Dottorato di Ricerca in Informatica
Università di Bologna, Padova

Settore scientifico disciplinare: INF/01
Ciclo XXI

Design and Performance Evaluation of Network-on-Chip Communication Protocols and Architectures

Nicola Concer

March 2008

Coordinatore:
Prof. Simone Martini

Relatore:
Dott. Luciano Bononi

______________________ ______________________

Abstract

The scale down of transistor technology allows microelectronics manufacturers such as Intel and Ibm to build always more sophisticated systems on a single microchip. The classical interconnection solutions based on shared buses or direct connections between the modules of the chip are becoming obsolete as they struggle to sustain the increasing tight bandwidth and latency constraints that these systems demand.

The most promising solution for the future chip interconnects are the *Networks on Chip* (NoC). NoCs are network composed by routers and channels used to interconnect the different components installed on the single microchip.

Examples of advanced processors based on NoC interconnects are the Ibm *Cell* processor, composed by eight CPUs that is installed on the Sony *Playstation III* and the Intel *Teraflops* project composed by 80 independent (simple) microprocessors.

On chip integration is becoming popular not only in the *Chip Multi Processor* (CMP) research area but also in the wider and more heterogeneous world of *Systems on Chip* (SoC). SoC comprehend all the electronic devices that surround us such as cell-phones, smart-phones, house embedded systems, automotive systems, set-top boxes etc...

SoC manufacturers such as ST Microelectronics , Samsung, Philips and also Universities such as Bologna University, M.I.T., Berkeley and more are all proposing proprietary frameworks based on NoC interconnects. These frameworks help engineers in the switch of design methodology and speed up the development of new NoC-based systems on chip.

In this Thesis we propose an introduction of CMP and SoC interconnection networks. Then focusing on SoC systems we propose:

- a detailed analysis based on simulation of the *Spidergon* NoC, a ST Microelectronics solution for SoC interconnects. The *Spidergon* NoC differs from many classical solutions inherited from the parallel computing world. Here we propose a detailed analysis of this NoC topology and routing algorithms. Furthermore we propose *aEqualized* a new routing algorithm designed to optimize the use of the resources of the network while also increasing its performance;
- a methodology flow based on modified publicly available tools that combined can be used to design, model and analyze any kind of System on Chip;
- a detailed analysis of a ST Microelectronics -proprietary transport-level protocol that the author of this Thesis helped developing;
- a simulation-based comprehensive comparison of different network interface designs proposed by the author and the researchers at AST lab, in order to integrate shared-memory and message-passing based components on a single System on Chip;
- a powerful and flexible solution to address the *time closure exception* issue in the design of synchronous Networks on Chip. Our solution is based on relay stations repeaters and allows to reduce the power and area demands of NoC interconnects while also reducing its buffer needs;
- a solution to simplify the design of the NoC by also increasing their performance and reducing their power and area consumption. We propose to replace complex and slow virtual channel-based routers with multiple and flexible small Multi Plane ones. This solution allows us to reduce the area and power dissipation of any NoC while also increasing its performance especially when the resources are reduced.

This Thesis has been written in collaboration with the *Advanced System Technology* laboratory in Grenoble France, and the *Computer Science Department* at Columbia University in the city of New York.

From this work the author published [BCG$^+$07, BC06, CPC08] and submitted for reviw [NMLb, NMLa, MNL09, CSB09]:

- L.Bononi, N.Concer *Simulation and Analysis of Network on Chip Architectures: Ring, Spidergon and 2D Mesh* The Proceedings of the 9-th IEEE Conference on Design, Automation and Test in Europe (DATE 2006), 6-10 March 2006 ICM, Munich, Germany. (ACM DL);

- L.Bononi, N.Concer, M.Grammatikakis *NoC Topologies Exploration based on Mapping and Simulation Models* The Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007);

- N. Concer, M. Petracca, and L.P. Carloni *Distributed Flit-Buffer Flow Control for Networks-on-Chip* The Proceedings of the Sixth International Conference on Hardware/Software Codesign & System Synthesis (CODES+ISSS), 2008;

- N.Concer, S.Iamundo and L.Bononi *aEqualized a novel routing Algorithm for Spidergon NoC* The Proceedings of the 12-th IEEE Conference on Design, Automation and Test in Europe (DATE 2009), April 2009 ICM, Nice, France. (ACM DL);

- L. Bononi, N. Concer, and M.D. Grammatikakis, *"System-Level Tools for NoC-based multicore sesign"* in Embedded Multicore Architectures. Ed. G. Kornaros, Chapter 6, CRC Press (Taylor and Francis Group), 40 pages, 2009, to appear.;

- M.Petracca, N.Concer and L.P.Carloni *Multi-Planes vs. Virtual Channel: a Comparative Analysis* **"submitted for conference publication"**;

- N.Concer, M.Petracca and L.P.Carloni *Wire Pipelining on Synchronous NoC* **"submitted for journal publication"**;

Contents

List of Figures

Part I

Networks on Chip

Chapter 1

Introduction

Introduction

This thesis addresses the analysis and design of algorithms and protocols for Network on Chip (NoC) interconnection systems. We propose to investigate the following research issues: *(i)* Architecture analysis: we intend to evaluate the main NoC architectures proposed in Literature such as 2D Matrix, Ring, Crossbar and the novel *Spidergon* NoC to understand their main characteristic and the cases where they constitute the best choice. *(ii)* protocol design: investigation and proposal of novel transport and routing algorithms for the *Spidergon* Network on Chip. *(iii)* Network design: investigate the major issues in the actual NoC implementation and interconnection. The analysis and the characterization of protocols and architectures for NoC systems that we propose throughout this Thesis have been obtained through computer-based simulation.

According to the International Technology Roadmap for Semiconductors projections, by the end of this decade complex systems, called *Multi Processor System-on-Chip* (MPSoC), will contain billions of transistors running at a frequency of many GHz [GDvM+03, BM02].

As depicted in Figure 1.1 the technology of semiconductors keeps on scaling down allowing more and more components to be installed within the same area of a chip.

As a consequence complex systems that once required many microchip for being

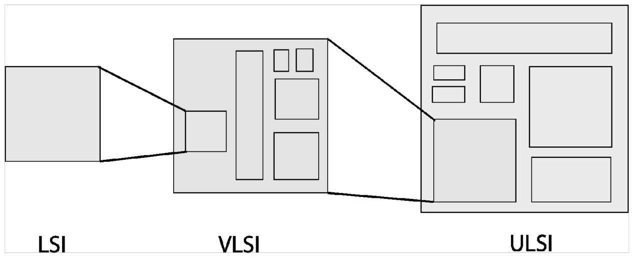


Figure 1.1: Improvement in the semiconductor technology leads a scale down of the components on a chip.

built, now can be installed on a single microchip containing all the logic of the system and the interconnection channels connecting them. Examples of these capabilities are the recent eight-cores IBM's *Cell* processor installed on the SONY's *Playstation III* [KPP06] and the more futuristic eighty cores INTEL's *Teraflop* [H+07] processors.

A central and key element in future complex MPSoC is the global *On-Chip Communication Architecture* (OCCA) or *On Chip Interconnect* (OCIN): the infrastructure that interconnects the components of a MPSoC and provides the means necessary for distributed computation among different processing elements [DT04].

The natural evolution of the bus-based solution reported in Figure 1.2(a) and the poorly scalable point to point networks seen in Figure 1.2 (b) are the new generation architectures called *Network on-Chip* (NoC) represented in Figure 1.2 (c) [MB06, DT01, HJK+00, NTIJ04].

Examples of innovative NoC architectures include the LIP6 *Spin* [AAZ03], the M.I.T. *Raw* [T+02], the VTT (and various Universities) *Eclipse* [For02] and *Nostrum* [GRK+05], PHILIPS's *Æthereal* NoC [GDvM+03], Stanford/Bologna Universities' *Netchip* [BM02, JMBM04] and ST MICROELECTRONICS ' Spidergon NoC [CLM+04, MRG+04, CGL+08, BC06].

NoC are packet-switched communication networks derived from the parallel computing domain. They are based on a well-defined protocol stack similar to the *ISO/OSI* seen in the network on computers.

A layered-stack approach to the design of the on-chip inter-core communications can be defined accordingly with the communication-based methodology that will

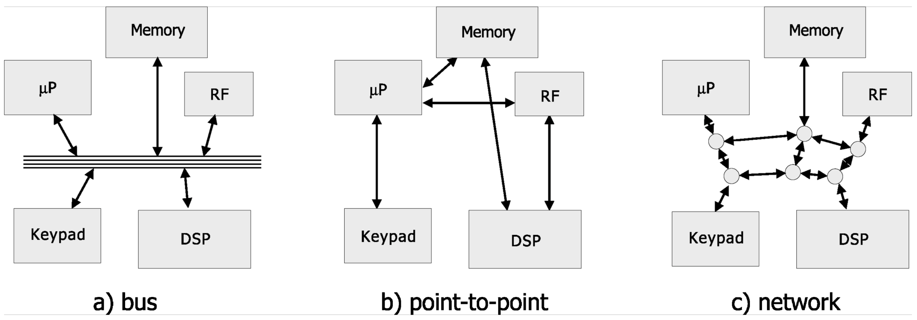


Figure 1.2: Examples of communication structures in Systems-on-Chip. a) traditional bus-based communication, b) dedicated point-to-point links, c) a chip area network.

be conceived for the system. Exploiting the lesson learned by telecommunication community, the global on-chip communication is decomposed into layers similar to the *ISO-OSI* Reference Model (see Figure 1.3). The protocol stack enables different services, providing to the programmer an abstraction of the communication framework. Layers interact through well-defined interfaces and they hide the low level details [DT01, MB06].

- **The Physical layer**: refers to all that concerns the electric details of wires, the circuits and techniques to drive information (drivers, repeaters, layout. . .);
- **Data link**: ensures a reliable transfer despite of the physical unreliability and deals with medium access control for sharing a common channel resource, with contention-based access;
- **Network level**: handles issues related to the topology and the consequent routing scheme;
- **Transport layer** manages the end-to-end services and the packet segmentation/re-assembly.
- **Upper levels** can be viewed merged up to the Application as a sort of adaptation layer that implements (in HW or through part of an OS) services and

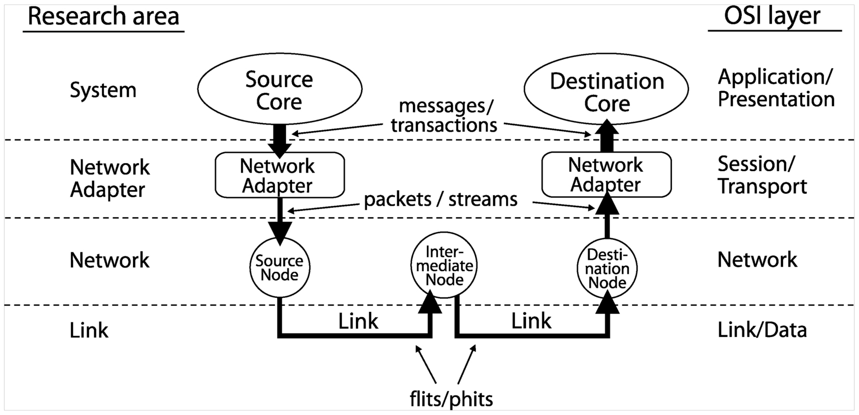


Figure 1.3: ISO-OSI Reference Model.

exposes the NoC infrastructure according to a Message Passing (MP) programming paradigm.

Despite the similarity discussed above, it is clear that the micro-network in the single chip domain differs from the wide-area networks under many different aspects:

- spatial locality of modules of the system: NoC components are installed on the same microchip so distances are minimal and predictable while the uniformity of the channels grant relatively reliable communications;
- reduced non-determinism of the on-chip traffic: as we will see NoC systems are divided in two main classes depending on the knowledge about the traffic they have to support. For all purpose NoC the traffic can vary depending on the application that is currently running. SoC networks instead are characterized by well known traffic patterns that are known at design time. The NoC then can be designed to exactly match the requirement of the application;
- energy and power constraints: NoC systems are designed to support either AC or battery-powered systems. Controlling and minimizing the need of power is a vital aspect especially for portable systems such as phones and multimedia devices that will heavily relay on NoC interconnects;

- need of low cost and low complexity solutions: NoC interconnects will be extensively used in many commercial-oriented electronic devices. The minimization of costs of all sort is an important aspect of for the development of these systems;

- reusability: by providing standard Network Interfaces the same network can be used to interconnect heterogeneous components that implement different tasks or algorithms and that are designed by third party companies.Examples of such standard interfaces are *Open Core Protocol* (OCP) [OI], *Virtual Component Interface* (VCI) used in the SPIN and Proteo [STAN04] NoCs *Advanced eXtensible Interface* (AXI) [AA] and *Device Transaction Level* (DTL) [PS];

Depending on the knowledge at design time about the traffic that a network will have to support we distinguish two main families of Network on Chip: possible solutions.

- **Chip Multi Processor**: (CMP) are networks designed to support any kind of traffic. At the network's design time no knowledge about the traffic is available. CMP hence are built to offer best effort services. Quality of services (QoSs) capabilities can be granted by differentiating the traffic into classes of priorities [GDvM$^+$03] each one assigned to a specific virtual channel (VC)[1] [TS04,GDvM$^+$03] or by providing guaranteed service levels on dedicated connection [TJ05]. CMP systems are mainly composed by similar components grouped such as sets of processors and memories. This allows the use of regular topologies often borrowed from the parallel computing world such as 2D Mesh, Torus, Ring etc… [DT04, MB06, DYN03];

- **System on Chip**: (SoC) are systems integrating heterogeneous components, often developed by third party companies at the purpose of building an *application specific systems*. In SoC often the traffic patterns are known since the design time hence the interconnection networks can be build to exactly match

[1]see Section 2.1.5

the application requirement [BJM+05, JBMM04, SCK07, Pin08]. In SoC for embedded applications designers use standard industrial CAD-tool flows for the synthesis of a platform-specific NoC and must cope with an increasing number of timing-closure exceptions due the differences in size across its heterogeneous processing cores. This problem becomes particularly hard when using nanometer technology processes [P+07] as the impact of global interconnect wires raises exponentially the number of *wire exceptions*, i.e. timing-closure violations due to the delay of a global wire exceeding the *target clock period* (the clock at which the system is deigned to run) T_{clk} [CSV02, HMH01].

The research results discussed in this Thesis will be mainly focused on System on Chip architectures.

1.1 Design Challenges

1.1.1 Protocol Design

The paradigm for interconnection systems based on Networks on Chip requires the design of new communication protocols. Despite NoC can be seen as a special case of a parallel computing architecture, these systems are characterized by tight constraints on the complexity of the adopted algorithms (because they have to be implemented electronically) and the available resources. For these reasons most of the well known parallel computing algorithms and protocols for parallel architectures appear excessively expensive or impossible to implement in a real NoC microchip. More specifically, beside many others not related with the computer science field, NoC interconnect require the development of new algorithms and protocols to solve the following issues:

- **Routing**: Routers are implemented as embedded circuits into the NoC interconnect. For this reason they have to be simple, fast and their buffer requirements have to be reduced to the minimal. For these reasons routers often adopt a forwarding scheme called *wormhole* [DYN03, DT04]. Wormhole allows a deep pipeline and a reduced buffering cost. Packets are divided into basic units called flits; the queues in each node have the flit granularity and the physical node-to-node links are managed by a flow control that works on a flit per flit basis;

- **Deadlock**: is an important issue that has to be solved by smart solutions. Parallel computing systems address the deadlock problem by adopting complex solutions based on Virtual Channels (VC) [DYN03,PD01]. Although still valid for the NoC domain, these techniques are expensive to realize so should be reduced to the minimum indispensable;

- **Mapping and resource allocation**: A major challenge for predicting performance and scalability of a particular NoC architecture relies on precise specification of real application traffic requirements arising from current and future applications as well as the scaling of existing applications. In example, it has been estimated that SoC performance varies by up to 250% depending on NoC design, and up to 600% depending on communication traffic [LRD01], while NoC power dissipation can be reduced by more than 60% by using appropriate mapping algorithms [HM03].

- **Data sharing**: often solving the deadlock problem at network level does not completely address the problem of circular dependencies among the resources of a system. Considering the Transport or higher layers of the ISO/OSI stack presented in Figure 1.3 in fact, a new kind of deadlock, called *Protocol or message-dependent Deadlock* [SP03a, D.K97]) can arise. Protocol deadlock happens when Processing Elements (PEs) and Storage Elements (SEs) share the same communication channels to send they relative messages. In example in the shared memory paradigm PEs send request messages to SEs which in turn replay by sending data or acknowledgements packets. If both the request and the replay messages share the same NoC channels and the network does not use a specific flow control protocol to guarantee that the sent messages will be accepted on the peer side, the system may fall in a deadlocked situation generated by the additional dependencies between input and output channels of the SE nodes. In Chapter 5 we will present the solution we proposed to address this issue;

- **System Synchronization**: NoC can be implemented as synchronous, mesochronous or asynchronous systems. In the first case all the component of the system work following the same clock signal. In the mesochronous case each component work at the same frequency but possibly with different phases while in the third case there are no restrictions upon the frequency of any component. In this Thesis we consider only synchronous cases. Here the shared clock must be set so that the messages exchanged throughout the channel of the NoC have enough time to traverse the link and be correctly delivered. This is a difficult property that by the shrinking of technology and the increase of working frequency requires ad-hoc solution to be guaranteed. In Chapter 7 we will discuss this problematic in detail and propose a solution to it.

1.1.2 Architecture Analysis

The most frequently used on-chip interconnection architecture is the shared medium arbitrated bus, where all communication devices share the same transmission medium. The advantages of the shared-bus architectures are simple topology, low area cost, and extensibility. However, for a relatively long bus line signals may not respect the tight synchronism constraints [MB06]. Moreover, every additional IP block connected to the bus causes increased propagation delay and eventually it could exceed the targeted clock period. This, in practice, limits the number of IP blocks that can be connected to a bus and thereby limits the system scalability [MB06, DT01].

To overcome the above-mentioned problems, several research groups have advocated the use of a communication-centric approach to integrate IPs in complex SoCs. The Network on Chip upon which the SoC relies has to maximize the parallelism of the system while minimizing the complexity of the routers, the resource requirements (buffers) and the overall system area and power consumption. In Literature we can find many different solutions. One of the purpose of this Thesis it to investigate the proposed solutions in order to understand the main characteristics that make a NoC topology be preferable upon the others.

1.1.3 Simulation

NoC systems are relatively small systems (up to 64 nodes) that have to support highly predictable traffic bursts. Evaluating such a systems then does not require a powerful and number crunching distributed simulation. What is important instead is the flexibility of the model that has to permit the rapid and "easy" integration of new algorithms and protocols in order to test their effective performance. This tool then has to support different NoC topology, Routing, QoS and transport-layer algorithms as well as different traffic patterns and injection loads;

1.2 Contributions of the Thesis

Network on Chip are a relatively new concept. Research in this area is still at the beginning as many issues are still open and many tools are required to better explore design and analyze these new concept [GIP+07, JZH].

The work we have been focused on this Thesis has covered multiple aspects that can be reassumed in the following propositions:

- a detailed analysis based on simulation of the *Spidergon* NoC, a ST Microelectronics solution for SoC interconnects. The *Spidergon* NoC differs from many classical solutions inherited from the parallel computing world. Here we propose a detailed analysis of this NoC topology and routing algorithms. Furthermore we propose *aEqualized* a new routing algorithm designed to optimize the use of the resources of the network while also increasing its performance;

- a methodology flow based on modified publicly available tools that combined can be used to design, model and analyze any kind of System on Chip;

- a detailed analysis of a ST Microelectronics -proprietary transport-level protocol that the author of this Thesis helped developing;

- a simulation-based comprehensive comparison of different network interface designs proposed by the author and the researchers at AST lab, in order to integrate shared-memory and message-passing based components on a single System on Chip;

- a powerful and flexible solution to address the *time closure exception* issue in the design of synchronous Networks on Chip. Our solution is based on relay stations repeaters and allows to reduce the power and area demands of NoC interconnects while also reducing its buffer needs;

- a solution to simplify the design of the NoC by also increasing their performance and reduce their power and area consumption. We propose to replace complex and slow virtual channel-based routers with multiple and flexible small Multi Plane ones. This solution allows us to reduce the area and power dissipation of any NoC while also increasing its performance especially when the resources are reduced.

1.3 Thesis Plan

This Thesis is divided in three main parts. In *Part I* we analyze the *Spidergon* network on chip comparing it to a number of different other NoC in particular:

Chapter 2: we discuss the general aspect characterizing the Network on Chip domain emphasizing the aspects critical to this Thesis;

Chapter 3: we analyze in detail the *Spidergon* which we will discuss throughout the whole Thesis. We also present the *aEqualized* routing algorithm, a proposal we made to enhance the use of the network channels reducing costs and improving performances;

Chapter 4: we present the design flow we adopt throughout the Thesis. We describe the tools we have either developed or adopted in order to perform the analysis proposed in this work.

In the the *Part II* of this Thesis we discuss the issue of data streaming support for Network on Chip. We present the analysis we obtained in developing protocols and architectures proposed by ST Microelectronics researcher with who we collaborated throughout our Ph.D. program. In particular:

Chapter 5: We discuss and analyze the Ctc transport protocol,used to regulate the flow of data between two peers in a message passing environment. Ctc is a transport-lever protocol developed by ST Microelectronics in collaboration with the author of this Thesis;

Chapter 6: We discuss and analyze different architectures proposed to combine *shared memory* and *message passing* components on the same Network on Chip.

