5.1.4.1 Latency

Figure 27 presents latency values, for flow T2→T3. With no disturbing traffic, a constant latency is observed. With disturbing traffic, the TRANC algorithm shows an advantage of 6.6% when compared to the West-First routing algorithm. The west-first takes the horizontal path first, thus being affected by the disturbing CD. The TRANC takes the vertical path first, being not affected by the disturbing traffic.



(a) No disturbing traffic

(b) Maximum disturbing traffic

Figure 22 – Average Latency of the flow T2 → T3 for different torus routing algorithm.

Using another flow, T4→T5, Figure 23, the west-first algorithm had an advantage of 4.4% compared to the TRANC algorithm. In this situation, the west-first used the wraparound link, while the TRANC not.

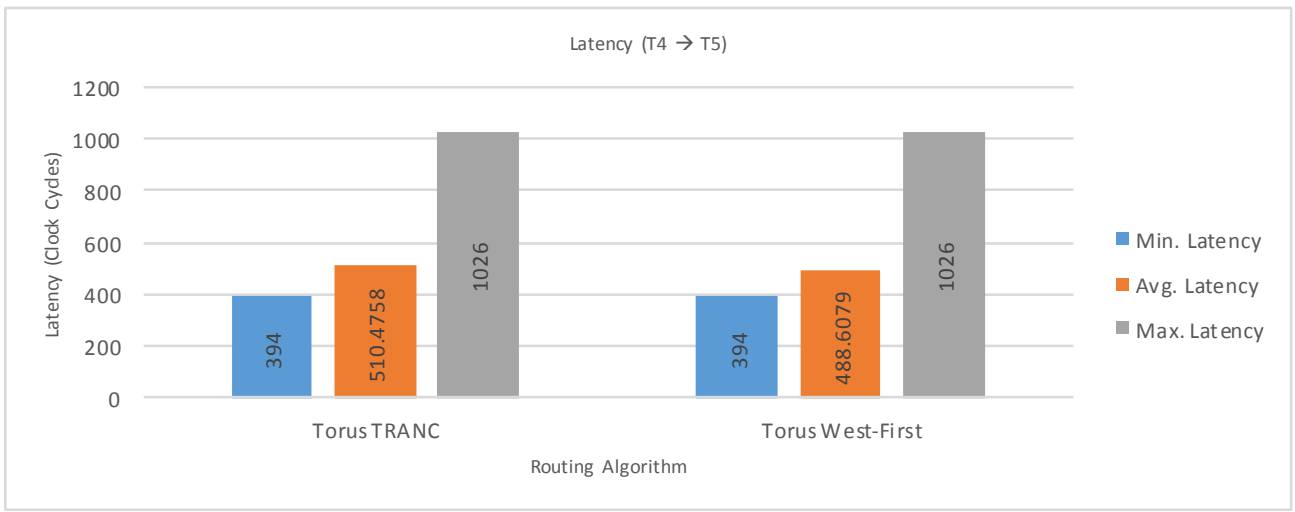
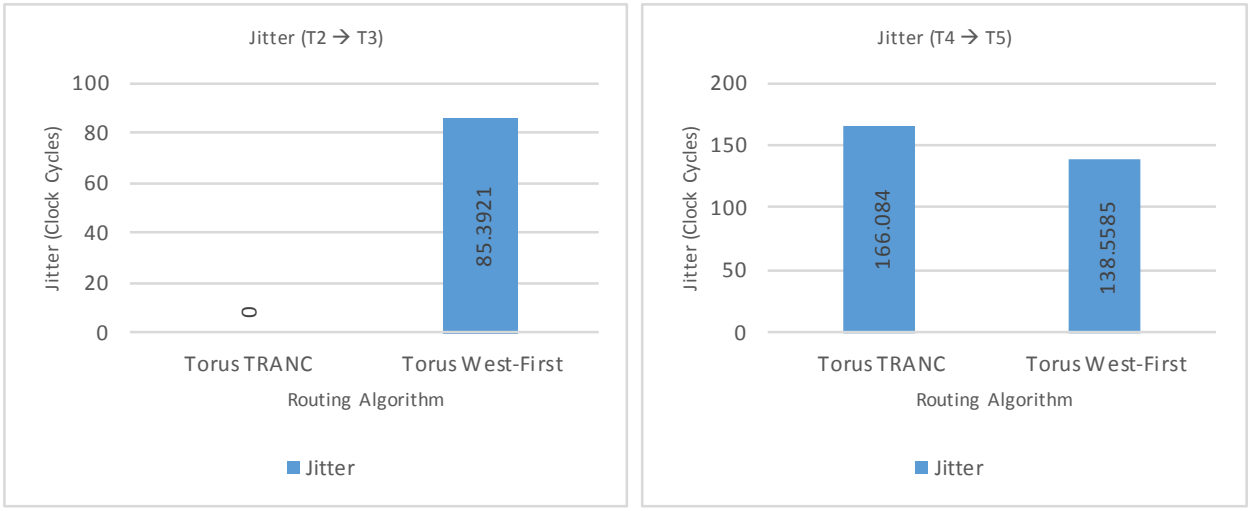


Figure 23 – Average Latency of the flow T4 → T5 with all disturbing flows in the path for different torus routing algorithm.

5.1.4.2 Jitter

Figure 29 compares the jitter of both flows (T2→T3 and T4→T5) with disturbing traffic, demonstrating that the jitter is a function of the task mapping coupled to the adaptation strategy. Therefore, there is not a general rule to select the best routing algorithm.



(a) Flow T2→T3 (b) Flow T4→T5

Figure 24 – Jitter for two flows, with maximum disturbing traffic for different torus routing algorithm.

5.1.4.3 Execution Time

The impact on the overall execution time when changing the routing algorithm was also minimal. Table 4 presents the execution time varying the buffer depth.

Table 4 – Execution time (ms) for different torus routing algorithms.

	Execution time (ms) without disturbing traffic		Execution time (ms) with disturbing traffic	
	Torus TRANC	**Torus West-First**	**Torus TRANC**	**Torus West-First**
Start	55.6	55.6	56.1	56.1
Ivlc	56.0	56.0	56.5	56.5
Idct	56.1	56.1	56.6	56.6
Iquant	56.2	56.2	56.6	56.7
Print	56.9	56.9	57.3	57.4

5.1.5 Torus vs. Partially Adaptive Algorithm vs. Arbitration vs. Original HeMPS

This section contains a comparison between the main parameters that were observed during this work. As the buffer depth presents the same effect in all sizes, it is not going to be compared. When comparing the Torus, we will be using the TRANC algorithm, as it presented the best results compared with the West-First. For partially adaptive algorithms, we will use the West-First as it presented similar results to the Negative-First and better results than the Adaptive Odd-Even algorithm. Distributed arbitration is chosen, using the YeAH router. All those parameters are compared with the original HeMPS architecture, which uses the XY routing algorithm, centralized arbiter, with a mesh architecture. By doing this comparison, we will be estimating which is the parameter that affects mostly the performance.

5.1.5.1 Latency

The latency is one of the most important performance metric, as it affects jitter and throughput. From previous results, without disturbing traffic, latency values are almost the same, regardless the NoC parameters. Figure 25 presents the latency values applying disturbing traffic, using flow T4→T5 as example.