In *Part III* we discuss and analyze the proposals we made in order to solve the issues related to the *Signal Propagation Delay* and to improve the performance of the system by also reducing power and area consumption. In particular:

Chapter 7: We describe the *Time Closure Exception* issue and we propose a solution based on the use of Relay Stations on the channel. We consider system-level and register-transfer lever (RTL) analysis used to compare both performance and actual chip-area of the proposed solution;

Chapter 8: We propose a new approach based on Multi Planes (MP) used to improve performance while reducing area and power consumption on a NoC. We propose to replace Virtual Channels (VC) with MPs and we show that this solution achieves better performance especially when resources are limited.

Chapter 2

Networks On Chip

Network on Chip are a special case of parallel computing systems characterized by the tight constraints such as resource availability, area and power consumption and cost of the NoC architecture.

Many of the currently adopted architectures and protocols derive directly from the distributed computing research area from which NoC are a special case. Nevertheless new and NoC-specific solutions are currently being published.

2.1 Architectures

2.1.1 Topologies

A key point on the NoC performance is the interconnect topology. A NoC topology should be regular and simple so to allow the use of simple and efficient routing algorithms. Simplicity in fact is directly bounded to the maximum frequency a circuit can run.

Guerrier and Greiner proposed a generic interconnect template called SPIN (Scalable, Programmable, Integrated Network) for on-chip packet switched interconnections, where a fat-tree architecture is used to interconnect IP blocks [AAZ03].
In this fat tree, every node has four children and the parent is replicated four times at any level of the tree. Figure 2.1(a) shows the basic SPIN architecture with N = 16 nodes, representing the number of functional IP blocks in the system. The size

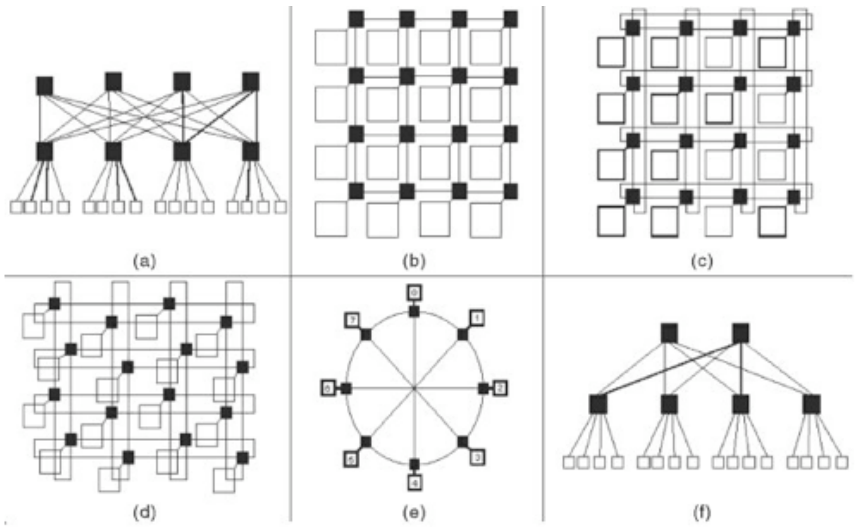


Figure 2.1: NoC Architectures: (a) Spin, (b) 2D Mesh, (c) Torus, (d) Folded Torus, (e) *Spidergon* /Octagon and (f) Butterfly Fat Tree.

of the network grows as $(NlogN)/8$. The functional IP blocks reside at the leaves and the switches reside at the vertices. In this architecture, the number of switches converges to $S = 3N/4$ where N is the system size in terms of number of functional IPs.

Kumar *et al.* proposed a mesh-based interconnect architecture called CLICHÉ (Chip-Level Integration of Communicating Heterogeneous Elements) [KJM+02]. This architecture consists of an $m \times n$ mesh of switches interconnecting computational resources (IPs) placed along with the switches, as shown in Figure 2.1(b) in the particular case of 16 functional IP blocks. Every switch, except those at the edges, is connected to four neighboring switches and one IP block. In this case, the number of switches is equal to the number of IPs. IPs and the switches are connected through communication channels. A channel consists of two unidirectional links between two switches or between a switch and a resource.

Dally and Towles [DT04, DT01] proposed a 2D torus as an NoC architecture, shown in Figure 2.1(c). The Torus architecture is basically the same as a regular mesh but the switches at the edges are connected to the switches at the opposite edge through wrap-around channels. Every switch has five ports, one connected to the local resource and the others connected to the closest neighboring switches. Again, the number of switches is $S = N$. The long end-around connections can yield excessive delays. However, this can be avoided by folding the torus, as shown in Figure 2.1(d).

ST Microelectronics proposed Octagon [KO] and its evolution for NoC Spidergon [CGL+08, CLM+04, BCG+07, MRG+04]. *Spidergon* has a regular and point-to-point topology which is symmetric with vertex- and edge-transitivity.
Thanks to this symmetry all nodes have a global knowledge of the network, allowing for simple routing and scheduling decisions. Thus, the router hardware implementation is simple and routing decisions are fast. Figure 2.1(e) shows a special case of Spidergon/Octagon NoC with 8 nodes and 12 bidirectional links. Each node is associated with a processing element and a switch. Communication between any pair of nodes takes at most $N/4$ hops.

Pande Grecu and Ivanov proposed an interconnect template following a Butterfly Fat-Tree (BFT) [GPIS04] architecture, as shown in Figure 2.1(f). In our network, the IPs are placed at the leaves and switches placed at the vertices. A pair of coordinates is used to label each node, (l, p), where l denotes a nodes level and p denotes its position within that level. In general, at the lowest level, there are N functional IPs with addresses ranging from 0 to $(N-1)$. The pair $(0,\ N)$ denotes the locations of IPs at that lowest level. Each switch, denoted by $S(l,p)$, has four child ports and two parent ports. The IPs are connected to $N = 4$ switches at the first level. In the j^{th} level of the tree, there are $N = 2j + 1$ switches.
The number of switches in the butterfly fat tree architecture converges to a constant independent of the number of levels. If we consider a 4-ary tree, as shown in Figure 2.1(f), with four down links corresponding to child ports and two up links corresponding to parent ports, then the total number of switches in level $j = 1$ is $N/4$. At each subsequent level, the number of required switches reduces by a factor of 2. In this way, the total number of switches approaches $S = N/2$, as N grows arbitrarily large [GPIS04].

Balfour and Dally present a very comprehensive analysis of NoC topologies and

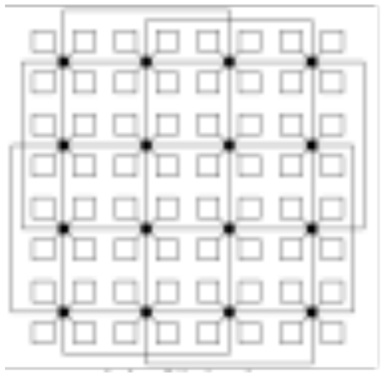


Figure 2.2: Concentrated Mesh NoC

architectures in [BD06a]. They propose the *Concentrated Mesh* (CMesh) reported in Figure 2.2, which is a mesh whose nodes are grouped in sets of 4 and the links on the borders are connected with mode distant set of nodes in a way similar to Torus. The authors discuss also the idea of duplicating certain NoC topologies, such as Mesh and CMesh, to improve the system performance. An extensive analysis of NoC architectures is presented also by Pande *et al.* and Jayasimha *et al.* in [PG$^+$05,JZH].

Literature proposes also many other hybrid solutions where the NoC is built *ad-hoc* rather than on a fixed topology. This NoC require a previous knowledge of the flow of data that the NoC will have to handle. Examples of tools generating ad-hoc networks are XPIPES from University of Bologna and Standford [JBMM04, JMBM04], COSI [Pin08] from Berkeley University and the work of Srinivasan *et al.* [SCK07].

2.1.2 Network Interfaces

As depicted in Figure 2.3 cores are attached to a NoC through a device called Network Interface (NI). NIs hide the lower network implementation details decoupling the communication from communication issues handling the former and leaving the latter to the connected cores.

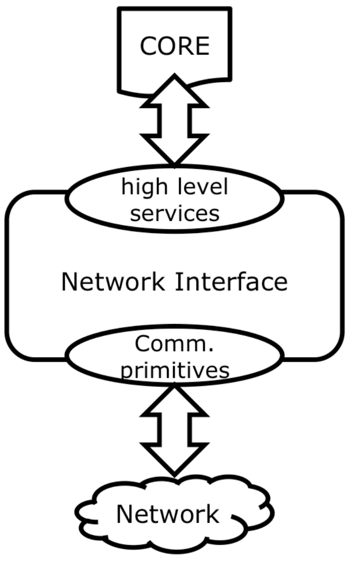


Figure 2.3: Logic representation of a Network Interface

The use of NIs allow *(i)* the reuse of architecture independent designs, *(ii)* the implementation of feature-specic sockets, and *(iii)* the simplication of system verication and testing.

To maximize the reusability NIs usually offer a standardized core interface implementing a set of API defined by industrial standards such as Open Core Protocol (OPC) [OI], Advanced eXtensible Interface (AXI) [AA], CoreConnect [IC], Virtual Component Interface (VCI) [cis], Device Transaction Level (DTL) [PS], etc...

NIs provide a set of communication services such as address decoding and mapping, packetization and de-packetization of the core's messages, packet reordering, request-reply matching and in case of prioritized services, NIs negotiate with the underlying routers for the use of the priority-specific services.

In order to guarantee a correct flow of data, beside the low level flow control, NIs usually implement an end-to-end flow control that assures the availability of buffering space on the NI side for the messages that will enter the network.

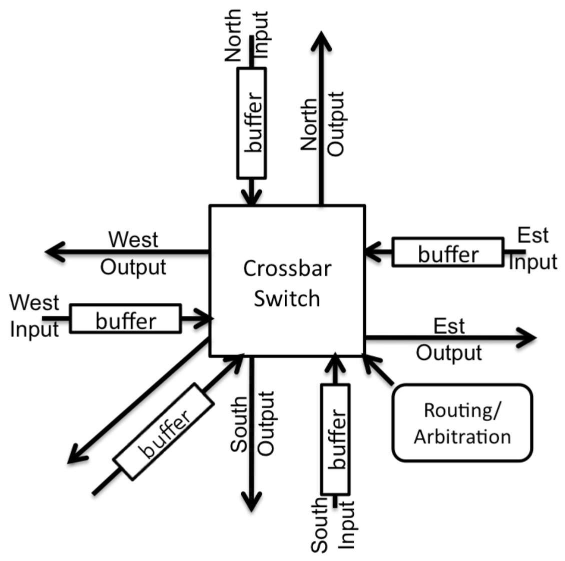


Figure 2.4: Logic representation of a classical input-queued router.

2.1.3 Routers

Routers are the main components of a NoC. In the NoC domain router design and architecture vary depending on the topology, the services and special features supported by the network architecture. Figure 2.4 depicts the classical router scheme [Dal90, PD01]: a router is composed by input buffers one for each port, a crossbar connecting all the input to all the outputs and the control logic that implements the routing and channel arbitration functions.

The main logic function implemented by routers is the routing algorithm. A routing function can be computed either by the source node (source routing) or by the router itself. In the first case the router needs only to read the packet header while in the distributed case each router will compute the packet's next hop by computing a routing function or reading a routing table [DT04, DYN03]. In Section 2.2.1 we cover in detail this important issue.

Routing is a vast research area common in all the distributed system research fields. Together with routing another important and vast research area tries to address the problem of resource *deadlock* that in Section 2.2.2 we discuss in more detail.

Other important research areas that affect the design of the routers are the broadcasting and multicasting of streams of data [GRK$^+$05], traffic priorities and express channels [KPKJ08, OM06b], flow controls [CPC08, SP03a] and adaptiveness [SP03a].

2.1.4 Channels

Channels interconnect the routers in order to form a network. They are characterized by their *width* and *length*. The width of a wire also called *phit* is the amount of data that it can transport in one clock cycle (or the number of data wires the channel is composed of) and it is measured in *bit*. Usually a phit corresponds to a *flit* (flow-control-unit) which is the smallest element of data that routers and NIs can handle.

The channel length is the distance that it has to cover to connect two routers. One main issue of a channel is that the delay taken by a flit to traverse it is a quadratical function of the wire length. A classical solution to this problem consists of segmenting the channel through buffers or couples of inverters so that the delay becomes a linear function.

Still, considering synchronous systems, the shrink of transistor technology and the relative increase of the clock frequency has made the wire delay a very important issue. The channel traversal delays of long wires in fact could hamper the synchronism of the system crating reliability issues of the system. In Chapter 7 we better explain this issue and propose a solution to it.

2.1.5 Virtual Channels

Virtual Channels (VC) are logical abstractions of a physical link. As depicted in Figure 2.5, each input and output channel of a router is provided of multiple input and output buffers that compete to access to the shared channel.

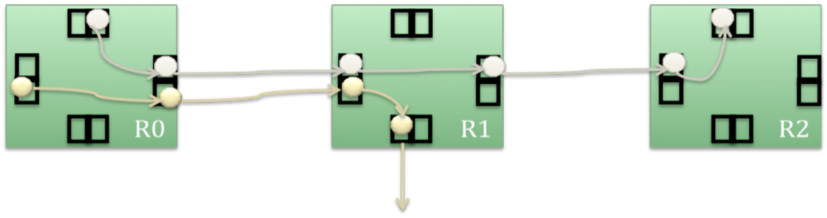


Figure 2.5: Example of performance improvement given by VC: a packet is forwarded even if another packet is blocked on another VC.

When VC are installed on a router, the routing function must be enriched by a *virtual channel selection function* that selects the next VC to use once the output port has been decided. In [LBBP94, LZL00] authors propose a comparison of different VC selection functions. In [DT04] authors discuss about the flit forwarding rule that on each cycle selects the flit to forward among the active VCs. In this Thesis we use the *Winner take all* [DT04] policy as the solution when no priority classes are defined.

Thanks to their flexibility virtual chennels can be used for a number of reasons:

deadlock avoidance VCs abstraction increases the number of options available to a routing function. Since each VC is independent from the others the VC selection function can be used to slect VC in a way that avoids the generation of circular dependencies and hence deadlock;

optimizing wire utilization Letting several logical channels share the physical wires, the wire utilization can be greatly increased. Advantages include reduced leakage power and wire routing congestion;

improving performance VCs can generally be used to increase the performance of a system by minimizing the frequency of stalls. In [AP95] Pinkston *et al.* propose an efficient deadlock recovery adaptive routing algorithm that maximises the performance of the system. A similar approach is proposed by Duato in [DYN03] where the author propose an important theorem stating that a routing algorithm can be considered deadlock free as long as it always provides a deterministic deadlock free *virtual lane* (concatenation of VCs) that a deadlocked packet can use to recover from the stall;

differentiated services VCs can be associated to different QoS services. The NI is in charge of negotiating the VC to use with the underlaying router. Once a packet is directed along a given priority queue it's up to the router to select the VC with the highest priority and forward its flits;

In [VSD97] authors argue that while VC can help on increasing performance, their complex implementation can reduce the router clock frequency reducing the real benefits of this solution. In [Peh01b] Peh and Dally show that a router with up to 4 VCs can run at frequencies similar to those of a classical VC-free whormhole router.

In Chapter 8 we compare the VC system with the Multi Plane approach and notice that VC are an interesting solution only when the queues of the routers are not set to their minimum.

2.2 Protocols Design

2.2.1 Routing

Routing packets along an interconnection network is a well known problem of the parallel computing systems. Literature offers many different algorithms suited for almost any kind of architecture proposed so far [MB06, CGL+08].

Routing algorithms can be subdivided into *source* or *distributed* routing functions. In the first case a source node computes the whole path of a packet through the NoC. In the latter the source node just forwards the packet to the connected router that will compute the first step and leaving to the following routers the burden of computing the following path.

A routing function can be either *minimal* or not. In the first case each hop performed by a packet takes it always closer to its final destination. In the case

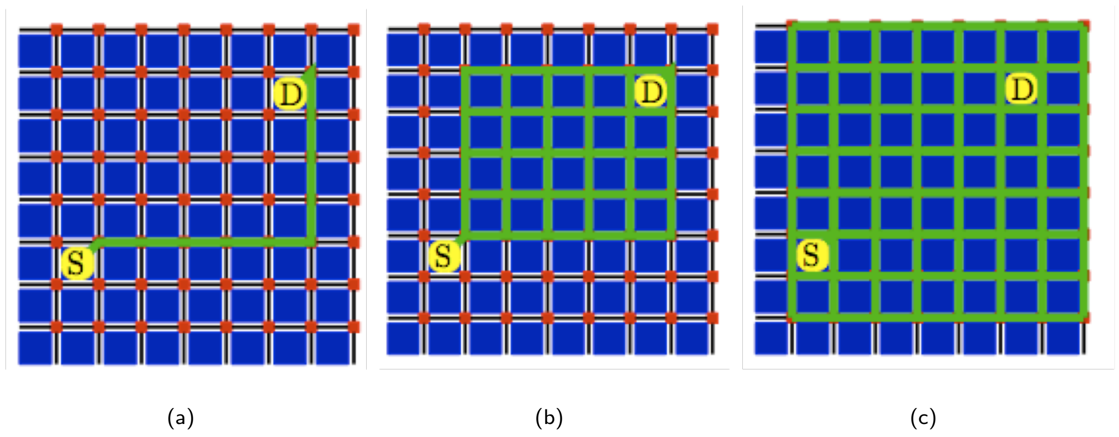


Figure 2.6: Example of (a) minimal deterministic, (b) minimal adaptive, (c) non minimal adaptive routing.

of non minimal routing a packet can be forwarded in any direction. Non minimal routing are useful for developing fault-tolerant algorithms.

Finally routing algorithms can be classified in *deterministic*, *adaptive* and *oblivious* [DT04, DYN03].

In the NoC domain deterministic algorithms are usually preferred because they are simple to implement and their behavior is easily predictable: given a source and a destination node, a deterministic routing algorithm computes (through a look-up table or a mathematical equation) the exact path towards the packets destination node. Figure 2.6 (a) shows an example of minimal deterministic routing algorithm: given a source S and a destination D the routing function returns only one single path connecting them.

Figure 2.6 (b) shows the result of a minimal adaptive routing algorithm. Adaptive routing algorithms compute the path of a given message considering the source and destination nodes address and also some information relative to the status of the network. Because of their nature these kind of algorithms can easily fall in a deadlock state but can greatly improve the performance of the system as they manage to evenly distribute the traffic along all the channels interconnecting two peers.

Figure 2.6 (c)shows the result of a non minimal adaptive routing algorithm.

These algorithms are difficult to predict and the greater degree of freedom can easily degenerate in deadlocks. Non minimal algorithms can be used in complex NoC systems featuring a certain degree of fault tolerance.

2.2.2 Deadlock Handling

Circular dependencies among shared resources like the channels of a Network on Chip is a serious issue that may block the functioning of the system. The adoption of the wormhole [DYN03] flow control scheme empathizes even more this problem as it allows a packet to be stored along multiple channels increasing the probability of packet stalls. In literature there are many algorithms that address this problem. Algorithms are usually classified as deadlock *avoidance* and deadlock *recovery* techniques.

Deadlock avoidance algorithms usually introduce some kind of restriction on the routing algorithm in order to avoid the formation of circular dependencies. A classical example is the *Dimension Order* (also called *XY*) routing algorithm for the Mesh topology. Given a source s and a destination node d a packet is first forwarded along the horizontal direction (X) and only when the packet reaches the column of the destination d it is forwarded along the Y direction towards d.

Deadlock avoidance algorithms are often used in SoC domain as their behavior is simply predictable, they are easy to implement and to test. Moreover they require much less resources than any other kind of algorithm;

Deadlock recovery algorithms allow the unrestricted use of the resources of the system with the purpose of maximizing the systems performance [DT04]. These algorithms rely on some mechanism that detects a possible deadlock to the protocol and a deadlock recovery mechanism that resolves the stalled state. In Literature there are many proposals of deadlock detection techniques. Usually they are based on timers that detect the possible blocking of the system. An important algorithm of this kind is DISHA [AP95] which following the Duato's theorem for the deadlock avoidance of adaptive routing [DYN03] functions. In particular Duato's theorem allows routers to exploit all the resources of the NoC. In case of a deadlock detection the blocked packets are moved on specialized virtual channels in order to be forwarded using a deadlock free routing algorithm. The DISHA algorithm is of special interest because it avoids the dropping of the blocked packets (fact extremely depreciable in the NoC domain) and also offers many improving possibilities which we think are worth to study.

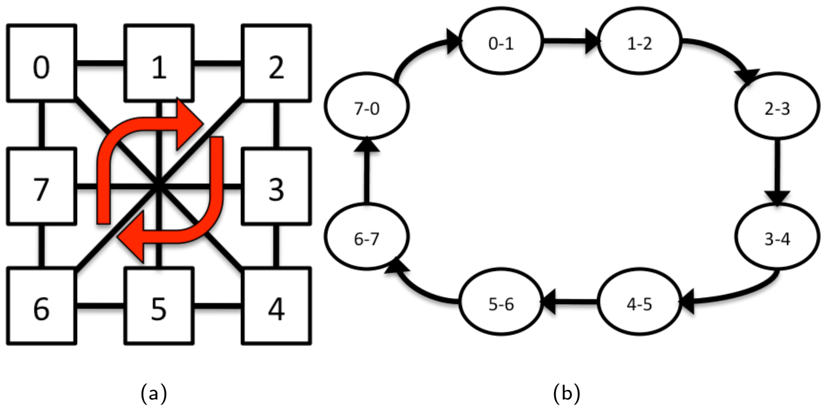


Figure 2.7: A *Spidergon* NoC and the channel dependency graph of its clock-wise ring channels.

A *Channel Dependency Draph* is a technique proposed by Duato [DYN03] that allows the study of the dependencies inside a network and find possible deadlocks. In a CDG nodes represent the channels of the system under analysis while edges represent the dependency between two adjacent virtual channels.

Figure 2.7(b) shows the CDG of the *Spidergon* circular dependency on the clock-wise ring channels highlighted in Figure 2.7(a). A single channel between $node_A$ and $node_B$ is named $ch_{A,B}$ and in case of multiple VCs we distinguish between the channels using the indexes *i.e.*: $ch_{(A,B)0}$ and $ch_{(A,B)1}$. In particular considering $node_0$ and $node_1$ of Figure 2.7(a) the arc between $ch_{0,1}$ and $ch_{1,2}$ in the CDG of Figure 2.7(b) indicates that according to the aFirst routing algorithm implemented by the routers, a packet traveling on $ch_{0,1}$ can be forwarded by the $node_1$ towards $node_2$ through the channel $ch_{1,2}$. As each node $node_i$ can forward a packet along the ring channels towards the $node_{i+1}$, *Spidergon* shows a *circular dependency* on its ring channels that must be solved to prevent deadlock.

2.2.3 Mapping

The problem of mapping Processing Elements and Storage Elements and in general any resource on a generic Network on Chip is a complex and delicate problem. In [LRD01, HM03] authors demonstrate that the performance of a system may vary up to the 250%.

The mapping problem is usually seen a the special case of graph embedding problem between a source graph G_S and a target graph G_T. An application is seen as a *Task Graph* [PR96, BCG+07]: a weighted graph that allows representing a parallel application as a directed graph. In a task graph, nodes represent the independent entities that characterize the parallel application, while edges between two nodes represent the communication between these two entities. Edges can have a weight representing the bandwidth of the communication. A target graph represents the NoC architecture; a node in the G_T represents a node in the NoC while an edge represents a channel on the network.

Literature offers many algorithms and tools to solve this problem. In particular the *Xpipe* framework [JMBM04] of the DEIS department of the Bologna University and the SCOTCH tool [PR96] of the University of Bordeaux represent two powerful systems to address this problem. Other algorithms considering also some quality of service constraints have been proposed by Pinto [Pin08] and Murali [MM04b].

The choice of the algorithm to choose or the tool to adopt depends on many factors such as the architecture we want to use or the information we have about the application to map (i.e. throughput and latency constrains). In the optic of our research activity we chose to adopt the SCOTCH [PR96, BCG+07] partitioning tool. Chapter 4 discusses the mapping issues in detail and here we show our analysis results .

Part II

Network Topologies

Chapter 3

The *Spidergon* NoC

This Section describes the regular and point-to-point topology of the *Spidergon* NoC [CGL+08, CLM+04, BCG+07, MRG+04]. In contrast to multistage networks, point-to-point ones connect each computing resource to a router element. Regular topologies, i.e. symmetric networks with vertex- and edge-transitivity, offer the advantage that all nodes have a global knowledge of the network, allowing for simple routing and scheduling decisions. Thus, the router hardware implementation is simple and routing decisions are fast. Another key advantage of a regular topology is its simple and efficient VLSI layout using existing design methodology that exploits uniformity and provides predictability in individual wire lengths and path delays.

The *Spidergon* network shown in Figure 3.1 connects a generic even number of nodes $N = 2n$, with $n = 1, 2, 3...$ as a bi-directional ring in both clockwise,

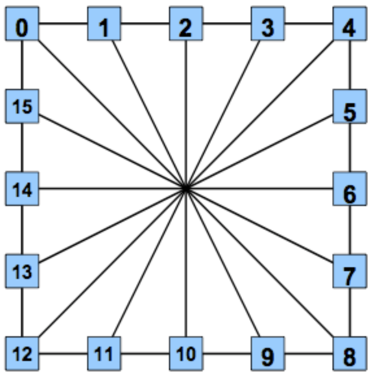


Figure 3.1: The *Spidergon* Network on Chip

and anti-clockwise directions with in addition a cross connection for each couple of nodes. By a formal definition, each $node_i$, with $0 \leq i < N$, is connected by three unidirectional links to other three nodes with the following rules:

- clockwise connection, i.e. $node_i$ to node $node_{(i+1)\ mod\ N}$;
- anti-clockwise connection, i.e. $node_i$ to $node_{(i-1)\ mod\ N}$;
- cross connection, i.e. $node_i$ to $node_{(i+n)\ mod\ N}$.

3.1 Basic *Spidergon* Routing Algorithms

The routing strategy determines the route from source to destination. Besides minimizing the number of steps, the routing scheme, especially in the NoC domain, must be simple to implement to allow an efficient router design, both in terms of reduced area cost and higher working frequency.

The *Spidergon* NoC routing adopts the wormhole switching technique. The switching strategy refers to how a packet traverses the route. While with virtual cut-through and store- forward switching strategy, flow control is per packet (router must allocate buffering resources for an entire packet), with wormhole routing a packet is further subdivided into *flits* (flow control units), with each flit having its unique flow control. Thus, the rest of the packet flits must follow the same path reserved for the header. Wormhole routing was indicated as the good choice for the NoC domain, even if it is more susceptible to deadlock than virtual cut through under heavy network load: this is because wormhole drastically reduces the amount of network buffering (queues with the flit granularity instead of packet) and allows for a deep pipelined packet communication.

Within the *Spidergon* NoC framework it has been proposed a deterministic, shortest-path network routing algorithm called *Across First* (aFirst)NoC [CGL$^+$08]: the relevant implementation on the on-chip routers is expected to be simple without posing the need for expensive routing tables.

The choice of a deterministic routing avoids costly flit reordering at packet reception: the routing path of any packet does not depend on the route of any other packet, i.e. the communication path is completely determined (a priori) by the source and destination addresses. The packet routing decisions are carried out using only local information available at each network node; in addition, the routing algorithm is local, *i.e.* identical (or symmetric) for all router nodes, as described below.

Each router has a unique address r_i in the network, $0 \leq i < N$, where N is the network size. Since the routing algorithm is local, and the NoC topology is vertex- and edge-transitive, we may describe the routing algorithm at any node. When a router receives a first flit (the header) of a new packet, the forwarding path is calculated. The algorithm compares the address of the current router (*curr*) to the address of the target router (*dest*) stored in the header flit, i.e. the router connected to the desired resource. If the two addresses match, then flits are routed to the local output port of the router, i.e. to the desired resource through its network interface. Otherwise, a clockwise, anti-clockwise, or cross output communication port is selected depending on the relative distance of the two nodes, as follows.

```
if (dest = curr)
  route packet to the NI direction;
  //(packet is consumed by the local computing resource)
else
if  dest  curr  mod N = {1, 2, 3 . d}
    route packet to clockwise (Right-R) direction;
if  dest  curr  mod N = {N-1, N-2, N-3 . N-d}
    route packet to anticlockwise (Left-L) direction;
if  dest  curr  mod N = {d+1, d+2, . N-d-1}
    route packet to cross (Across-A) direction;
```

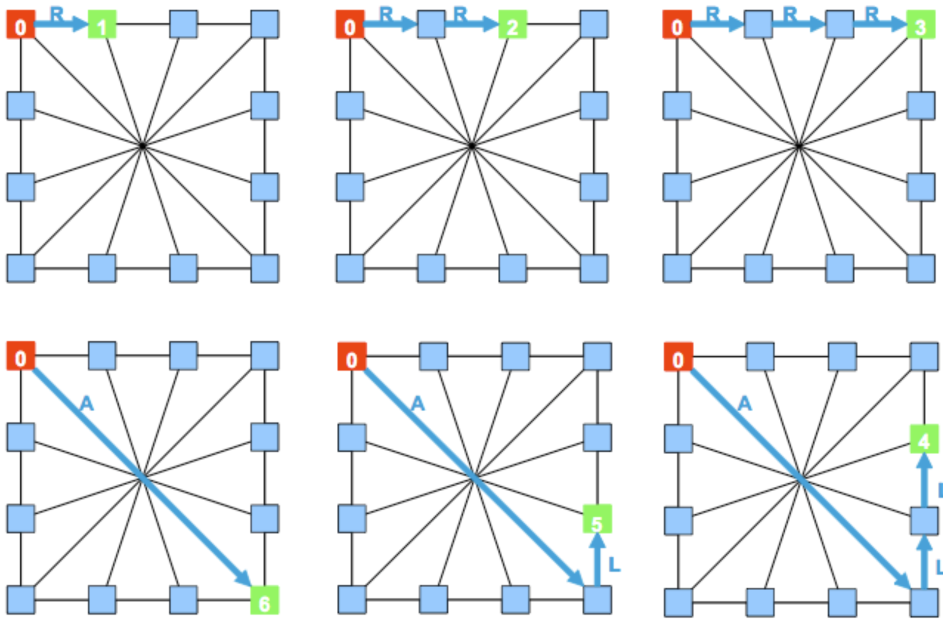


Figure 3.2: The *Spidergon* Network on Chip

Figure 3.2 shows the routing paths in a *Spidergon* NoC with $N = 12$, considering the $node_0$ as starting point; the clockwise direction is indicated with R (right), the anti-clockwise with L (left) and the cross connection with A (across). Due to the symmetry of the topology, the routing scheme is showed only for half of the all possible destination nodes. The idea is to move along the ring, in the proper direction, to reach nodes which are close of the source node (not far from d hops), otherwise to use the cross link to be in part of the network that is far away.

Notice that the described routing algorithm of the *Spidergon* NoC implies that the cross communication port is selected at most once, and always in the beginning of each packets route. Moreover it can be observed that when the packet starts to move in the ring, it follows the same direction (right or left) for the entire path.

The previous two properties can be exploited to optimize the routing scheme, further reducing the relative implementation: there will be two steps, one at packet injection, requiring the same complexity of the presented first basic scheme, and the other during the packet path, being extremely simple and fast.

According to the first description of the routing scheme the packet header has to carry the identifier of the destination node (dest), which amounts to $log_2(N)$ bits (configuration with N nodes).

3.2 *Spidergon* Analysis

In this section we present and analyze the state of the art algorithms and protocols that have been proposed for the *Spidergon* NoC and that we use throughout all the Thesis.

3.2.1 Virtual Channel Selection Algorithms

To the best of out knowledge the *Spidergon* NoC can support the following three algorithms for the selection of the virtual channels. These algorithms are independent from the chosen routing algorithm and are relative to the VC installed on the ring connection of the NoC. The across connection in fact does not have virtual channels as the routing algorithm does not need them.

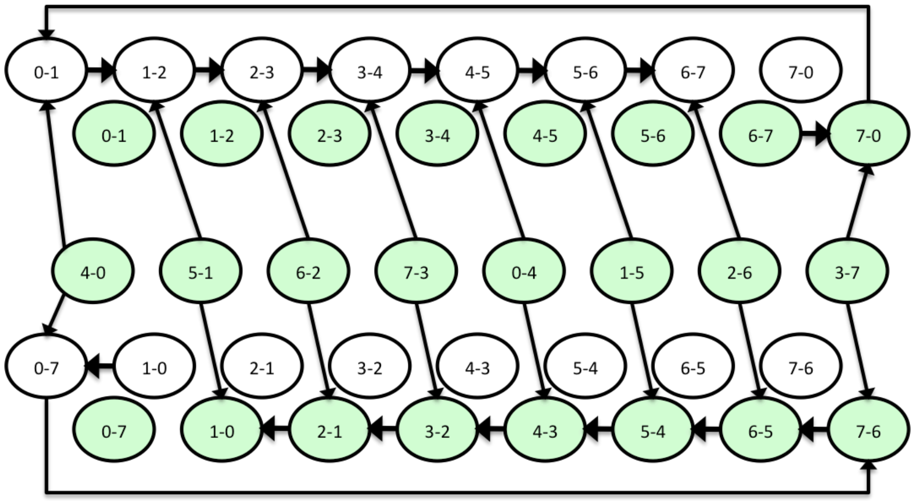
Single Deadline

Single Deadline algorithm defines a particular node of the network as deadline. The choice of the VC to use depends on whether a packet has to pass through this predefined deadline-node or not. Given a Source and a Destination node the VC is selected through the function:

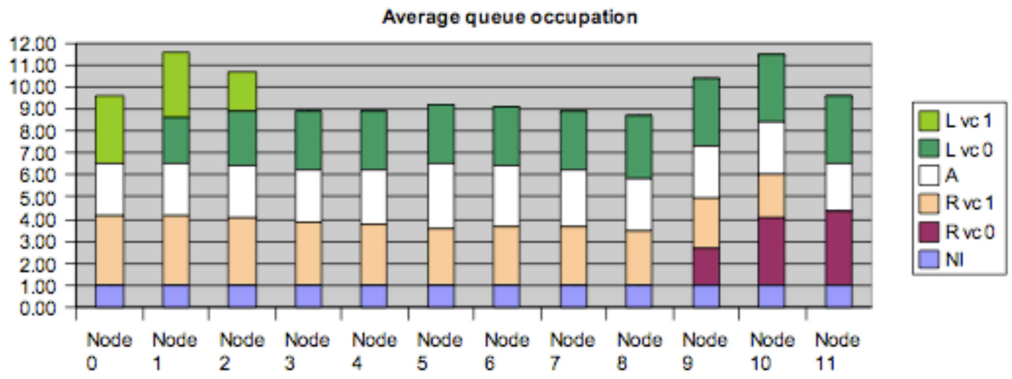
```
if ( Current_Node_ID > Destination_ID )
use VC 0
else
use VC 1
```

Figure 3.3(a) shows the channel dependency graph[1] for this VC selection algorithm where bright nodes represent VC_0 and dark nodes represent VC_1 . The

[1]see Section 2.2.2



(a)



(b)

Figure 3.3: Channel dependency graph (a) and average queue occupation (b) for Single Deadline algorithm.

graph is relative to a 8-node *Spidergon* NoC adopting the aFirst routing algorithm described in the previous Section.

As it is clear from the CGD of Figure 3.3(a), virtual channels add more nodes in the channels dependency graphs. When provided of VCs the routing algorithm decides the direction of each dependency-edge while the VC selection algorithm selects which VC an arrow has to reach.

Figure 3.3(b) depicts the average queue length of each VC in the considered eight-node *Spidergon* with uniform traffic pattern scenario. Each rectangle in the bar-diagram indicates the usage of a specific queue among the two VCs installed on the right (R) and left (L) and the single queue installed on the across (A) channels. From Figure 3.3(b) it is clear that the Single Deadline algorithm is a very simple solution that poorly exploits the given resources. In particular it is clear that packets travel always along one single virtual channel and switch to the other only when they reach the deadline $node_0$. Although very simple to implement, this algorithm actually wastes most of the resource installed of the network.

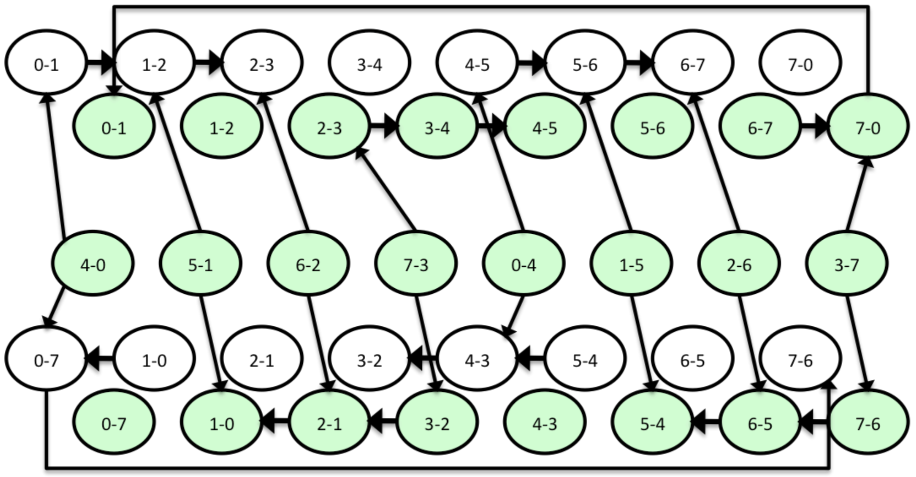
Multiple Deadline

Multiple Deadline algorithm is an enhancement of the Single Deadline algorithm seen before. Multiple Deadline uses N predefined deadlines nodes. The reason to use more deadlines is to achieve a better utilization of the resources of the system. The function implemented by the algorithm is the following:

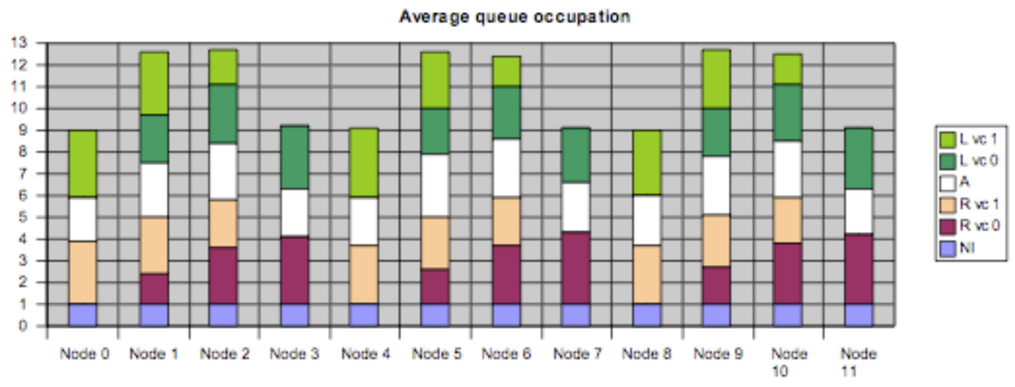
```
If (Current_Q > Destination_Q)
use VC 0
else
use VC 1
```

Where ND is the Network Diameter and $Current_Q$ and $Destinatin_Q$ are defined as:

$$Current_Q \;=\; Current_{id} \; mod \; ND$$



(a)



(b)

Figure 3.4: Channel dependency graph (a) and average queue occupation (b) for Multiple Deadline algorithm.

$$Destination_Q \quad = \quad Destination_{id} \; mod \; ND$$

Figure 3.4(a) shows the channel dependency graph for the Multiple Deadline VC selection algorithm. The graph is relative to a 8-node *Spidergon* NoC adopting the aFirst routing algorithm described previously with four deadlines on nodes $node_0$, $node_2$, $node_4$ and $node_6$.

Figure 3.4(a) shows that by defining more deadlines nodes the dependencies among the channels are more variegate. As it is clear from the average queue utilization diagram of Figure 3.4(b) improves the use of the different VC. As in the case of Single Deadline also this algorithm does not exploit all the VC of the system.

Distance Based Deadline

Distance Based Deadline selects a VC on the *Distance* (D) between the current node and the packet destination. It tries to use the networks VCs like a highway exit lane:

```
if (Distance > 1 )
use VC 0
else
use VC 1
```

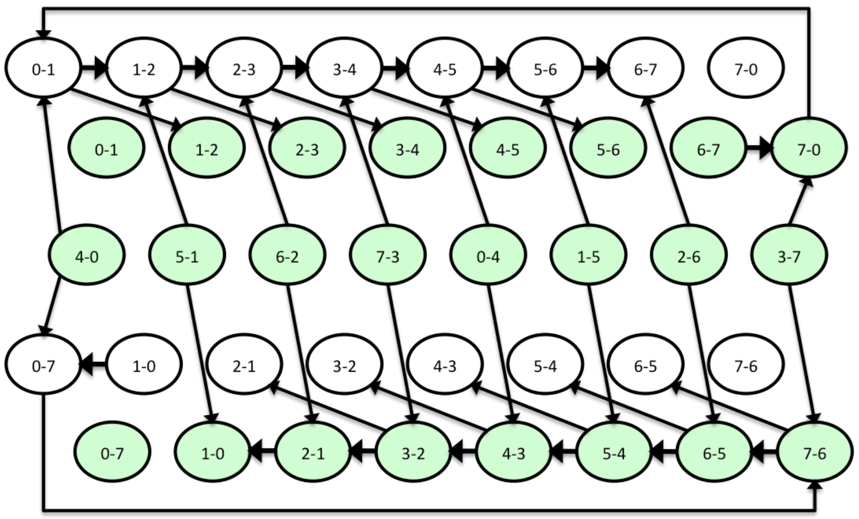
Figure 3.5(a) shows the channel dependency graph for the Distance Based Deadline VC selection algorithm. As for the previous algorithms, the graphs is relative to a 8-node *Spidergon* NoC adopting the aFirst routing.

Figure 3.5(a) shows that by following this different approach and considering the packet distance to the destination rather than fixed deadlines, under a uniform traffic pattern the dependencies among the channels are more variegate. As it is clear from the average queue utilization diagram of Figure 3.5(b), this algorithm greatly improves the usage of the storage installed on the channels exploiting all the virtual channels of the system.

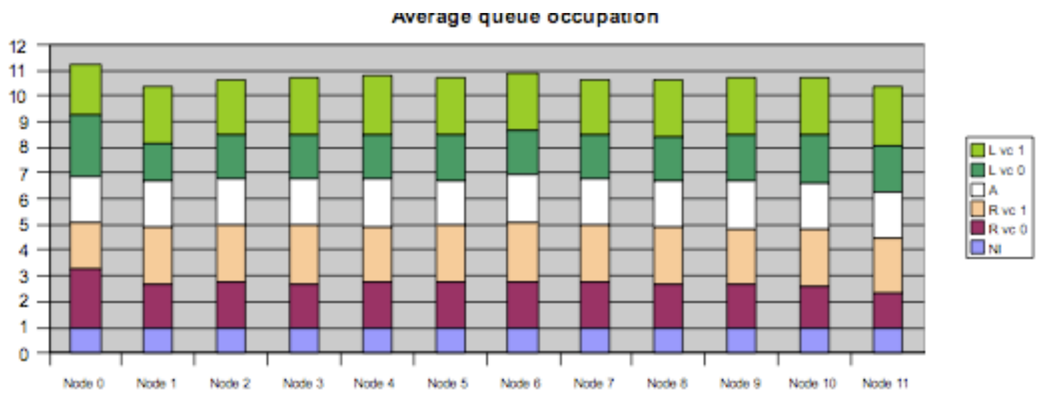
Virtual Channel Selection Algorithms Comparison

Figure 3.6 compares the three presented algorithm showing the average packet latency of ton a *Spidergon* network with uniform traffic pattern. From the chart it is clear that the distance-based algorithm outperforms the others. As we already discussed in fact this algorithm is capable of explaining in a even way all the resources installed on the network on chip.