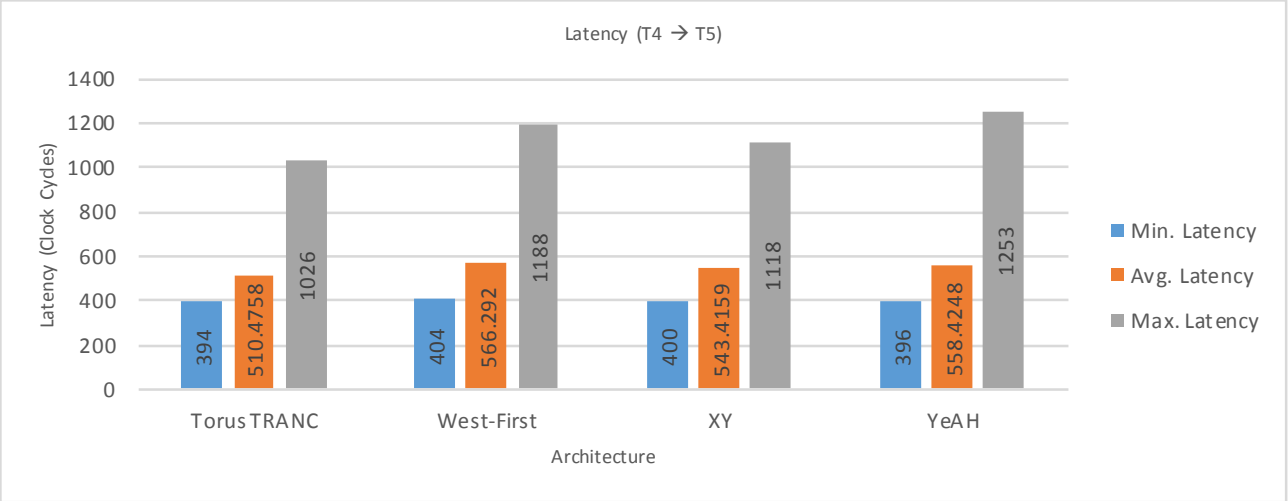


Figure 25 – Overall comparison of latency in the flow T4 → T5

The torus topology with the TRANC algorithm presents an improvement of 10% related to the original HeMPS. Which matches with the differences previously observed when comparing mesh and torus architectures. Comparing the routing algorithms (west-first versus XY) and the arbitration policy (YeAH *versus* XY) there is no perceptible gains.

5.1.5.2 Jitter

With no disturbing traffic the jitter is almost zero. Although, with the scenario with disturbing traffic, we can see that there is an overall improvement using Torus TRANC compared to the other parameters.

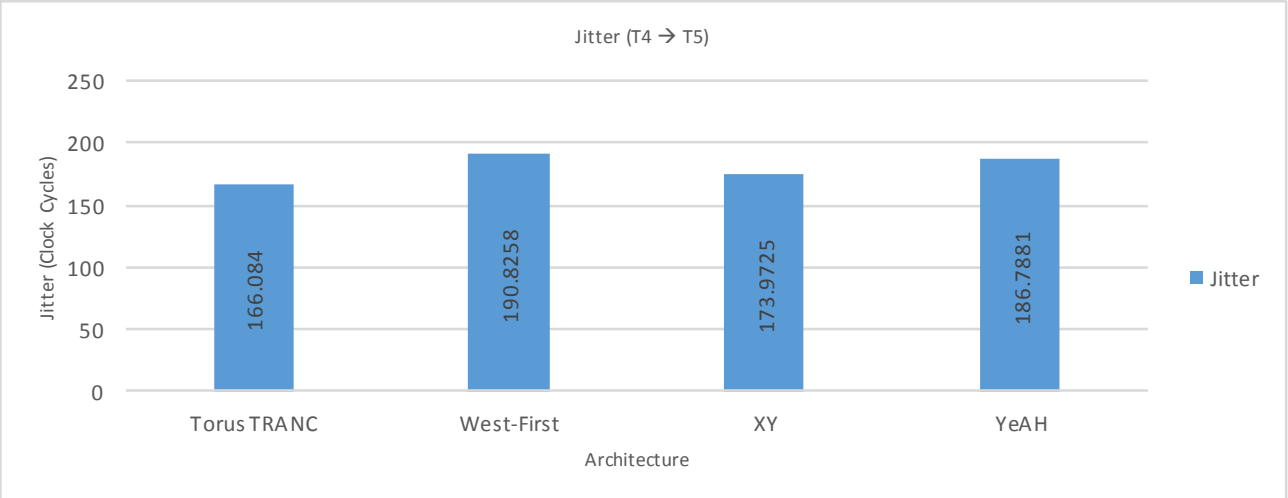


Figure 26 – Overall comparison of jitter in the flow T4 → T5

In our supplementary material, the difference that the Torus TRANC presents in relation to the other flows, is much greater in some specific cases, when the packets must travel from edge to edge of the mesh. As it was stated earlier, the usage of it shall present an overall higher overall performance, although it still presents an area increase, which can be reduced by the usage of a folded Torus topology.

5.1.5.3 Execution Time

The execution time is not affected varying the NoC parameters, as presented in Table 5.

Table 5 – Execution time (ms) for different architectures.

	Execution time (ms) without disturbing traffic				Execution time (ms) with disturbing traffic			
	TRANC	WF	XY	YeAH	TRANC	WF	XY	YeAH
Start	55.6	55.5	55.5	55.6	56.1	56.3	56.3	56.1
Ivlc	56.0	56.0	56.0	56.0	56.5	56.7	56.7	56.5
Idct	56.1	56.1	56.1	56.1	56.6	56.8	56.8	56.6
Iquant	56.2	56.1	56.1	56.2	56.6	56.8	56.9	56.7
Print	56.9	56.8	56.8	56.9	57.3	57.6	57.6	57.4

The application chosen by the Authors, an MPEG decoder, is a representative application for MPSoCs. As a general conclusion for this set of experiments, the bandwidth offered at each NoC link NoC (1.6 Gbps) is enough to support the traffic of this application, together with disturbing flows. Therefore, the design of the NoC router may be the simplest one, reducing the area and power overhead due to the communication infrastructure. It is important to remember NoCs offer parallels transactions and scalability, features not supported by bus-based infrastructures.

5.2 Synthetic Application

A second set of simulations was executed, using the same producer/consumer application as the main application. The goal is to have an application with a higher injection rate, since we were not able to fully stress the network using the MPEG decoder with the disturbing flows. This experiment cannot be seen as general scenario, its goal is to investigate how the NoC supports applications supposed to congest the NoC.

The results using the synthetic applications can be fully viewed on our supplementary material. For the latency, the synthetic scenario follows the tendencies observed in the first set of experiments. For example, the torus has a better latency response in comparison with the mesh. Related to the routing algorithms in the mesh topology, the partially adaptive algorithms are benefited when they can avoid congested areas.

Nonetheless, we were not able to find a change in the results, comparing both scenarios. As a general conclusion, for NoC-based MPSoCs, routers offer enough bandwidth and internal NoC congestion is not a relevant issue.

6 CONCLUSIONS

"Don't cry because it's over. Smile
because it happened."

Dr. Seuss

The work described in this document has produced a set of techniques and systems with the objective to evaluate which parameters affect applications running in NoC-based MPSoC. In order to find out which are those parameters, the HERMES Multiprocessor System-on-Chip was used as case of study.

The first contribution of this work are the monitoring tools. To collect performance data from HeMPS, a set of non-intrusive monitors were created and placed in several variations of HERMES routers. The *data link monitors* record traffic statistics from all NoC links (North, East, West and South), and the *local monitor* record statistics at the local ports. These monitors generate text files during simulation. These text files are then used as inputs to our set of tools that generates the necessary metrics to analyze each parameter of the network. The tools consist in one graphical tool (datalink.py) that can identify hotspots, and the path taken by the packets in the NoC. The second tool (local.py) generates reports to obtain the metrics required to make the system analysis. These tools were extensively validated with different routing algorithms, verifying if the paths taken by packets were correct, and if the number of transmitted packets was correct.

The second contribution of this work corresponds to the evaluation of NoC architectural parameters in MPSoC performance. Due to limited time to develop this work, some features were not explored, such as virtual channels and duplicated physical channels. A test case scenario was designed to test the NoC parameters, with an application disturbed by different synthetic flows. This test case scenario was designed to benefit the torus topology, adaptive routing algorithms and distributed arbitration, thus exposing the possible advantages of such characteristics in MPSoC performance. The complexity of this task was immense, since it was necessary to create different MPSoC instances (adapting it to new routing algorithms, new NoC topologies, etc), and execute thousands of simulations. The use of scripts and a grid infrastructure to simulate the MPSoC instances allowed to obtain the extensive set of results presented in the supplementary material (2.112 charts).

Results point out that the use of the specific real applications does not generate enough traffic to fully stress the employed NoC. For example, the average injection rate of the MPEG decoder was around 1% of the available individual channel bandwidth. Hence, a second test cased was developed using a synthetic producer/consumer application as the application under evaluation. Such scenario achieved, in average, 15% of the available individual channel bandwidth.

Analyzing the obtained results, it was observed that the difference that appears when using different buffer sizes is minimal, meaning that the use of a smaller buffers is advantageous in terms of area without significant performance loss. Therefore, this work recommends the usage of small buffers, with 4 or 8-flit depth. Moreover, the use of partially adaptive algorithms can be advantageous in some cases, since the jitter is reduced. The use of the torus topology is recommended for performance improvement, but it presents higher area overhead.

Finally, as proposed in the beginning of this work, we were able to determine how parameters of the NoC affect the MPSoC applications. The answer is that the topology has a greater impact, and a combination of a torus topology, with a routing algorithm designed for this topology with the use of a small buffer will bring a reduction on the routers' area without performance loss in relation to the traditional mesh, XY, 16-flit buffer depth NoC, used today by HeMPS.

REFERENCES

[BEN02] Benini, L.; De Micheli, G. *Networks on chips: a new SoC paradigm*. Computer, v.35(1), 2002, pp. 70-78.

[CAR09] Carara, E.; Oliveira, R.; Calazans, N.; Moraes, F. *HeMPS: A Framework for NoC Based MPSoC Generation*. In: ISCAS, 2009, pp. 13456-1348.