From now on unless differently stated all the simulation results will be using the distance-based VC selection algorithm.

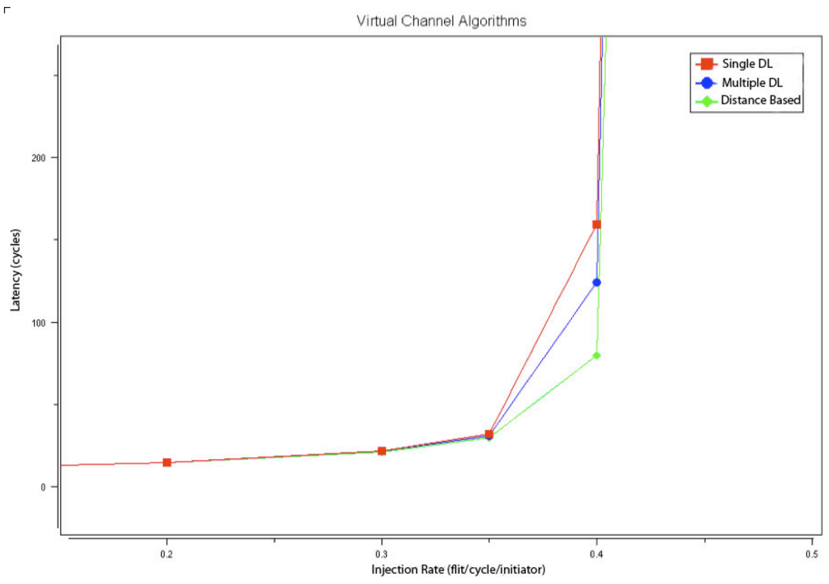


(a)



(b)

Figure 3.5: Channel dependency graph (a) and average queue occupation (b) for Distance Based Deadline algorithm.



(a)

Figure 3.6: Average packet latency comparison for single and multiple deadline and distance based VC selection algorithm

3.2.2 Routing Algorithms

Routing algorithms are one of the main research area we are interested in. They heavily influence the performance of the system, the resource requirements and finally costs.

In the NoC domain routing algorithms should be simple but at the same time fast, efficient and reliable. In our research and analysis activity we studied a number of solutions that will be presented in the following pages.

aLast routing algorithm

Across Last, (aLast) is a deterministic source routing and minimal algorithm derived form the aFirst algorithm described in Section 3.1. As the name suggests the algorithm allows a packet to use the across channels only as last path step in the network (Across Last = aLast).

As depicted in Figure 3.7(a) ALast forwards packets along the ring channels

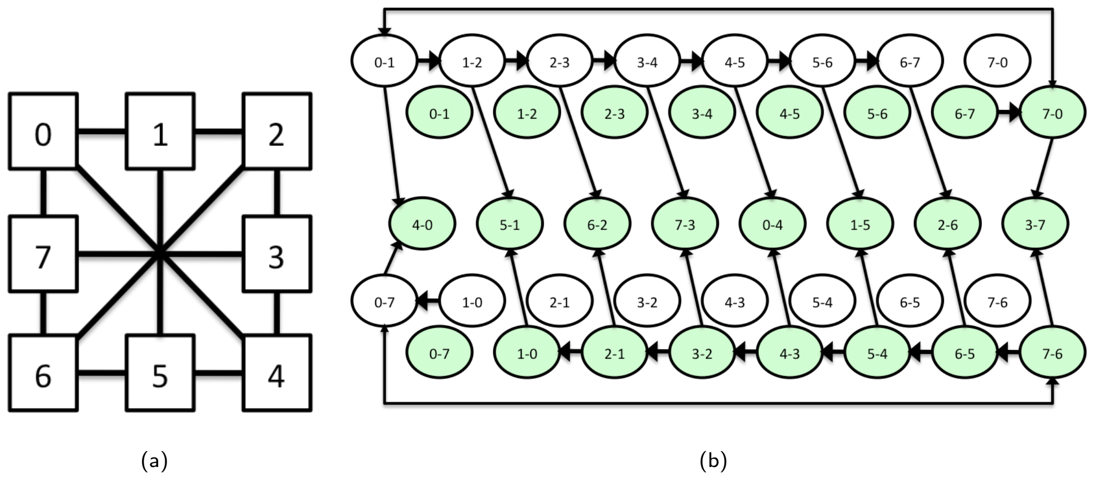


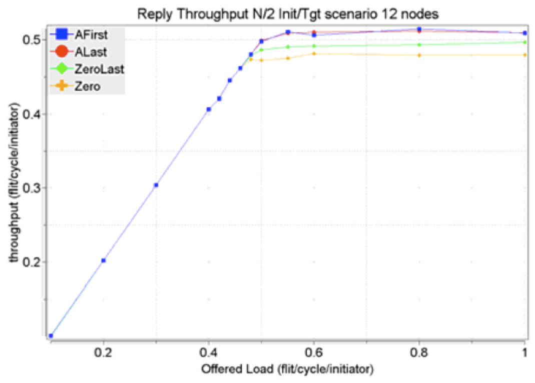
Figure 3.7: Eight nodes *Spidergon* (a) and relative channel dependency graph (b) of the aLast routing algorithm with the single deadline VC selection algorithm.

towards the node in front of the final Target destination, only then it performs the Across step to the packet destination. As aFirst requires two virtual channels on the ring links to break the dependency formed by the packets on the peripherals links. In Figure 3.7(b) is depicted the channel dependency graph for this algorithm using the single deadline VC selection algorithm in a network with 8 nodes.

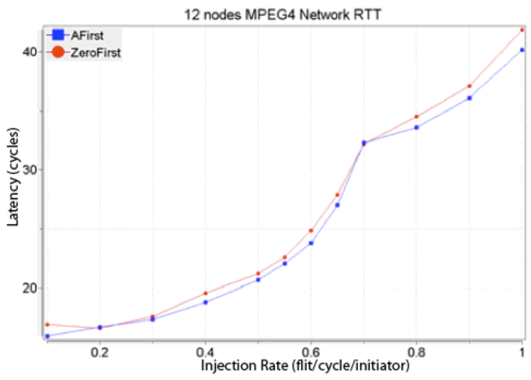
Restricted path routing algorithms

ZeroFirst (ZF) and *ZeroLast* (ZL) routing algorithms address the deadlock problem in a different than the algorithms seen so far. ZF and ZL in fact do not use any virtual channel on the ring connections. Instead they break the circular dependency on the peripheral links by not allowing packets to pass through a specific node (the node zero). This node can still send and receive packets, and in the case of ZeroFirst it can still forward packets coming from the Across channel. The only restriction is that it can not be an intermediate node in a packet path along the ring channels. Packets that normally should pass through the node zero are forwarded by the source router to a *non minimal* path along a direction that depends on the used algorithm:

- ZeroFirst, if the packet node will pass the node zero: o after performing an



(a)



(b)

Figure 3.8: Throughput on a *Spidergon* NoC comparing for different routing algorithms (a) under uniform traffic pattern and (b) under non uniform traffic pattern.

across step: it is forwarded on the ring channel in the direction of the packet's destination o to reach its destination without passing through the across link: it is forwarded along the across channel in order to reach the closest node on the destination side of the ring.

- ZeroLast if the packet node will pass through the node zero: o to reach its destination: it is forwarded along the ring in the opposite direction with respect to the packet destination. In this way the packet will reach a node facing its destination and will perform an across step. o to reach the node in front of its destination: the packet will be forwarded along the ring channel in the direction of its destination.

ZeroFirst and ZeroLast then are source routing [DYN03] algorithms that have also to require some logic indicating if a packet will or will not pass not through the node zero during its path toward its destination.

If the path does not include the node zero ZF and ZL algorithms act like normal distributed routing algorithms while if the does the packet is forwarded along a different path. The following routers will treat the redirected packet as a normal one, no further actions will be required.

As reported in Figure 3.8(a) ZL and ZF compared to AF and AL perform poorly in the case of uniform traffic patterns with many SEs. The reason is mainly due to the packet redirection caused by the restriction imposed by the two algorithms. ZF and ZL in fact can force packet to follow non minimal paths. This will increase the traffic on the network, augmenting the channel contentions and hence decreasing the overall system performance.

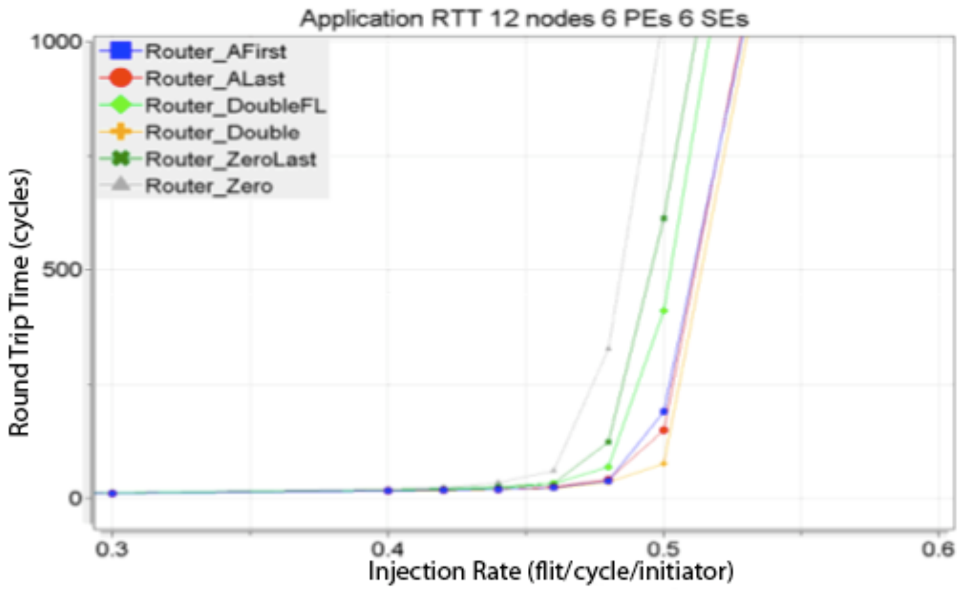
ZeroFirst and ZeroLast behave well in cases where packets are not forced to pass through the zero node because the application does not require it. This happens often in real scenarios where few SE elements collect the traffic of more PEs. In Figure 3.8(b) for example is depicted the ZF and AF performance on an Mpeg4 decoder scenario (see Figure 4.6 on Section 4.5). The two algorithms behave very similarly. AF is slightly better thanks to its virtual channels that allow a better traffic flow. VCs benefits anyway are much more visible on bigger networks where many and different packet flows follow the same path to reach different destinations. For small application such as Mpeg4 the Zero algorithms permit to reduce the Spidergon buffer requirements and hence the size and the cost of the NoC.

The Double algorithm

The Double algorithm adopts the previously defined aFirst and aLast algorithms depending on the kind of the packet to be transmitted. More precisely we call *DoubleFL* (First-Last) the algorithm adopting the aFirst algorithm for the request packets and aLast for the replies. *DoubleLF* (Last First) instead is the dual case .

The use of different algorithms for different kinds of packets might be useful to reduce channel contention on a given application mapping. Thanks to the different virtual networks used by requests and replies there is not the need to use also two virtual channels on the across link. Using aFirst and aLast together for a single packet kind in fact would provoke the across channels to generate new circular dependencies in the algorithm channel dependency graph and then deadlock.

As depicted in Figure 3.9 in a scenario with a given mapping, an uniform traffic and the same number of PEs and SE the two Double algorithms may be helpful



(a)

Figure 3.9: Round trip time on a *Spidergon* NoC with 8 Initiator and 4 Targets.

to improve the system performance. In this particular case for example DoubleFL performs much better than the other algorithms. The choice of the right algorithm to use anyway has to be done very carefully because as shown in the Figure 3.9 DoubleFL algorithm instead performs worse than the other minimal algorithm (not considering ZeroFirst and ZeroLast).

3.3 The aEqualized Routing Algorithm

We propose the *Across Equalized* (aEqualized) algorithm: a routing algorithm that optimally exploits the resources of the system reducing the area and power demands of the NoC.

Our algorithm is thought for networks where a small number of nodes attract most of the network traffic behaving as hotspots. In the multi core SoC domain this requirement is a common situation: a number of Processing Elements (PEs) are often connected to a small number of Storage Elements (SEs) forming a tree of connections with the SEs as roots [HG07, KKS05a].

3.3.1 Routing Algorithms Characterization

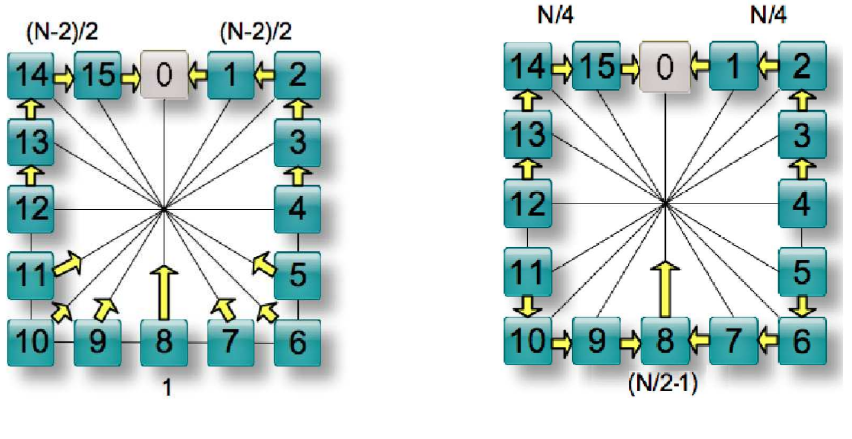


Figure 3.10: Routing behavior towards node zero using aFirst (a) and aLast (b)

The routing algorithms described so far are simple to implement and can actually solve the routing issue. A major drawback of these two algorithms is that the across channels are under-utilized as they can be used only once and only as first or last step.

Figure 3.10 considers a single SE towards which each PE communicates. In the aFirst (a) case, PE's packets travel essentially through the ring channels so that SE's ring channels must support the traffic generated by $\frac{(N-2)}{2}$ PEs. The SE's across input channel instead is used only by the node set in front of the target (node 8 of Figure 3.10). On the other hand, the reply packets traveling from SE to the PE nodes on the opposite side of the ring can only use the hotspot's across channel to reach their destination.

In the case of the aLast algorithm (Figure 3.10(b)) the SE's across channel is over-loaded as it receives the packets generated by the $\frac{(N-1)}{2}$ nodes on the opposite half of the NoC. Channels on the ring instead receive the traffic of only $\frac{(N)}{4}$ nodes each. SE-generated packets instead pass through the ring channels and they may use the across link as last step to get to their final destination.

3.3.2 aEqualized Algorithm

The aEqualized algorithm requires the traffic pattern of the NoC to be known at design time and uses these knowledge to re-redistribute the load of the NoC channels so that all resources are better utilized.

This algorithm combines the aFirst and aLast algorithms seen in Section 3.1. More specifically an aEqualized network is composed by routers implementing either aFirst (tagged as "aFirst") or aLast (tagged as "aLast") routing algorithm.

The choice of the tag to use depends on the adopted traffic pattern and the position of the hotspot nodes on the network. Restricting the use of only one of the two algorithms per node allows us to avoid additional channel dependencies caused by the across channels and hence new potential deadlock situations.

To explain the tagging operation we start from the simple example shown in Figure 3.11, that assumes a single hotspot placed on node 0. In Figure 3.11 nodes in green (bright) are tagged as aLast, nodes in magenta (dark) are tagged aFirst and those in white are the hotspots. The node's tagging operation is made by the following steps:

- initially set all nodes as "aLast" so that the hotspot node receives data originated from:
  - $(N/2) - 1$ nodes on the across channel;
  - $(N/4)$ nodes on each ring channel;
- define $M = (N - 1)$ and $R = (M \ mod \ 3)$
- in function of the remainder R tag the ϕ nodes on the clockwise and counter clockwise side of the hotspot node as aFirst and the remaining λ nodes as aLast in such a way that:
  - $R = 0$: $\phi = M/3$ and $\lambda = M/3$;
  - $R = 1$: $\phi = M/3$ and $\lambda = M/3 + 1$;
  - $R = 2$: $\phi = M/3 + 1$ and $\lambda = M/3$;

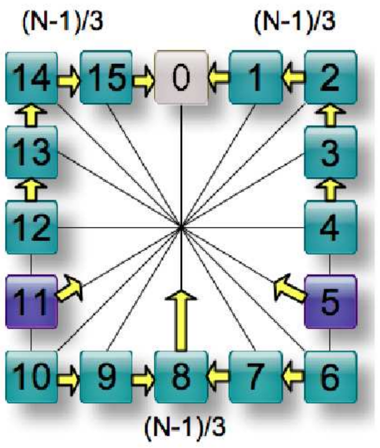


Figure 3.11: The Routing aEqualized data-flow considering a single hot spot. Nodes in green are tagged as aLast and those in magenta are tagged as aFirst

- tag all the Storage Elements as “aFirst” (the reason will be clear later)
- note that $2 * \phi + \lambda + 1 = N$

For sake of clarity in Figure 3.11 and in the following, only $\lambda - \frac{N}{4}$ nodes are actually tagged as aFirst and hence colored in magenta. These nodes are those whose behavior is actually changed by the aEqualized algorithm. In fact because of their distance from the hotspot, the other nodes will communicate with the SE through the Clockwise or Counter Clockwise link independently from the implemented algorithm.

By tagging the nodes in the proposed way we define three sets of nodes each one communicating with the hotspot node through a different SE's input channel. The three sets have a size that differs at maximum of one node depending on the network size N.

Figure 3.12 depicts a more complex example where more SEs are inserted. It's clear that by considering a uniform traffic directed to the SE nodes, when the network size is not multiple of three the traffic equalization will result less balanced.

As a consequence of the tagging restrictions imposed by the aEqualized algorithm, request and reply packets exchanged between any PE - SE pair follow the

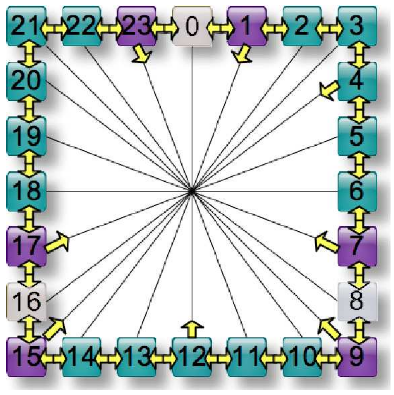


Figure 3.12: The Routing aEqualized data-flow considering three hotspots

same path but in a opposite direction. This characteristic allows to better exploit the channels of the network and also allows to reduce the number of across link actually used bye the packets traversing the *Spidergon* NoC.

A main drawback for this algorithm is that it imposes some restriction on the mapping of the devices on the network: aFirst nodes should not be used as last step to reach a destination through the across channel as the router won't have the right to use the requested link. Hence a storage element should not be placed on a node connected to an aFirst-tagged router by the across link. Figure 3.13 shows an example: two SE-hotspot nodes (nodes 0 and 8) are placed on the same across channel and each hotspot is tagged as "aFirst".

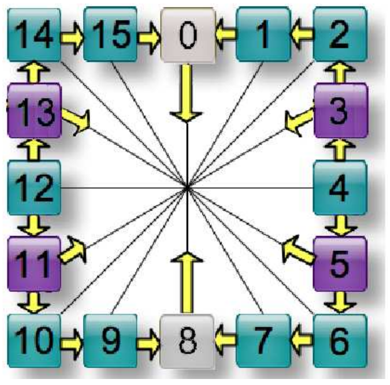


Figure 3.13: A drawback of the aEqualized routing algorithm: hotspot should not be connected to aFirst-tagged nodes

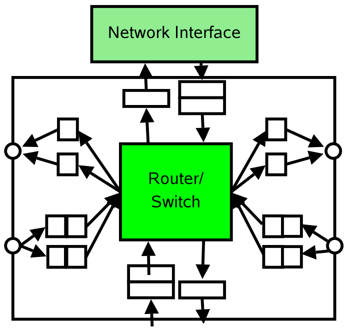


Figure 3.14: Standard node router with 2 virtual channels on the CW and CCW (left and right) links and one on the Across and NI ones (up and down)

Let's consider a communication between node 10 and node 0. Node 10 is tagged as “aLast” so it would forward its packets in the counter clockwise direction towards node 8 that is supposed to forward them to node 0. Node 8 though is tagged as “aFirst” so it does not have the right to use the across channel as last step. Instead it would forward the packets also in the counter clockwise direction towards node 7 breaking, in this way, the minimal path requirement.

3.3.3 System Level Analysis

The *Spidergon* architecture has been modeled and configured with synchronous interconnected routers. Each router was provided with two virtual channels on the clockwise and counter clockwise channels and was linked to a single external IP through a network interface layer as depicted in Figure 3.14.

Depending on the simulated scenario, each IP acted either as a processing element (PE) or as storage element (SE). PEs (also called initiators) generate request packets directed to the SEs selected in a random way with a uniform probability distribution. SE nodes receive the requests packets from the PE and send back response packet modeling a *shared memory* system. All PEs/SEs are provided with infinite FIFO output queues that temporarily store the produced packets once the underlying NoC is not able to absorb them at a sufficient rate.

We compared the considered scenarios by measuring the packet latency metric

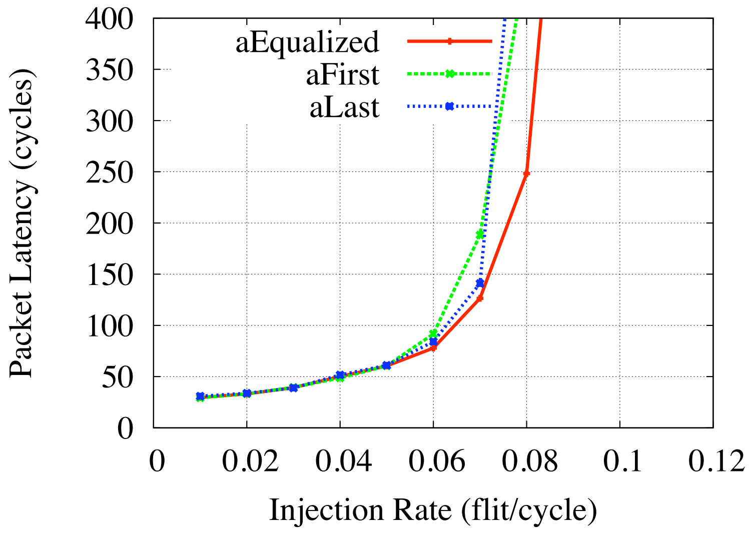


Figure 3.15: Average packet latency of a 24-nodes *Spidergon* with three hotspot Storage Elements

that is the average time (in clock cycles) taken by a packet to enter the network, traverse it and reach its final destination.

Graphs in Figure 3.15 show the average packet latency on a *Spidergon* NoC with 24 nodes with three hotspots addressed randomly by the PEs. The three algorithms perform in a similar way and only aLast performs slightly worse than the others.

The benefits given by the traffic equalization on the hotspots' channels are hidden by the bottleneck present at the hotspots' Network Interface channel. Here packets are blocked by the great amount of data contending the same shared resource: the channel towards the NI device.

The main improvement introduced by aEqualized algorithm is not on the performance side but rather on the resource requirements side. We consider a physical link as required if there are packets passing through it. In SoC environment traffic patterns are usually fixed and well known so that in the final physical design the not used links and router ports can be not implemented saving costs, area and power consumption.

Considering all the connections as bidirectional we noted that in all experiments all the ring channels have been used either to send request or reply packets. The

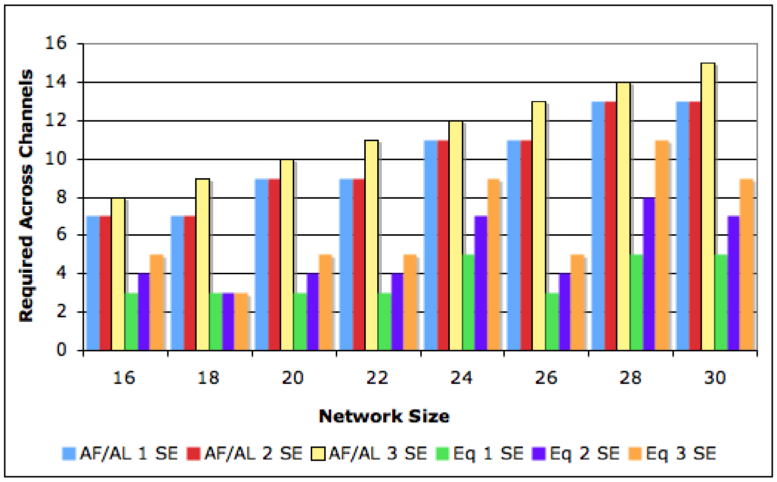


Figure 3.16: Number of required Across bidirectional links on a *Spidergon* NoC with respect to the size of the network

number of required Across channels instead depends on the selected routing algorithm and on the number of SEs in the network.

Figure 3.16 reports the number of bidirectional across links actually used by the packets with respect to the size of the network and the number of hotspots. By granting essentially the same performance as aFirst and aLast, aEqualized algorithm uses up to 60% less across channels than the other two algorithms (network with 26 nodes). This improvement is due to the fact that request and reply packets pass through the same path in the opposite direction. AFirst and aLast algorithms instead essentially use all the across channels of the network either to forward request (aFirst) or reply (aLast) packets. By adopting a uniform traffic pattern, considering also the ring channels of the NoC, the aEqualized algorithm allows a reduction of the required bidirectional connections by up to 13% by maintaining similar performances.

A veritable bottleneck in the *Spidergon* architecture is the single link connecting the router to the node's Network Interface. The hotspot node is the target collecting a large amount of traffic and its NI channel is the single connection handling all the incoming data. Figure 3.17 shows a variation on the architecture of the *Spidergon* node that we investigate to improve the performance of the network. Here a

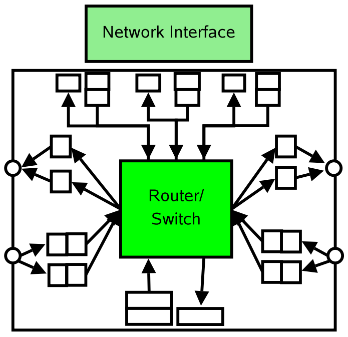


Figure 3.17: *Spidergon* router with three links towards to the Network Interface

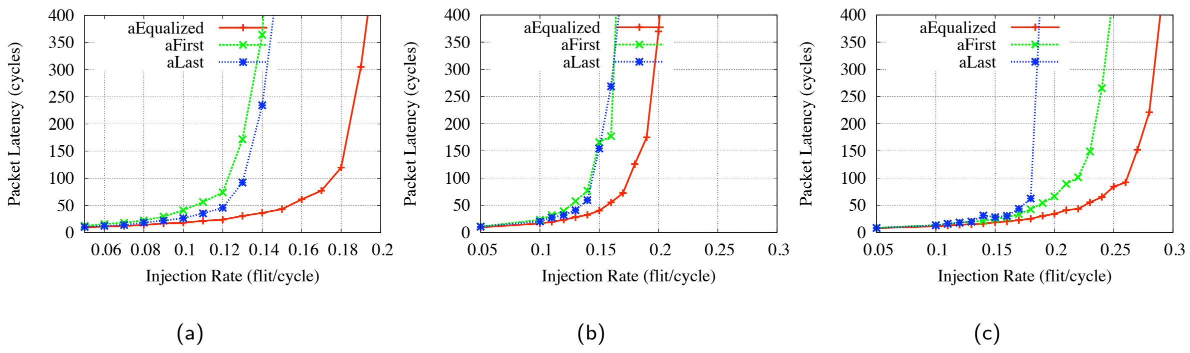


Figure 3.18: Average packet latency of a 24-nodes *Spidergon* with one (a), two (b) and three (c) hotspots using three independent NI channels

node is provided with three independent physical links each one directed towards the Node's NI. Each NI channel is dedicated to a single and pre-defined input channel in order to remove the contentions towards the Network Interface. Packets generated by the NI and directed to the underlying router instead are placed in one of the three randomly selected NI output queues.

Graphs in Figure 3.18 show average packet latency measured with the proposed router improvement on a 24-nodes *Spidergon* with one, two and three targets and a random uniform traffic pattern.

Once removed the NI bottleneck, aEqualized clearly outperforms the other two routing algorithms while also reducing the required physical links. The worst performing algorithm is aFirst as only the nodes facing the hotspots can use the across channel and the NI links dedicated to it. The rest of the traffic reaches the hotspots

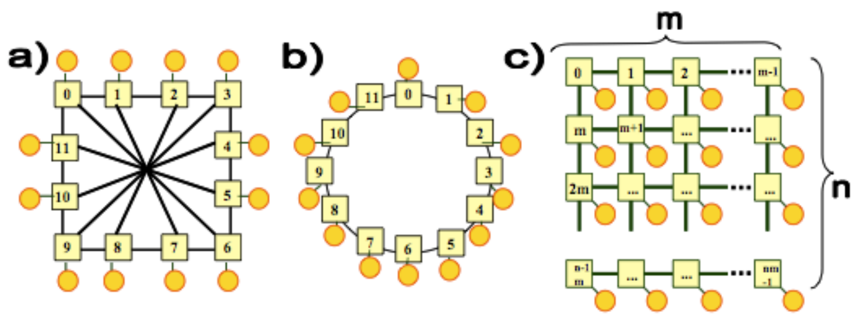


Figure 3.19: NoC topologies: IPs connected to numbered nodes on (a) Spidergon, (b) Ring, (c) $(m * n)$ 2D Mesh.

through the two ring-related channels. ALast has an variable performance as the hotspots across channels may be overloaded of traffic. AEqualized algorithm instead optimizes the use of all three new NI channels sensibly improving the performance of the system.

Simulations with one, two and three hotspots and different network sizes of 8 up to 30 nodes confirm the reported results. Relative graph though are not reported for space limitations.

3.4 NoC Comparison

In the paper [BCG+07] presented at the DATE conference in 2006 we proposed a comparison between the well known Ring and 2D Mesh architectures [DYN03] and the new *Spidergon* NoC .

In this study we analyzed three scenarios: single and double hot spots and the uniform traffic patterns.

The Mesh NoC used a Dimension Order (called also XY algorithm [DYN03]) routing algorithm where at first, flits are forwarded towards their destination initially along the X direction (the horizontal link) until the column of the target node is reached. Then, flits are forwarded along the Y direction (vertical link) up to the target node.

In Ring, the routing strategy is straightforward, i.e. clockwise or counterclockwise direction is selected depending on the shortest path while in *Spidergon* we adopted the aFirst routing algorithm described previously in this chapter. It is to be noted that the Mesh routing algorithm resolves also the deadlock problem by limiting the path selection choices. Ring and *Spidergon* instead require VCs.

A significant worst case index, named the Network Diameter (*ND*) is defined as the maximum shortest path length between any pair of nodes in the topology. The average network distance E[D] is defined as the average path length of all different paths in the network. By assuming a NoC of N nodes we have:

- in a Ring topology:
  $ND = floor(N/2)$
  $E[D] = N/4$
- in $(m * n)$ 2D Mesh:
  $ND = (m + n - 2)$
  $E[D] = (m + n)/3$
- in *Spidergon* :
  $ND = ceiling(N/4)$
  $$E[D] = \begin{cases} (2x^2 + 4x + 1)/N \text{ if } N \sim 4x \\ (2x^2 + 2x - 1)/N \text{ if } N = 4x + 2 \end{cases}$$

Under the worst case analysis assumptions, the network diameter of real 2D Mesh topologies with N nodes shows quite unpredictable fluctuations between the ideal ($\sqrt{N} * \sqrt{N}$ mesh values and the Ring diameter values, as shown in Figure 3.20(a).

The analysis shows that the Spidergon NoC has lower *ND* than regular 2D Meshes at least up to 40-45 nodes (and after, depending on the value of N, see Figure 3.20(a)).

In Figure 3.20(b), we show the analysis results for the average network distance $E[D]$ for Ring, ideal and real 2D Meshes, and *Spidergon* . It results that *Spidergon* outperforms Ring, and works on the middle of the value range of the real mesh implementations.

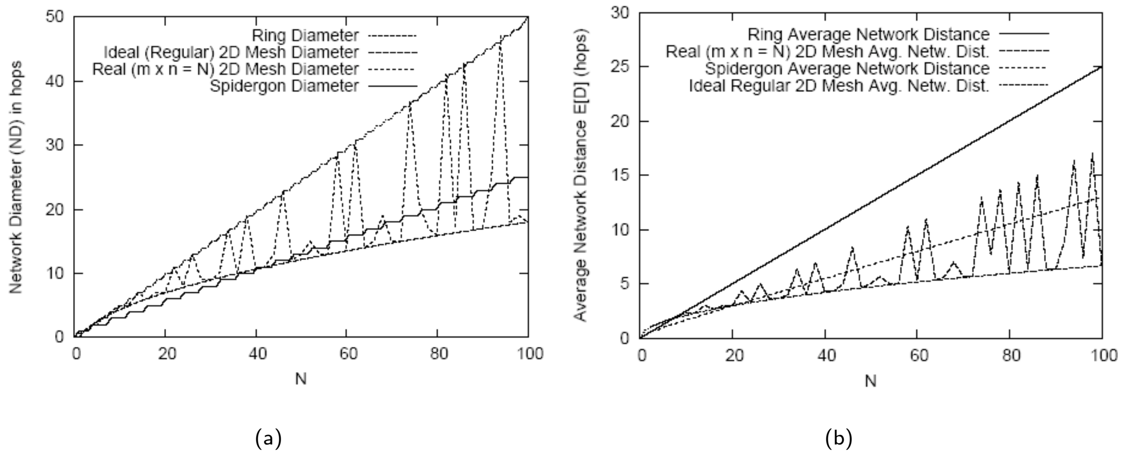


Figure 3.20: (a) Network Diameter ND and (b) Average Network Distance, vs. number of nodes N in Ring, ideal and real 2D Mesh and Spidergon NoCs.

Ideal mesh behavior is obtained by real Meshes only under specific N values (that is when $N = m * n$ and $m \sim n$). These results are quite indicative of the difference that may exist between theory results in ideal cases and real scenarios, for Mesh topologies. Results in Figures 3.20 show that *Spidergon* is expected to have competitive and linear behavior, on the average and worst case scenarios, due to node symmetry and regular topology with respect to real Ring and Mesh topologies.

In the following we will investigate the NoC support for communication under the routing strategies that we discussed previously

The modeling and simulation of the NoC architectures have been performed with the OMNeT++ simulation framework [SVE07]. OMNeT++ is a public source, and flexible simulation environment with strong GUI support that allows a fast and high-level simulation environment for NoC exploration topologies.

The node model for the *Spidergon* NoC is shown in Figure 3.21. Each node has an external network interface to connect the IP to the NoC.

The external IP can act as a packet source and/or as a packet destination (sink) depending on the simulated scenario. Packet sources adopt a exponential inter-arrival distribution of constant size packets (6 flits in our simulations), with variable parameter λ. The first (head) flit of a packet is sent to the routing mechanism of

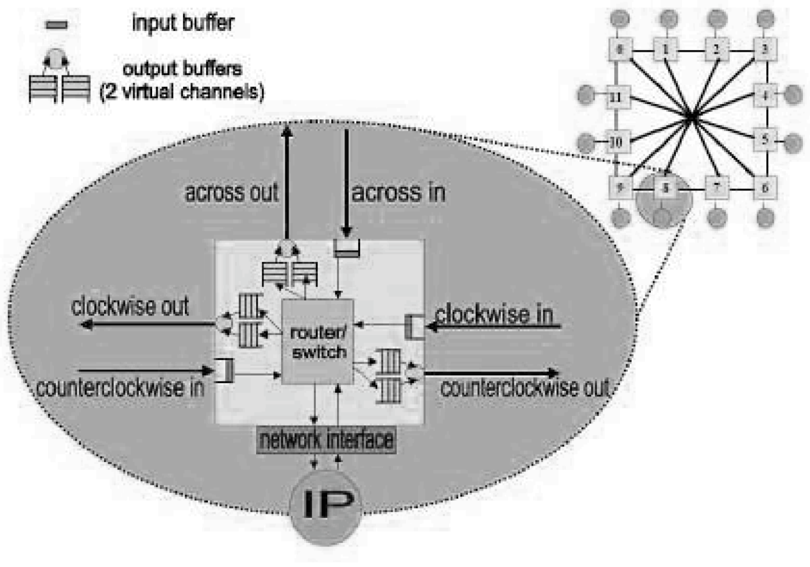


Figure 3.21: Model of a *Spidergon* STNoC node

the node, and then transferred on the output queue of the target channel (if room). Once the head flit has been processed by the routing element of a node, a switching mechanism is defined to forward all immediately following packet-flits to the buffers of outgoing links of the target path to the destination node. Application packets are consumed from the IP memory in a FIFO order.

The scheme in Figure 3.21 refers to *Spidergon* nodes. On the other hand, Ring and Mesh nodes considered in this analysis have been defined with the same node architecture, excepted the number of links, the cumulative buffers sizes, and the routing policies.

Specifically, Ring nodes have clockwise and counterclockwise links only, and Mesh nodes may have from 2 up to 4 links, by including *N, S, W* and *E* direction links. Incoming links have a one-flit buffer, while outgoing links have a pair of output buffers (used both for virtual channel management and deadlock avoidance) in Ring and *Spidergon* topologies, and one single buffer in Mesh topologies. All output buffers may contain up to three-flits.

Experiments have been performed by modifying the overall buffer capacity of nodes and buffer symmetry depending on the expected link usage. Results indicated

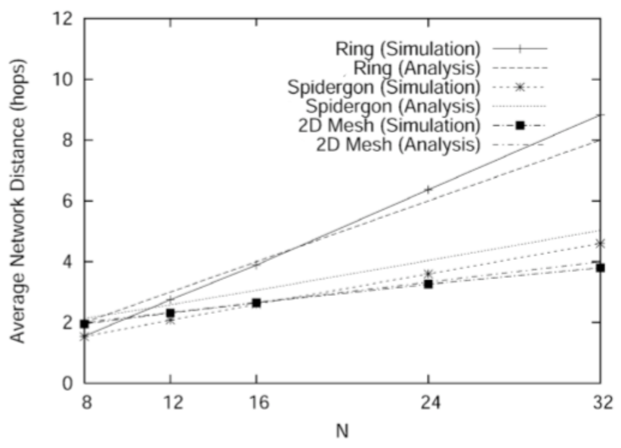


Figure 3.22: Analytical and simulation-based average network distances (hops).

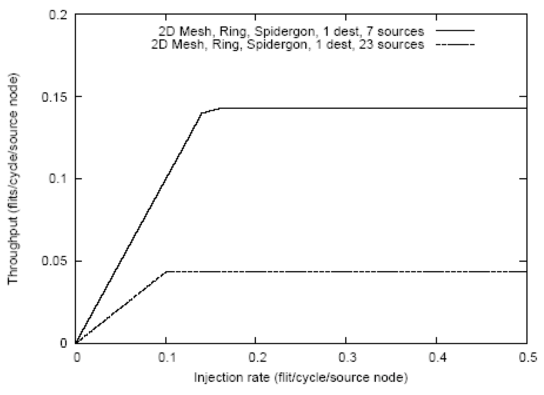
that small buffer tuning have some marginal impact on the peak performances. In the following we will illustrate and comment the results obtained in three basic scenarios: the single and double hot-spot target scenario, and the homogeneous sources and destinations scenario.

The first set of data shown is related to the validation of the simulation and analytical model. Figure 3.22 shows the analytically estimated average distance *E[D]* and the simulation-based value obtained. Despite some differences in the data, due to stochastic variability, the figure confirms that Ring has the worst average performances, while Spidergon and 2D Mesh topologies work close to each other in the range from 8 to 32 nodes.

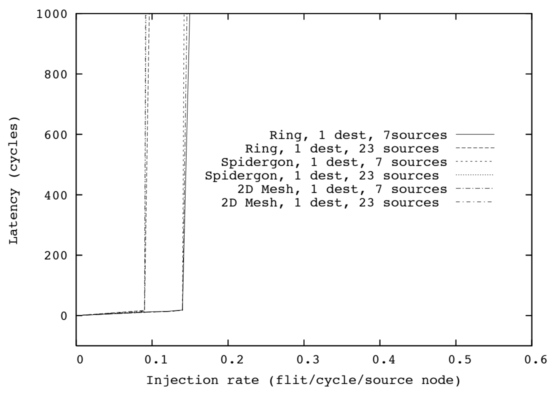
Single Hot-Spot Scenario

Figure 3.23(a) shows the throughput index of the NoC architecture as a function of the injection rate parameter of the source nodes when hot-spot target is present in the system (that is, one single destination node for all packets). Destination nodes have been taken in different points on the Mesh topology (in symmetric Ring and Spidergon this would not have difference).

The result from Figure 3.23(a) is that the throughput index presents no differences with respect to the implemented topology when one single target destination is



(a)



(b)

Figure 3.23: Average (a) throughput and (b) latency on a single hot spot traffic pattern.

adopted for all communications. The only difference is given by varying the number of source nodes. When all the sources homogeneously increase the injection rate, this translates to linear absorption from the (single) destination node, up to the destination node saturation is obtained. This means that the most significant system bottleneck under hot-spot traffic destination scenarios is the destination node, and not the NoC architecture and the channel buffering resources.

This result is quite different from the interpretation that can be obtained by assuming a uniform load distribution among many sources and many destinations. This does not mean that the NoC architecture is irrelevant, because the NoC architecture behaves better when parallel local communication is present. On the other hand, in todays common SoCs scenarios, when the system memory is external, the behavior obtained with different NoC topologies would converge to the behavior shown in Figure 3.23(a).

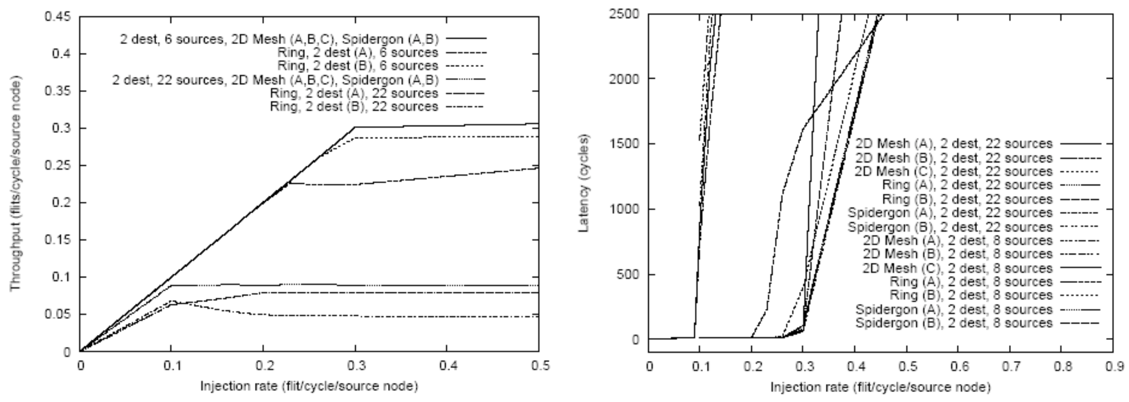
In other words, the scalable and symmetric architecture of *Spidergon* would give the same advantages of more complex solutions, like 2D Mesh, under the hot-spot communication viewpoint. In addition, *Spidergon* can outperform ring or a complex bus hierarchy when multiprocessors are presents (these data have been obtained and were not included in this paper due to space limitations).