[CAS13] Castilhos, G. *Gerência distribuída de recursos em MPSoCs – Mapeamento e Migração de Tarefas*. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2013, 66 p.

[DAL93] Dally, W.J.; Aoki, H. *Deadlock-free adaptive routing in multicomputer networks using virtual channels*, Parallel and Distributed Systems, IEEE Transactions on , vol.4, no.4, pp.466,475, Apr 1993

[DAR12] Dara Rahmati, Hamid Sarbazi-Azad, Shaahin Hessabi, and Abbas Eslami Kiasari. 2012. *Power-efficient deterministic and adaptive routing in torus networks-on-chip*. Microprocess. Microsyst. 36, 7 (October 2012), 571-585.

[DUA01] Duato, J.; Yalamanchili, S.; Ni, L. *Interconnection Networks: an engineering approach*. Morgan Kaufmann, Revised Edition, 2002, 624 p.

[HWA92] Hwang, K. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, New York, 1992, 672 p.

[JER05] Jerraya, A.; Wolf, W. *Multiprocessor Systems-on-Chips*. Morgan Kaufmann Publishers Inc, 2005, 602p.

[KON13] Kornaros, G.; Pnevmatikatos, D. *A Survey and Taxonomy of On-Chip Monitoring of Multicore Systems-on-Chip*. ACM Transactions on Design Automation of Electronic Systems, v.18(2), 2013, 38p.

[KUM13] Kumar, R.; Deshpande, H.; Choi, G.; Sprintson, A.; Gratz, P. *Bidirectional interconnect design for low latency high bandwidth NoC*. In: ICICDT, 2013, pp. 215-218.

[MEL04] Mello, A; Ost, L.; Calazans, N.; Moraes, F. *Evaluation of Routing Algorithms in Mesh Based NoCs*. PPGCC-PUCRS Technical Report Series, 2004, 11 p.

[MEL05] Mello, A.; Möller, L.; Calazans, N.; Moraes, F. MultiNoC: *A multiprocessing system enabled by a network on chip*. In: DATE users forum, 2005, pp. 234-239.

[MOD11] Modarressi, M.; Tavakkol, A.; Sarbazi-Azad, H. *Application-Aware Topology Reconfiguration for On-Chip Networks*. IEEE Transactions on Very Large Scale Integration Systems, v.19(11), 2011, pp. 2010-2022.

[MOR04] Moraes, F.; Calazans, N.; Mello, A; Möller, L.; Ost, L. *Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip*. Integration, the VLSI Journal, v.38(1), 2004, pp. 69-93.

[NI93] Ni, L.; McKinley, P. *A Survey of Wormhole Routing Techniques in Direct Networks*. Computer, v. 26(2), 1993, pp. 62 –76.

[OST03] Ost, L. *Redes Intra-Chip parametrizáveis com interface padrão para síntese em Hardware*. Dissertação de Mestrado, Programa de Pós-Graduação em Cinência da Computação, FACIN-PUCRS, 2003, 102p.

[PAS08] S. Pasricha; N. Dutt. *On-Chip Communication Architectures*. Morgan Kauffman, 2008, 544p.

[PLA11] *Plasma CPU*. Available at: http://plasmacpu.no-ip.org/

[SCH10] Schranzhofer, A.; Jian-Jia C.; Santinelli, L.; Thiele, L. *Dynamic and adaptive allocation of applications on MPSoC platforms*. In: ASP-DAC, 2010, pp. 885-890.

[STE12] Stefan, R.; Molnos, A.; Ambrose, A.; Goossens, K. *A TDM NoC supporting QoS, multicast, and fast connection set-up*. In: DATE, 2012, pp. 1283-1288.

[WOL04] Wolf, W. *The Future of Multiprocessors System-on-Chips*. In: DAC, 2004, pp. 681-685.

[WOL12] M. Wolf. *Computers as components: principles of embedded computing system design*. New York: Morgan Kaufmann Publishers, 2012, 528p.

[WOS07] Woszezenki, C. *Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs*. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, FACIN-PUCRS, 2007. 121p.

[ZAM11] Zamzam, D.; Abd El Ghany, M.A.; Hofmann, K.; Ismail, M. *Highly reliable and power efficient NOC interconnects*. In: *NORCHIP, 2011*, pp. 1-4.

[ZEF03] Zeferino, C. A. *Redes-em-Chip: Arquiteturas e modelos para avaliaçao de área e desempenho*. Tese de Doutorado, Programa de Pós-Graduação em Computação, UFRGS, 2003, 194p.