Moreover, *Spidergon* introduces a degree of scalability and flexibility that would not be found in current bus architectures. For this reason, Spidergon appears as the good trade-off solution for obtaining the same Figure 3.22: analytical and simulation-based average network distances (hops) Figure 3.23(a): NoC throughput, one hot-spot destination node system (that is, one single destination node for all packets). Destination nodes have been taken in different points on the Mesh topology (in symmetric Ring and *Spidergon* this would not have difference). The result from Figure 3.23(a) is that the throughput index presents no differences with respect to the implemented topology when one single target destination is adopted for all communications. The only difference is given by varying the number of source nodes.

When all the sources homogeneously increase the injection rate, this translates to linear absorption from the (single) destination node, up to the destination node saturation is obtained. This means that the most significant system bottleneck under hot-spot traffic destination scenarios is the destination node, and not the NoC architecture and the channel buffering resources. This result is quite different from the interpretation that can be obtained by assuming a uniform load distribution among many sources and many destinations. This does not mean that the NoC architecture is irrelevant, because the NoC architecture behaves better when parallel local communication is present. On the other hand, in todays common SoCs scenarios, when the system memory is external, the behavior obtained with different NoC topologies would converge to the behavior shown in Figure 3.23(a).

In other words, the scalable and symmetric architecture of *Spidergon* would give the same advantages of more complex solutions, like 2D Mesh, under the hot-spot communication viewpoint. In addition, *Spidergon* can outperform ring or a complex bus hierarchy when multiprocessors are presents (these data have been obtained and were not included in this paper due to space limitations).

Moreover, Spidergon introduces a degree of scalability and flexibility that would not be found in current bus architectures. For this reason, Spidergon appears as the good trade-off solution for obtaining the same performances of more complex



(a) (b)

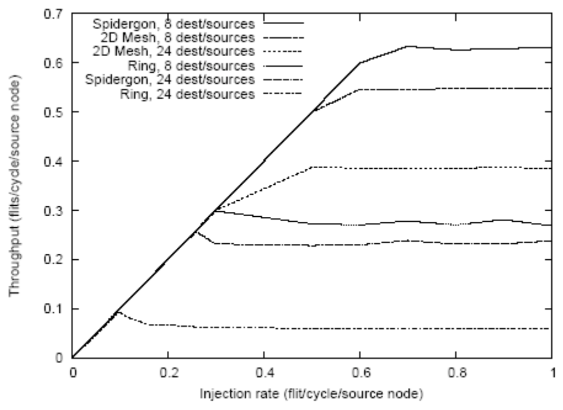
Figure 3.24: Average (a) throughput and (b) latency on a double hot spot traffic pattern.

architectures, under common scenarios in current SoCs.

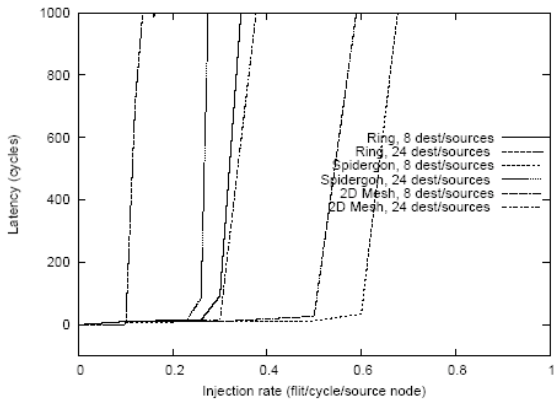
Figure 3.23(b) shows the average latency obtained by Spidergon, 2D Mesh and Ring topologies under one single hot-spot destination node, as a function of the number of nodes N and the injection rate parameter of multiple source nodes. Data show that the latency sharply increases when the target node saturation is obtained, with little differences due to the NoC topology adopted. By assuming an homogeneous injection rate, the latency increases early when the number of source nodes increases, as expected.

Double Hot-Spot Scenario

Simulations have been performed by considering a pair of hot-spot target scenarios, and by allocating the targets in different positions inside the NoC topologies. For 2D Mesh, scenario A is with 2 targets on the opposite corners (nodes 1 and N), scenario B is with one target in the corner (node 1) and the second one in the middle (node 5 with $2 * 4 = 8$ Mesh and node 14 with $4 * 6 = 24$ Mesh), and scenario 3 is with both targets in the middle (nodes 5 and 6 with $2 * 4 = 8$ Mesh, and nodes 14 and 15 with $4 * 6 = 24$ Mesh). In Ring and Spidergon, scenario A is with two targets in opposition (North-South position) on the ring, and scenario B is with two targets in



(a)



(b)

Figure 3.25: Average (a) throughput and (b) latency on a homogeneous traffic pattern.

North and West positions on the ring. The results reported in Figure 3.24 (a) and (b) basically confirm the system behavior and conclusions discussed for one hot-spot target.

Homogeneous Sources/Destinations Scenario

Figure 3.25(a) shows the throughput results with respect to the NoC topology and the number of nodes, under homogeneous scenarios with uniform distribution of sources and destinations. Specifically, all the nodes behave like sources and can be addressed as destination for packets, with uniform probability distribution. When all node sources increase the injection rate, this translates to linear absorption from all the destination nodes, up to the set of destination nodes and/or the network become saturated. This performance index illustrates that Spidergon and 2D Mesh topologies outperform Ring, and scale better when the number of nodes is low. Under this scenario, 2D Mesh shows a better throughput than Spidergon only with many nodes and when the local injection rate of all source nodes is greater than 0.3 flits/cycle. On the other hand this scenario is hardly obtained in real systems, and this does not constitutes a good motivation to prefer the adoption of 2D Mesh in favor of the Spidergon topology. As expected, the bottleneck emerging in this scenario is basically given by the communication infrastructure. This is confirmed also by the worst performances obtained by the Ring topology.

Figure 3.25(b) illustrates the average latency obtained by Spidergon, 2D Mesh and Ring topologies under homogeneous source and destination distribution scenarios. All the nodes behave like sources and can be addressed as destination for packets with uniform probability distribution. Latency is shown as a function of the number of nodes N, and the injection rate parameter of multiple source nodes.

Data show that the latency sharply increases when the network saturation is obtained, with some differences due to the different saturation properties of the NoC topology adopted. By assuming an homogeneous injection rate, Ring topology saturates first, and the latency generally increases early when the number of system nodes increases, accordingly with the throughput results obtained for the same scenarios. patterns originated by common applications, and analysis of routing protocols and additional NoC topologies.

3.5 Conclusions

We presented a novel routing algorithm called *Across Equalized* or aEqualized for the Spidergon Network on Chip. AEqualized balances the traffic on the input channels of the network's hotspots by assigning to each router either the aLast or aFirst routing algorithm. This "tagging" operation depends on the position of each single node with respect to the hotspots of the network.

Considering the standard implementation of a Spidergon router: with four ports (NI, Across, Clockwise and Counter Clockwise), in our case study, this algorithm essentially maintains the performances of the other two algorithms by reducing the number required across links by up to 60% and the general required connection by a 13%.

Considering an enriched version of the routers with three ports towards the network interface this algorithm sensibly improves the performance of the system by maintaining the reduced amount of links and buffers.

In our future work we plan analyze the aEqualized routing algorithm under more realistic traffic patterns and to improve the enriched routers channels assignment policy.

We also illustrated the modeling and simulation-based analysis of low degree topologies (Ring, 2D Mesh and Spidergon) focusing the on-chip domain requirements and overcoming the classical parallel computing and networking results. To the best of our knowledge, this is the first work considering irregular mesh topologies, whose analysis is motivated since regular meshes cannot be always assumed as realistic topologies in SoCs. On the other hand, the first deep analysis of the novel Spidergon topology is presented, resulting in a good compromise among the well-known topologies. This has been demonstrated by analyzing Spidergon characteristics with respect to real 2D Mesh and Ring topologies, and by considering common hot- spot communication scenarios that characterize current SoC architectures. This analysis has shown that Spidergon can be considered a good solution under the system design, the ease of implementation and management viewpoint, with performance and scalability results that are in line with other more complex solutions under most common assumptions and scenarios. Future work will include the extension of the analysis and simulation with more NoC nodes, specific traffic patterns originated by common applications, and analysis of routing protocols and additional NoC topologies

Chapter 4

Task Mapping

Network-on-chip (NoC) provides a high performance, scalable and power-efficient communication infrastructure to either Chip Multi-Processor (CMP) and System on Chip (SoC) systems [NTIJ04]. A NoC usually consists of a packet-switched on-chip micro-network, foreseen as the natural evolution of traditional bus-based solutions, such as AMBA *AXI* [amb], and IBM's *Core Connect* [IC]. Innovative NoC architectures include the LIP6 *SPIN* [AAZ03], the M.I.T. *Raw* [MT02], the VTT (and various Universities) *Eclipse* [For02] and *Nostrum* [GRK$^+$05], PHILIPS' *Æthereal* NoC [GDvM$^+$03], and Stanford/Uni-Bologna's *Netchip* [BM02, JMBM04].

These architectures are based on direct point-to-point topologies, in particular Meshes, tori and fat trees offering simple and efficient routing algorithms based on small area, high frequency routers.

A major challenge for predicting performance and scalability of a particular NoC architecture relies on precise specification of real application traffic requirements arising from current and future applications, or scaling of existing applications. For example, it has been estimated that SoC performance varies by up to 250% depending on NoC design, and up to 600% depending on communication traffic [LRD01], while NoC power dissipation can be reduced by more than 60% by using appropriate mapping algorithms [HM03].

Future MPSoC applications require scalable NoC topologies to interconnect the IP cores. We have developed new tools for NoC design space exploration and effi-

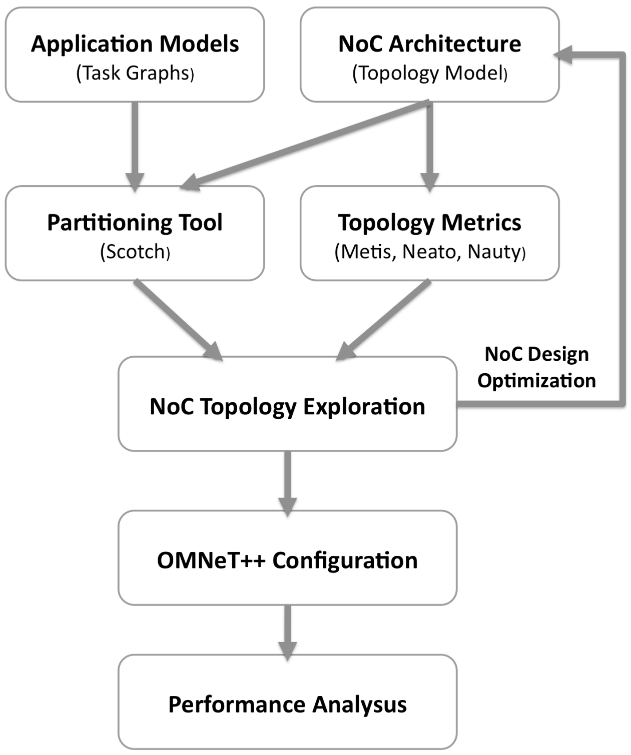


Figure 4.1: Our design space exploration approach for system-level NoC selection.

cient NoC topology selection by examining theoretical graph properties, as well as application mapping through task graph partitioning. These tools are derived by extending existing tools in parallel processing, graph theory and graphical visualization to NoC domain. Besides enabling efficient NoC topology selection, our methods and tools are important for the design of efficient multi-core systems.

Our NoC design space exploration approach is explained in Figure 4.1. We consider both theoretical metrics, e.g. number of nodes and edges, diameter, average distance, bisection width, connectivity, maximum cut and spectra, as well as embedding quality metrics for mapping various applications onto NoC resources, such as computing, storage and FPGA elements.

The mapping algorithm of the partitioning tool obtains an assignment of application components onto the NoC topology depending on abstract requirements formulated as static or dynamic (runtime) constraints on application behavior components and existing NoC architectural and topological properties.

Previous papers have studied application embedding onto conventional symmetric NoC topologies. Hu and Marculescu examined mapping of a heterogeneous 16-core task graph representing a multimedia application onto a Mesh NoC topology [HM03], while Murali and De Michelli used a customized tool (called SUNMAP) to map a heterogeneous 12-core task graph representing a video object plane decoder and a 6-core DSP filter application onto a Mesh or torus NoC topology using different routing algorithms [MM04a, MM04b].

Related Work: The proprietary SUNMAP tool, proposed by Stanford and Bologna University, performs NoC topology exploration by minimizing area and power consumption requirements and maximizing performance characteristics for different routing algorithms. Alike our approach shown in Figure 4.1, the XPIPES compiler can eventually extract efficient synthesizable SystemC code for all network components, i.e. routers, links, network interfaces and interconnect, at the cycle- and bit-accurate level. Our study generalizes previous studies by considering a plethora of theoretical topological metrics, as well as application patterns for measuring embedding quality metrics. It focuses on conventional NoC topologies, e.g. Mesh and torus, as well as practical, low-cost *circulants*: a family of graphs offerings small network size granularity and good sustained performance for realistic network sizes (usually below 64 nodes).

In Section 4.2 we describe the tools we used to study different NoC architectures in order to understand their topological properties. In Section 4.1 we describe traffic patterns used in our analysis. In particular, we focus on the TGFF tool used for generating *synthetic application task graphs* in our simulations.

In Section 4.3, we describe the problem of *application task graph mapping*. We define the adopted metrics to rate the quality of a given mapping and describe the SCOTCH partitioning tool used to map a given task graph onto the considered network on chip.

In Section 4.4 we describe OMNET++ , the simulation framework we used to obtain out system-level simulations.

In Section 4.5, we report a case-study consisting of task generation, mapping analysis, and bit- and cycle-accurate system-level NoC simulation for a set of synthetic tree-based task graphs, as well as for a more realistic Mpeg4 encoder application.

Finally, in Section 4.6, we draw conclusions and consider interesting ramifications of our work.

4.1 Synthetic Traffic Models

Applications are often represented as *task graphs*, thus expressing the necessary communication and synchronization patterns for realizing a particular computation.

Task graphs are basic IP blocks with clear, unambiguous and self-contained functionality interacting together to form a NoC application.

Task graph embedding is also used by the operating system for reconfiguring faulty networks, i.e. providing fault-free virtual sub-graphs in "injured" physical system graphs to maintain network performance (bandwidth and latency) in the presence of a limited number of faults.

Vertices (or nodes) represent computation, while links represent communication. A node numbering scheme in *directed acyclic graphs* (DAGs) takes into account precedence levels. For example, an initial node is labelled node 0, while an interior node is labelled j, if its highest ranking parent is labelled $j-1$.

Undirected and directed acyclic task graphs represent parallelism at both coarse and fine grain. Examples of coarse grain parallelism are inter-process communications, control and data dependencies and pipelining. Fine grain parallelism is common in multimedia processing, e.g. in data parallel prefix operations and loop optimizations.

4.2 Graph Theoretical Analysis

In order to explore inherent symmetry and topological properties in alternative

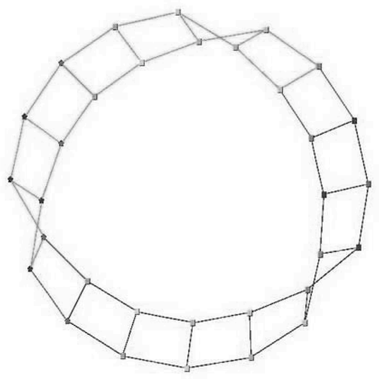


Figure 4.2: Metis visualization of the Spidergon NoC layout.

constant degree NoC topologies (especially chordal rings) we have considered exploring theoretical knowledge by combining together several available open-source and free packages. The eventual goal is to support NoC selection at system-level using an array of customized design tools.

More specifically, this approach is based on several steps. After NAUTY and METIS analyze automorphisms as explained below, NEATO can display the graph so that and graph properties and topologically-equivalent vertices are shown; two vertices are equivalent (identical display attributes), if there is a vertex-to-vertex bijection preserving adjacency. From these graphs we can observe scalability issues, e.g. concerning bisection width.

- Karypis' and Kumar's METIS provides an extremely fast, multilevel graph partitioning embedding heuristic that can also extract topological metrics, e.g. diameter, average distance, in/out-degree, and bisection width [KK97]. Concerning edge bisection, for small graphs, ($N < 40$) nodes, a custom-coded version of Lukes's exponential-time dynamic programming approach to partitioning provides an exact bisection if one exists [Luk75]. For larger graphs, METIS partitioning is used to approximate a near-minimum bisection width;

- McKay's NAUTY computes the automorphisms in the set of adjacency-preserving vertex-to-vertex mappings. NAUTY also determines the *orbits* that partition graph vertices into equivalence classes, thus providing symmetry and topological metrics [McK];

- AT&T's NEATO is used for visualizing undirected graphs based on spring-relaxation and controlling the layout, while supporting a variety of output formats, such as PostScript and Gif [Nor].

These properties can help discover NoC topologies with:

- small, constant network extendibility;
- small diameter for less than 100 nodes;
- large edge bisection width that scales;
- efficient (wire balanced) point-to-point routing without pre-processing;
- efficient intensive communication algorithms, e.g. broadcast, scatter and gather;
- good fault tolerance;
- efficient VLSI layout with short, mostly local (small chordal links) wires;

4.2.1 Generating Synthetic Graphs using TGFF

In 1998, Dick and Rhodes originally developed *Task Graphs For Free* (TGFF) as a C++ software package that facilitates standardized pseudo-random benchmarks for scheduling, allocation and especially hardware-software co-synthesis [DRW98]. TGFF provides a flexible, general-purpose environment with a highly configurable random graph generator for creating multiple sets of synthetic, pseudo-random directed acyclic graphs (DAGs) and associated resource parameters that model specific application behavior. DAGs may be exported into postscript, VCG graphical visualization or text format for importing them into mapping or simulation frameworks; notice that VCG is a useful graph display tool that provides color and zoom [VCG].

TGFF users defines a source (*.tgffopt) file that determines the number of task graphs, the minimum size of each such graph, and the type of nodes and edges

through a set of parameterized commands and database specifications.
For example, random trees are constructed recursively using series-parallel chains, i.e. at least one root node is connected to multiple chains of sequentially linked nodes.

Ranges for the number of chains, length of each chain and number of root nodes are set by the user using TGFF commands. Notice that chains may also rejoin with a given probability by connecting an extra (sink) node to the end of each chain. TGFF includes many other support features, such as:

- Indirect reference to task data; task attribute information is provided through references to processing element tables for node types or transmission tables for communication edge types.
- user-defined graph attributes: generating statistics for node or edge performance, power consumption, or reliability characteristics;
- real-time processing through an association of tasks to periods and deadlines;
- multi-rate task graphs: tasks exchange data at different rates either instantaneously or using queues;
- multi-level hierarchical task graphs, where each task is actually a task graph; this is possible by interpreting task-graph 1 as the first task in task-graph 0, task-graph 2 as the second task in task-graph 0, etc; there are certain restrictions.

Application graph structures are generated using TGFF in several research and development projects. For example, TGFF is being used for application task graph generation in heterogeneous embedded systems, hardware software co-design, parallel and distributed systems and real-time or general-purpose operating systems.

Within the NoC domain, TGFF is commonly used in energy-aware application mapping, hw/sw partitioning, synthesis optimization, dynamic voltage scaling and power management. In this respect, all tree-like benchmarks (see Section 4.5) have been generated using our customized version of the TGFF package. Since these task graphs are deterministic, we had to modify TGFF to avoid recursive constructions and impose lower bounds on the number of tasks.

4.3 Task Mapping for SoC

A mapping algorithm selects the most appropriate assignment of tasks onto the nodes of a given NoC architecture. In complex, realistic situations, all combinations of task assignments must be considered. In most cases, a near-optimal solution that approximately minimizes a cost function is computed in reasonable time using heuristic algorithms. The heuristic takes into account the type of tasks, the number and type of connected nodes, and related constraints, e.g. possibly required architecture, operating system, memory latency and bandwidth, or total required memory for all tasks assigned to the same node.

After the mapping algorithm obtains a near optimal allocation pattern for the given task graph, the operating system can initiate automated task allocation onto the actual NoC topology nodes.

4.3.1 Quality Metrics for Application Embedding

Static or dynamic *mapping* is a network transformation technique based on graph partitioning; a mapping is static if it is computed prior to application execution, and is never modified afterwards. Graph *partitioning* decomposes a target graph into clusters for a broad range of applications, such as VLSI layout or parallel programming.

More specifically, given a graph $G(n, m)$ with n weighted vertices and m weighted edges, graph partitioning refers to the problem of dividing the vertices into p cluster sets, so that the sum of the vertex weights in each set is as close as possible (balanced total computation load), and the sum of the weights of all edges crossing between sets is minimized (minimal total communication load).

Unfortunately, even in the simple case where edge and vertex weights are uniform and $p = 2$, graph partitioning onto an arbitrary NoC topology is NP-complete [GJS76]. Hence, in general, there is no known, efficient algorithm to solve this problem, and it is unlikely that such an algorithm exists. Thus, we resort to heuristics that partially compromise certain constraints, such as balancing the communication load, or (more typically) using approximate communication load minimization constraints, i.e. maximizing locality and look ahead time by statically mapping intensive inter-process communication to nearby tasks. These constraints are often specified in an abstract way through a cost function, which may also consider more complex constraints, such as minimizing the total communication load among all NoC components, e.g. for optimizing total power consumption during data exchanges. Although this function is is clearly application dependent, it is usually expressed as a weighted sum of terms representing load on different NoC topology nodes and communication links, considering also user-defined optimality criteria, e.g. in respect to architecture, such as shortest-path routing, such as number or speed of processing elements, communication links, and storage elements.

Graph partitioning heuristics are usually based on recursive bisection using either global (inertial or spectral) partitioning methods or local (Kernighan-Lin) refinement techniques. Results of global methods can be fed on local techniques, which often leads to significant improvements in performance and robustness. Thus, with bipartitioning, the graph is partitioned into two halves recursively, until a desired number of sets is reached; notice that quadrisection and octasection algorithms may achieve better results.

Popular global partitioning methods are classified into inertial (based on 1-d, 2-d or 3-d geometrical representation) or spectral (using Eigenvectors of the Laplacian of the connectivity graph). For a long time, the Kernighan-Lin algorithm has been the only efficient local heuristic and is still widely used in several applications with some modifications, such as Fiduccia and Mattheyses [FM82].

Graph *embedding* optimally assigns data and application tasks (IPs) to NoC resources, e.g. RISC/DSP processors, FPGAs or memory, thus forming a generic binding framework between SoC application and NoC architectural topology. Graph embedding also helps map existing applications onto a new NoC topology by porting (with little additional programming overhead) existing strategies from common NoC topologies. Embedding algorithms are usually based on graph partitioning.

Mathematically, an embedding of a source graph G_S onto a given target graph G_T is an injective function from the vertex set of G_S to the vertex set of G_T. Performance metrics for evaluating the embedding quality of a partitioning algorithm includes application-specific embedding quality and platform-specific performance metrics, such as time for executing the selected mapping time.

Common graph-theoretic application-specific embedding quality metrics are listed below.

- **Edge dilation**: of an edge of G_S is defined as the length of the path in G_T onto which an edge of G_S is mapped. The dilation of the embedding is defined as the maximum edge dilation of G_T. Similarly, we define average and minimum dilation metrics. These metrics measure latency overhead during point-to-point communication in the target graph G_T. A low dilation is usually beneficial, since most communication devices are located nearby, and hence the probability of higher application throughput increases;

- **Edge Expansion**: refers to a weighted-edge graph G_S. It multiplies each edge dilation with its corresponding edge weight. The edge expansion of the embedding is defined as the maximum edge expansion of G_T. Similarly, we define average and minimum edge expansion metrics;

- **Edge Congestion**: is the maximum number of edges of G_S mapped on a single edge in G_T. This metric measures edge contention in global intensive communication;

- **Node Congestion**: is the maximum number of paths containing any node in G_T where every path represents an edge in G_S. This metric is a measure of node contention during global intensive communication. A mapping with high congestion causes many paths to traverse through a single node, thus increasing the probability of a network traffic bottleneck due to poor load balancing;

- **Node Expansion**: also called load factor or compression ratio, is the ratio of the number of nodes in G_T to the number of nodes in G_S. Similarly, maximum node expansion represents the maximum number of nodes of G_S assigned to any node of G_T;

- **Number of Cut Edges**: i.e. edges incident to vertices of different partitions. Cut edges represent extra (inter-module) communication required by the mapping. This metric is used for comparing target graphs with identical number of edges, the smaller metric the better.

In the following Sections we will examine dilation, expansion and congestion metrics for a number of traffic patterns interesting to the SoC domain, as well as for mapping interesting communication patterns arising from real applications onto Spidergon and other prospective NoC topologies. Thus, many algorithms originally developed for common Mesh and torus topologies may be emulated on the Spidergon . Furthermore, since embedding of common application graphs, e.g. binary trees on Mesh, has already been investigated, we can derive embedding of these graphs onto Spidergon through composition.

4.3.2 The Scotch Partitioning Tool

The Scotch project (1994-2001) at *Université Bordeaux I* - LaBRI focuses on libraries for statically mapping any possibly-weighted source graph onto any possibly-weighted target graph, or even onto disconnected sub-graphs of a given target graph [PR96]. Scotch maps graphs in linear time to the number of edges in the source graph, and logarithmic time to the number of vertices in the target graph.

Scotch has two forms of license: private version licensed for commercial applications, and public version available for academic research. The academic distribution consists of a Unix executable along with library documentation, sample graphs

```
0
8 24
0 100
6 2 5 4
5 6 6 7 2 3 1 0
7 2 5 4
2 2 5 4
4 6 3 1 0 6 7 2
3 2 4 5
1 2 4 5
0 2 4 5
```

Figure 4.3: Source file for SCOTCH partitioning tool.

and free access to source code. SCOTCH builds and validates source and target graphs and then displays obtained mappings in colorful graphs [PR96]. It easily interfaces to other partitioning or theoretical graph analysis programs, e.g. METIS or NAUTY, due to standardized vertex/edge labeling formats.

SCOTCH operates by taking as input a *source* file (.src) that represents the application task graph to be mapped. Figure 4.3 shows a snapshot of a sample source file.

The first three lines of the file represent some configuration info such as file version number, number of vertex and edges and other file-related options.

From the fourth line onwards, the source file represents the communication task graph, where the first entry column represents the considered node's *id*, the second the number of destinations, and then the list of destination ids. For example the third line in Figure 4.3 says that node 6 communicates with two destinations: nodes 5 and 4. In case of different communication bandwidth next to each destination id there is the traffic bandwidth between the source and the specific destination.

Geometry files have a .xyz extension and hold the coordinates of the vertices

```
8
5 5
6 0
7 1
2 3
4 2
3 4
1 7
0 6
S Strat=b{job=t,map=t,poli=S,
strat=m{asc=f{type=b,move=80,
pass=-1,bal=0.005},
low=h{pass=10},type=h,vert=80,rat=0.7}x}

M Processors 8/8 (1)
M Target min=1 max=1 avg=1 dlt=0 maxmoy=1
M Neighbors min=2 max=6 sum=24
M CommDilat=1.666667 (20)
M CommExpan=1.666667 (20)
M CommCutSz=1.000000 (12)
M CommDelta=1.000000
M CommLoad[0]=0.000000
M CommLoad[1]=0.500000
M CommLoad[2]=0.333333
M CommLoad[3]=0.166667
```

Figure 4.4: Target file for Scotch partitioning tool.

of their associated graph. They are used by visualization programs to compute graphical representations of mapping results.

Target files are the result of a mapping computation in SCOTCH . Figure 4.4 shows the result of such a mapping. The first element states the number of nodes mapped. The following two columns are the pairs:

$$< \textit{architecture node id},\ \textit{application node id} >$$

SCOTCH then generates the metrics relative to the mapping that we discussed above. We have modified the TGFF package for graph generation to adopt SCOTCH format for defining application-source graphs so that:

- Source graphs (*.src) are generated either by the user or through the TGFF tool (see Section 4.2.1);

- Geometry files (*.xyz) are generated either by the user i.e. Spidergon NoC or by the SCOTCH partitioning tool for common graphs, such as Mesh or Torus;

- Target NoC topology graphs (*.tgt) are generated automatically from corresponding source graphs using the SCOTCH partitioning tool.

Mapping algorithms for simple application graphs, such as rings or trees have been studied extensively in parallel processing, especially for direct networks, such as hypercubes and meshes [Lei06]. For general graphs several mapping algorithms exist, e.g. Kerninghan-Lin algorithm for VLSI layout, or simulated annealing techniques.

Simulated annealing first defines an initial mapping based on the routing function, e.g. shortest-path, dimension-order or non-minimal path. Then, this algorithm always accepts injection of new disturbances that reduce an appropriately defined cost function that measures the relative cost of the embedding, while it accepts only with a decreasing probability the injection of new disturbances that increase the relative cost function. SCOTCH features extremely efficient multi-level partitioning methods based on recursive graph bipartitioning [PR96]. More specifically, initial and redefined bipartitions use:

- Fiduccia-Mattheyses heuristics that handle weighted graphs;

- randomized and backtracking methods;
- greedy graph-growing heuristics;
- a greedy strategy derived from Gibbs, Poole, and Stockmeyer algorithm;
- a greedy refinement algorithm designed to balance vertex loads.

SCOTCH application developers can select the best partitioning heuristic for each application domain by changing partitioning parameters.

4.4 The OMNeT++ Simulation Framework

OMNeT++ is an object-oriented modular discrete event network simulator [Knu91]. The source code is freely available for the academic community, while it requires a license for commercial use. OMNeT++ offers a number of libraries and tools that allow a user to rapidly develop complex simulation projects providing:

- a graphical tool to define the simulator skeleton: this allows the user to easily define the different agents acting in the environment to be simulated as well as delineating the relations and hierarchies existing among them;
- a library for automatic handling of inter-process signaling and messaging;
- a library implementing the most important, commonly used statistical probability distribution functions;
- an interesting graphical user interface, that allows the user inspect and interact with the simulation at run-time by allowing hip/her to modify parameters, inspect objects or plot run-time graphs;
- a number of tools that collect, analyze and plot the simulation results, and
- many freely developed models for wired/wireless network communication protocols like TCP-IP, IEEEE 802.11 or ad hoc routing protocols.

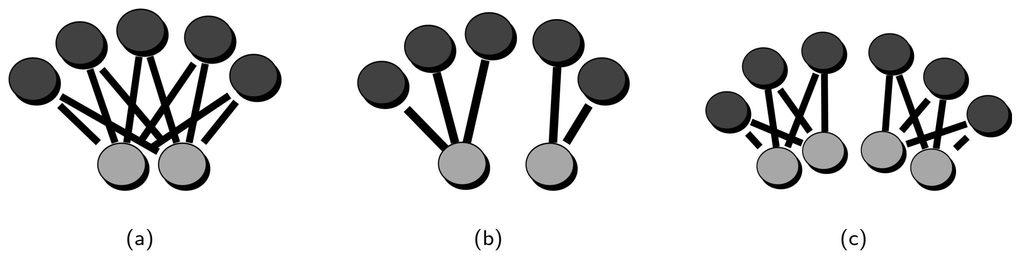


Figure 4.5: Application models for: (a) 2-rooted forest (SRF) , (b) 2-rooted tree (SRT), (c) 2-node 2-rooted forest(MRF) application task graphs.

In contrast to an already existing SystemC model, the OMNeT++ model hides many low-level details relative to NoC implementation in order to concentrate on understanding the effects caused by major issues like core mapping, routing algorithm selection and communication buffer sizing of the router and network interface nodes. Of course, we do not completely ignore details on these resources (especially the router), but rather treat them as constant parameters that also influence network performance.

4.5 A Case Study

In this case study, we consider embedding application task graphs onto several prospective NoC topologies.

First we describe the traffic patterns that we intend to use then we describe the NoC we analyzed and the result of embedding the considered applications onto the considered NoC.

Finally we present the OMNeT++ based simulation results of a selected subset of the considered applications and NoC.

4.5.1 Application Task Graphs

Any application can be modeled using a directed or undirected task graph. In our study, we consider three classes of tree-like benchmarks obtained through the

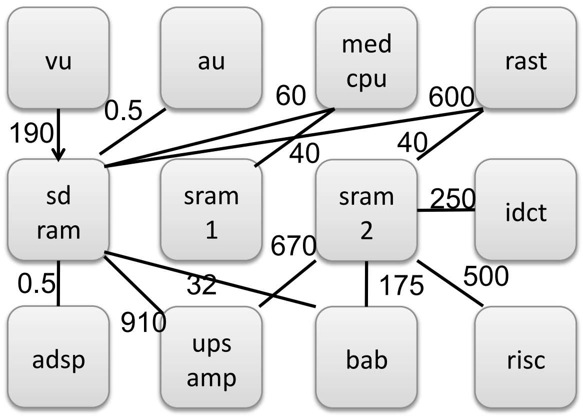


Figure 4.6: The Mpeg4 decoder task graph.

TGFF package. Each task graph had subset of nodes acting as traffic generators (initiators) and the remaining nodes acted as sink (target):

- **Single multi-Rooted Forest**: (SRF) the target sub-set of nodes is addressed by all the initiator nodes (Figure 4.5(a));
- **Multiple node-disjoint Single-Rooted Trees**: (SRT) initiator nodes are partitioned in subsets each set then communicates to one single target node (Figure 4.5(b));
- **Multiple node-disjoint Multi-Rooted Forests**: (MRF) is the combination of the first two traffic patterns: initiator and target nodes are slit in disjoint sets. Each set of initiator communicate with a single set of target nodes (Figure 4.5(c)).

We also considered a real 12-node Mpeg4 task graph (shown in Figure 4.6).

All the considered task graphs, with the exception of the Mpeg one, are undirected with unit node weights, and all have unit edge weights and scale with the NoC size. Hence the number of tasks always equals the network size, which ranges from 8 to 64 with step 4.

4.5.2 Prospective NoC Topology Models

The choice of NoC topology has a significant impact on MPSoC price and performance. The bottleneck in sharing resources efficiently is not the number of routers, but wire density which limits system interconnection, affects power dissipation, and increases both wire propagation delay and RC delay for driving the wires. Thus, in this study, we focus on regular, low-dimensional, point-to-point packet-switched topologies with few short, fat and mostly local wires.

As target NoC topology models we have considered low-cost, constant degree NoC topologies, such as single-dimensinal Array, Ring, 2-D Mesh and the Spidergon NoC.

We also considered the Crossbar architecture to have a comparison with the classical bus-based architectures. A large Crossbar is prohibitively expensive (in terms of number of links), but optimal solution in terms of embedding quality metrics (with unity edge dilation) for all patterns. Modern Crossbars connect IP blocks with various data widths, clock rates. and socket or bus standards, e.g. OCP, and AMBA AHB or AXI. Although system throughput, latency and scalability problems can be resolved by implementing the Crossbar as a multistage network based on smaller Crossbars and resorting to complex pipelining, segmentation and arbitration, a relatively simple, low-cost alternative is the unbuffered Crossbar switch. Thus, we also compare performance of the unbuffered Crossbar relative to ring, 2-d Mesh and Spidergon.

Although multistage networks with multiple layers of routers have nice topological properties, e.g. symmetry, small degree and diameter and large bisection, they have small network extendibility, many long wires and large wire area, thus they are not appropriate for NoC realization.

4.5.3 The Spidergon Network on Chip

Spidergon is a state-of-the-art low-cost on-chip interconnect developed by ST Microelectronics [CGL+08]. It is based on three basic components: a standardized

Figure 4.7: The Spidergon topology translates to simple, low-cost VLSI implementation.

network interface (NI), a worm-hole router, and a physical communication link.

Spidergon generalizes the ST Microelectronics' circuit-switched ST Octagon NoC topology defined as a cartesian product of basic octagons with a computing resource connected to each node. Spidergon is based on a simple bidirectional ring, with extra cross links from each node to its diagonally opposite neighbor. It is a *chordal ring* that belongs to the family of *undirected k-circulant graphs*, i.e. it is represented as a graph $G(N; s_1; s_2; ...; s_k)$, where N is the set of nodes, and $0 \leq s_i \leq |N|$, where s_i is an undirected edge between any node l and node $(l + s_i) mod |N|$.

Thus, more formally, Spidergon is a vertex-symmetric 3-circulant graph with an even number of nodes $|N| = 2n$, where $n = 1, ..., k = 2$, $s_1 = 1$ and $s_2 = (l + n) mod |N|$.

Chordal rings are circulant graphs with $s_1 = 1$, while double loop networks are chordal rings with $k = \{2, 4, 5, 9, 15, 16, 17\}$. Since early 1980s with the design of Illiac IV, these families have been proposed as simple alternatives to parallel interconnects, in terms of asymptotic graph optimality, i.e. minimum diameter for a given number of nodes and constant degree, see Moore graphs [E.W]. These theoretical studies ignore important design aspects, e.g. temporal and spatial locality, latency hiding and worm-hole routing, and NoC-related constraints.

The total number of edges in Spidergon is $\frac{3N}{2}$, while the network diameter is $\lceil \frac{N}{4} \rceil$. For realistic NoC configurations with up to 60 nodes, Spidergon has a smaller diameter and number of edges than fat-tree or Mesh topologies, leading to latency reduction for small packets. For example, the diameter of a 4x5 Mesh with 31 bi-directional edges is 7, while that of a 20-node Spidergon with 30 bi-directional edges and less wiring complexity is only 5.

As shown in Fig. 4.7, Spidergon has a practical low-cost, short wire VLSI layout implementation with a single crossing. Notice that VLSI area relates to edge bisection, while the longest wire affects NoC latency.

In this article we considered the Across-First (aFirst) Spidergon routing algorithm . It is a symmetric algorithm and since the topology is vertex-transitive it can be described at any node. For any arriving packet, the algorithm selects the cross communication port at most once, always at the beginning of each packet route. Thus, only packets arriving from a network resource interface need to be considered for routing. All other packets maintain their sense of direction (clockwise, or anti-clockwise) until they reach their destination.

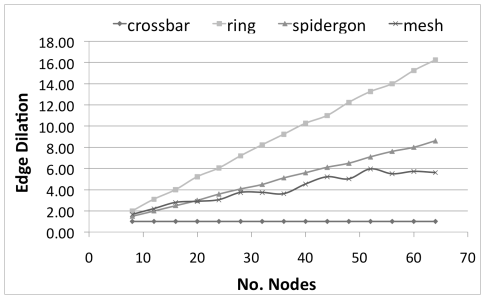
AFirst can be made deadlock-free by using virtual channels that break cycles in the channel dependency graph [Dal90, DA93, DYN03]. Furthermore, optimized, load-balanced virtual channel allocation based on static or dynamic datelines (points swapping of virtual circuits occurs) may provide efficient use of network buffer space, thus improving performance by avoiding head-of-line blocking, reducing network contention, decreasing communication latency and increasing network bandwidth [CGL+08].

4.5.4 Task Graph Embedding Analysis

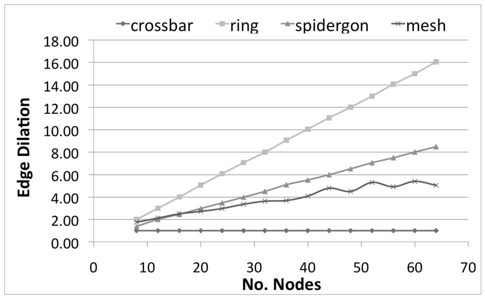
Through Scotch partitioning, we have mapped the application graphs described in Section 4.5.1 onto several low-cost NoC topologies (represented with *.tgt target files) using different partitioning heuristics. Scotch partitioning was tested with common examples. We have considered only shortest-path and avoided multi-path routing due to the high cost of packet reordering. Notice that Scotch can plot actual mapping data using 2-d color graphical representation; this enhances the automated task allocation phase with a nice, user-friendly GUI.

In Figure 4.8 we compare edge dilation for embedding the previously described master-slave tree-like benchmarks, i.e. single multi-rooted forests, multiple node-disjoint single-rooted trees and multiple node-disjoint multi-rooted forests, onto our candidate NoC topologies using the efficient default strategy in the Scotch partitioning tool; notice however that Scotch mapping is not always optimal, even if theoretically possible.

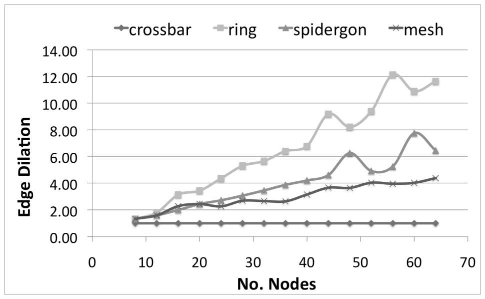
By examining these figures, we make the following remarks and comparisons.



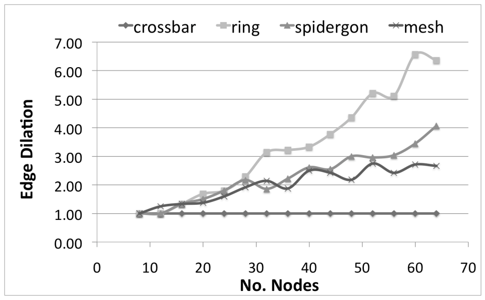
(a)



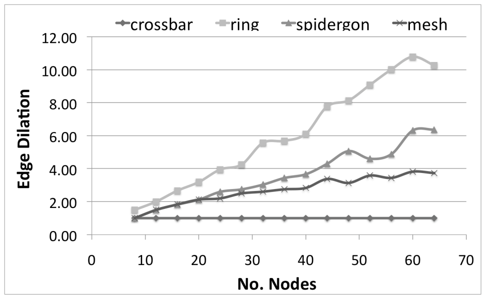
(b)



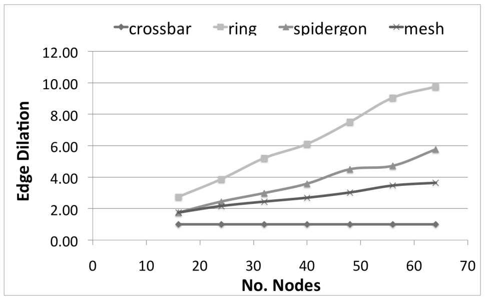
(c)



(d)



(e)



(f)

Figure 4.8: Edge Dilation for: (a) 2-rooted and (b) 4-rooted forest, (c) 2 node-disjoint and (d) 4 node-disjoint trees, (e) 2 node-disjoint 2-routed and (f) 4 node-disjoint 4-routed forests in function of the network size.

- Ring is the NoC topology with the largest edge dilation;
- For master-slave trees, Spidergon is competitive compared to 2-d Mesh for $N \leq 32$. Moreover, for node-disjoint trees or forests, Spidergon is competitive to Mesh for larger network sizes (e.g. up to 52 nodes), especially when the number of node-disjoint trees or forests increases, i.e. when the degree of

multiprogramming increases. This effect arises from the difficulty to realize several independent one-to-many or many-to-many communication patterns on constant degree topologies;

- Notice that Mesh deteriorates for 44 and 52 nodes due to its irregularity. This effect would be much more profound if we had considered network sizes that are multiples of 2 (instead of 4), especially sizes of 14, 22, 26,. . . 58 and 62 nodes.

Figure 4.9 shows our results for edge expansion normalized to the best result, obtained from embedding the Mpeg4 source graph onto candidate NoC topologies using the SCOTCH partitioning tool.

We notice that the Crossbar has the smallest edge expansion so this value is used as reference for the normalization. This is an expected result since in a Crossbar every node is connected to the others through a single channel. Spidergon and Mesh have a very similar edge expansion (where Spidergon has slightly better value) while the Ring topology has the highest value of all.

Figure 4.9: Edge expansion for 12-node Mpeg4 for different topologies (target graphs).

Finally the NoC mapping considered so far have been obtained in seconds on a PENTIUM IV with 2GB of Ram memory and running Linux.

4.5.5 Simulation Models for the Proposed NoC Topologies

In the NoC domain, IPs are usually connected to the underlying interconnect through a network interface (NI) which provides connection management and data packetization (and de-packetization) facilities.

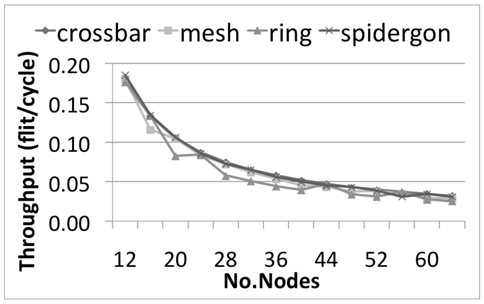
Each packet is split into data units called *flits* (flow control units). The size of buffer queues for channels is a multiple of the flit data unit, and packet forwarding is performed using flit-by-flit routing. The switching strategy adopted in our models is worm-hole routing. In *worm-hole*, the head flit of a packet is actively routed towards the destination by following forward indications of routers, while subsequent flits are passively switched by pre-configured switching functions to the output queue of the channel belonging to the path opened by the head flit. When buffer space is available on the input queue of the channel of the next switch in the path, a flit of the packet is forwarded.

In the NoC domain, flit-based worm-hole is generally preferred to virtual cut-through or packet-based circuit switching because its pipelined nature facilitates flow control and end-to-end performance, with small packet size overhead and buffer space. However, due to the distributed and finite buffer space and possible circular waiting, complex routing deadlock conditions may arise.

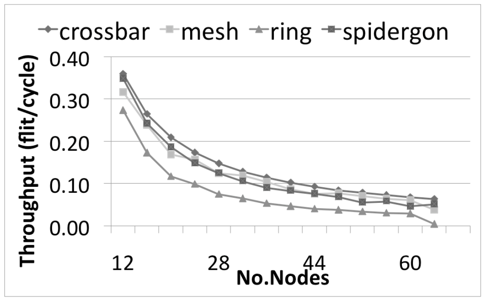
The considered Mesh architecture resolves this point through a deadlock avoiding routing algorithm called *Dimension Order* (or *XY algorithm*) that limits path selection [DYN03]. At first, flits are forwarded towards their destination initially along the X direction (the horizontal link) until the column of the target node is reached. Then, flits are forwarded along the Y direction (vertical link) up to the target node, usually asynchronously.

The bidirectional Ring architecture deals with message-dependent deadlock using virtual channels (VC) [Dal90]; this technique maximizes link utilization and allows for improved performance through smart static VC allocation or dynamic VC scheduling. VCs are implemented using multiple output queues and respective buffers for each physical link. A number of VC selection policies exists in the literature [MB06]; we adopt the *Winner Takes All* algorithm for VC selection and flit forwarding [DT04].

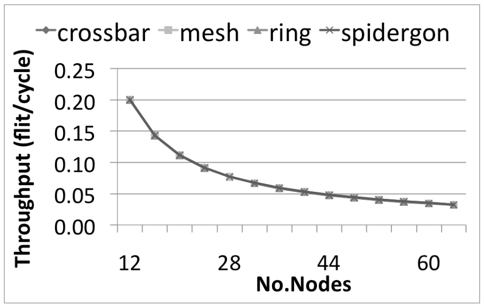
In this article, we also consider an unbuffered Crossbar. Each node in this Crossbar is directly connected to all others, i.e. without any intermediate nodes. Thus, we model an unbuffered (packet-switched) full Crossbar switch with round robin allocation of input channels to output ones. The key for this interconnect is channel arbitration. In particular, when a first flit is received, the arbiter checks if the requested output channel is free. If this is true, the input channel is associated



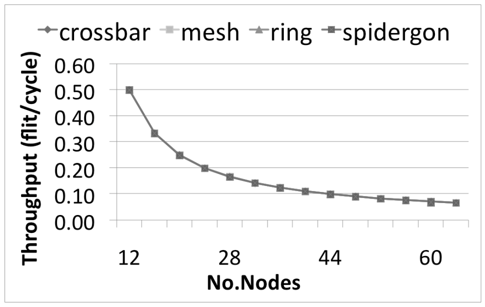
(a)



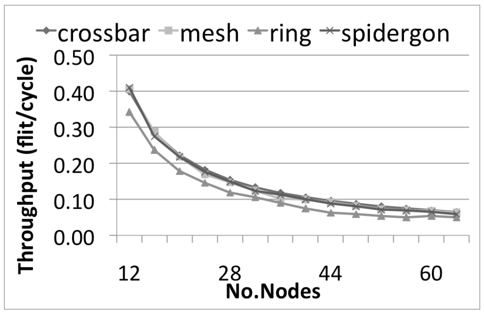
(b)



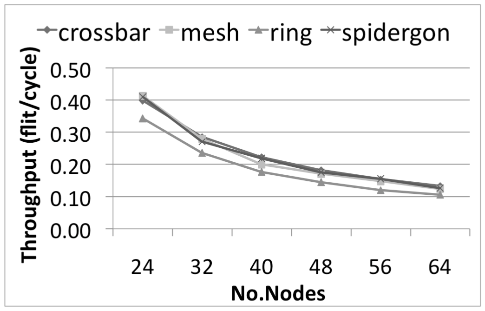
(c)



(d)



(e)



(f)

Figure 4.10: Maximum throughput as a function of the network size for: (a) 2-rooted forest (b) 4-rooted forest (SRF) , (c) 2-rooted tree (d) 4-rooted tree (SRT), (e) 2-node 2-rooted forest and (f) 4-node 2-rooted forest (MRF) and different NoC topologies.

to the output one until the whole packet is transmitted, otherwise the flit remains in the input register (blocking the relative input channel) until the arbiter assigns the requested channel.

Finally to avoid the protocol deadlock [SP03b, HG07] caused by the dependency between the targets input and output channels, we configured the network's router with two *Virtual Networks* (VN) [BCH95]: one for requests and one for replies packets. Flits to forward are selected from VNs in a round robin way, and the respective VC in ring is selected with the *Winner Takes All* algorithm [DT04], where flits of a single packet are sent until either the packet stalls or is completely transmitted.

In our experiments, all target input buffers and initiators' output buffers are assumed to be infinite: this allows us to focus on network performance by including deadlock avoidance schemes, but avoiding packet loss due to external devices from playing a bias role in network analysis. However, finite buffers can be treated using the same methodology.

We have modeled the Crossbar, Ring, Mesh and Spidergon architectures using a number of synchronous (shared clock) network routers, with each router connected to a network interface (NI) through which external IPs with compatible protocols can be connected [Hwa01].

In our model, depending on the simulated scenario, each IP acts either as a processing (PE), or as a storage element (SE). Traffic sources (called initiators) generate packet requests directed to target nodes (SEs) according to their configuration. All routers forward incoming flits according to the previously defined path computation algorithm, provided that the following router has enough room to store them. Otherwise, flits are temporarily stored in the channel output queues. Since Crossbar has no intermediate buffers, flits remain in the infinite output buffer of the initiator, until they can be injected into the network.

According to the application type (e.g. Mpeg4) storage elements receive request packets and generate instantaneously the respective reply packets to be forwarded to the initiator. All studied architectures have been modeled using similar routing techniques and PE/SE components are always adapted to specific architecture needs.

Figure 4.10 represents the average throughput of replies for all initiators. For each experiment, the offered load is the initiators' maximum injection rate. In the simulation test bench, requests and replies have the same packet length (5 flits), while for each request corresponds exactly one reply.

Due to traffic uniformity, the reply throughput at each initiator increases when augmenting the real injection rate, until the node saturates. After this point, the router is insensitive to the offered load, but continues to work at the maximum possible rate. Thus, the throughput remains constant (at a maximum point), while the initiators' output queue sizes (assumed as infinite) actually diverge to infinity very quickly.

By examining the graphs in Figure 4.10 we draw the following observations:

- As expected, the Ring performs generally worse than all studied NoC topologies;

- The Spidergon behaves better than Mesh for small networks (up to 16 nodes) and remains competitive for larger network sizes in all considered traffic patterns. However, notice that SCOTCH considers all minimal paths between any two nodes, while the OMNeT++ model uses only the subset of minimal paths defined by the XY routing algorithm. Use of a specific routing algorithm with the SCOTCH mapping tool is an interesting future task;

- For the 4-rooted forest, Mesh outperforms Spidergon in larger networks only for regular Mesh shapes (32, 36, 48, 52 nodes);

- Under 2 and 4 node-disjoint tree patterns, all considered architectures saturate at the same point;

- In the 2- and 4-rooted tree cases, considering the total amount of injected flits per cycle generated by all initiators, we obtain two constant values: 2 and 4 flit/cycle which is exactly what the two and four storage elements can absorb from the network. In this case, the bottleneck is given by the SEs and not by the architectures which operates under the saturation threshold;

- Crossbar has the best performance in all studied cases, with a smooth and seamless decreasing throughput.

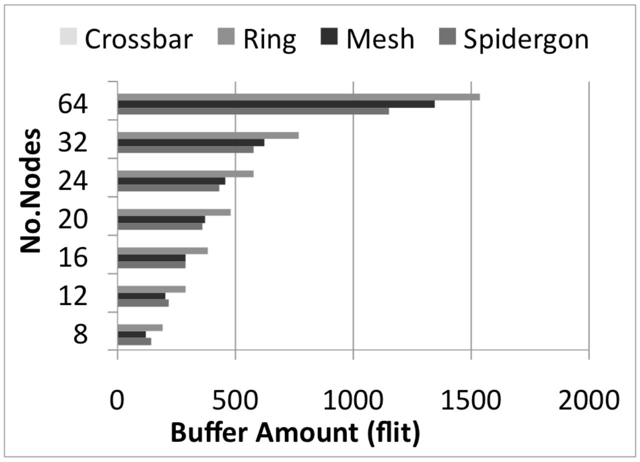


Figure 4.11: Amount of memory required by each interconnect.

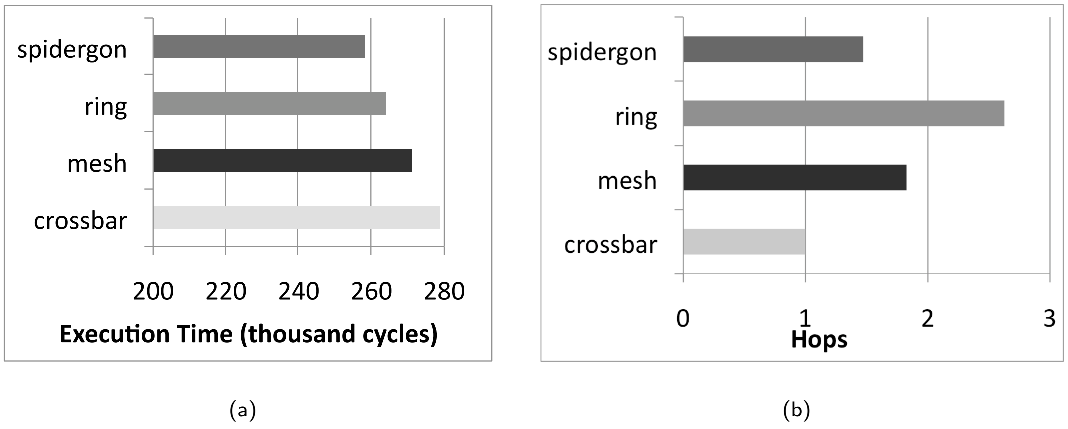


Figure 4.12: Task execution time (a) and average path length (b) for Mpeg4 traffic on the considered NoC architectures.

4.5.6 Mpeg4 a Realistic Scenario

In addition to the previous synthetic task graph embedding scenarios, we examined performance of bidirectional Ring, Mesh, Spidergon and unbuffered Crossbar architectures for a real Mpeg4 application modeled by using the task graph illustrated in Figure 4.6. In order to compare these topologies, we set up a transfer-speed test where all architectures are mandated to transfer a fixed amount of Mpeg4 packets. Initiators generate requests for SEs (according to the task graph in Figure 4.6), and SEs reply with an instantaneous response message for each received request. Requests and replies have have a length of 4 flits. In addition, notice that some PEs have a generation rate that heavily differs from others.

In our modeling approach, we have chosen to assign to each intermediate buffer a constant size of three flits. As shown in Figure 4.11, the buffer memory in Mesh and Spidergon is comparable (and lower than ring), and Spidergon buffer allocation becomes lower than Mesh when the network size increases. The XY routing algorithm used in the Meshe NoC, and Across-first routing used in Spidergon have the advantage of avoiding deadlock without requiring virtual channels. Ring topology uses two virtual channels for each physical channel in the circular links, to avoid deadlock hence Ring has an high need of buffer space. Notice that the Crossbar architecture does not use network buffering hence in Figure 4.11 its columns are always zero. The Crossbar in fact uses buffering only on the Network Interface, and as for the other architectures, this buffering is not considered in the computation.

We analyzed the application delay measured as the number of elapsed network cycles from the injection of the first request packet of the load to the delivery of the last reply packet of the same load. In our simulator, the packet size is measured in flits; this unit relaxes the need to know the actual bit-size of a flit.

Furthermore, since we focus on topological constraints rather than real system dimensioning, we assume that each channel is able to transmit one flit per clock cycle. As proposed in [KKS05b], in order to define a flit injection rate for each different PE of the Mpeg4 task graph, in the transfer speed test we use as reference the highest demanding PE (called UPS-AMP device, see Figure 4.6). All remaining nodes inject flits in a proportional rate with respect to the UPS-AMP device. These rates are reported in tabular form in Table 4.1.

From Figure 4.12 (a), we observe that Ring and Spidergon have the best performance while, quite surprisingly, Mesh and Crossbar perform worse than expected. The explanation can be obtained by considering the allocated buffer size for the 12-node architectures shown in Figure 4.11: Mesh and unbuffered Crossbar have less buffer memory, i.e. 204 flits for Mesh and 0 for Crossbar vs. 288 flits for Ring and 216 for Spidergon. To summarize results, by computing the percentage

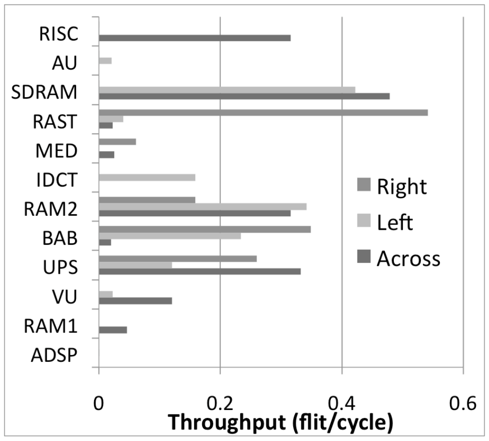
	Offered Load (Mb/sec)	**% w.r. UPS**	**% w.r. Tot**
VU	190.0	12.03	5.48%
AU	0.5	0.03	0.01%
MED	100.0	6.33	2.98%
RAST	640.0	40.51	18.47%
IDCT	250.0	15.82	7.21%
ADSP	0.5	0.03	0.01%
UPS	1580	100.0	45.59%
BAB	205.0	12.97	5.91%
RISC	500.0	31.65	14.43%
TOT	3466.0	219.37	100.00%

Table 4.1: Initiators average injection rate and relative ratio with respect to the UPS-AMP node.

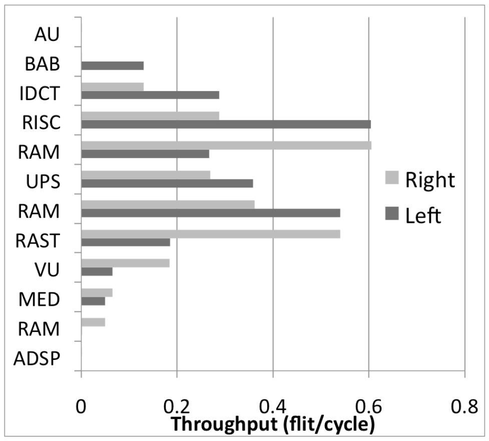
difference between the data transfer performance reported in the first histogram of Figure 4.12 (a), we conclude that for near optimal Mpeg4 mapping scenarios, Spidergon is faster than Ring by 0.6%, faster than Mesh by 3.3% and faster than unbuffered Crossbar by 3.2%. Next, we obtain more detailed insights on the steady state performance metrics and resource utilization of the proposed architectures, for the considered scenarios.

In the second histogram of Figure 4.12 (b) we illustrate the average path length of flits (and its standard deviation) obtained with the Mpeg4 mapping in the data transfer experiments. Ring forces some packets to follow longer paths than other topologies, but in this way it effectively uses its buffer space more efficiently. Except for unbuffered Crossbar (which saturates soon). Spidergon provides a good tradeoff among proposed topologies, resulting in shorter and more uniform paths.

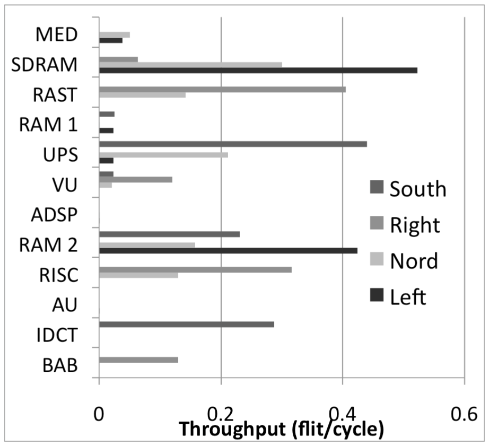
In the analysis of node throughput reported in Figure 4.13, we observe that in all topologies the most congested links are those connected to the busiest nodes (SDRAM, UPS-AMP, and RAST of Figure 4.6). Despite the higher number of



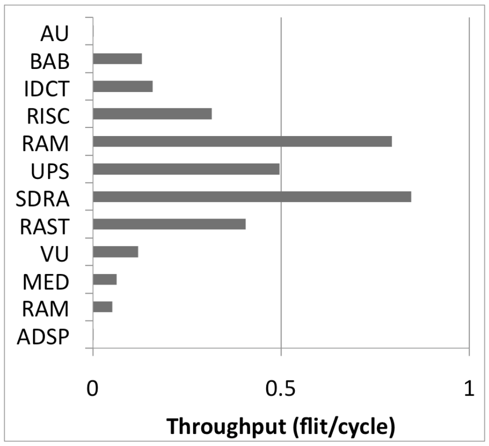
(a)



(b)



(c)

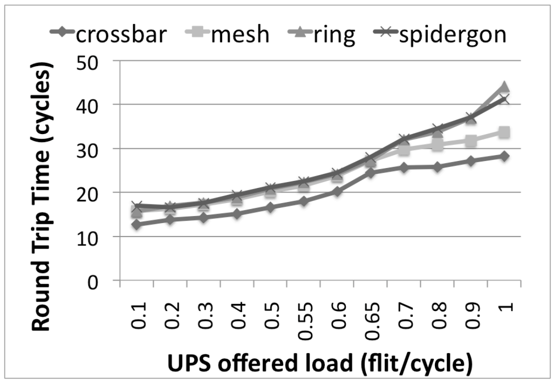


(d)

Figure 4.13: Average throughput on router's output port for (a) Spidergon, (b) Ring, (c) Mesh and (d) unbuffered Crossbar architecture.

channels that the Mesh disposes, Spidergon and Mesh actually forward packets along the same number of links. The Mesh XY routing algorithm does not exploit all paths this architecture provides, while Spidergon provides better channel balancing. Because of its shape, the Ring exploits much more its channels. In the Crossbar the busiest channels are those toward the SDRAM and SRAM2 nodes, the two veritable network hot spots, and the UPS-AMP node which generates more than the 45% of the network traffic.

The absence of intermediate buffers makes the Crossbar architecture very sen-



(a)

Figure 4.14: Network RTT in function of Initiators' offered load.

sitive to realistic unbalanced traffic. In particular, Crossbars may show end-to-end source blocking behavior since a packet addressed to SDRAM may have to wait in the output queue of the initiator, while buffered multi-hop paths could allow initiators to inject more packets into the network (if buffer space is available in the path), thus facilitating an emptying behavior of source-queue packets addressed to different targets.

Figure 4.14 shows the average network round trip time (RTT), i.e. the average time required for sending a request packet and obtaining its respective reply packet from the network (only network time is computed, i.e. the time in the infinite queue of the initiator is excluded). Note that in the following figures, the UPS-AMP node injection rate is taken as reference and reported on the X axis, while the injection rate for other nodes can be computed proportionally following Table 4.1. The average injection rate of the initiators (total offered load) can be obtained by multiplying this value by a constant factor (percentage of total initiator load) which is 2.1937 for the Mpeg4 scenario. For all the topologies, the RTT time slowly increases until congestion starts (rate below 0.6flits/cycle).

The UPS-AMP network congestion appears for a UPS-AMP injection rate between 0.6 and 0.7 flits/cycle. When the path used by the UPS-AMP saturates and becomes insensitive to the offered load (around 0.7 flits/cycle), other initiators using different paths may still augment their input ratio, increasing network congestion

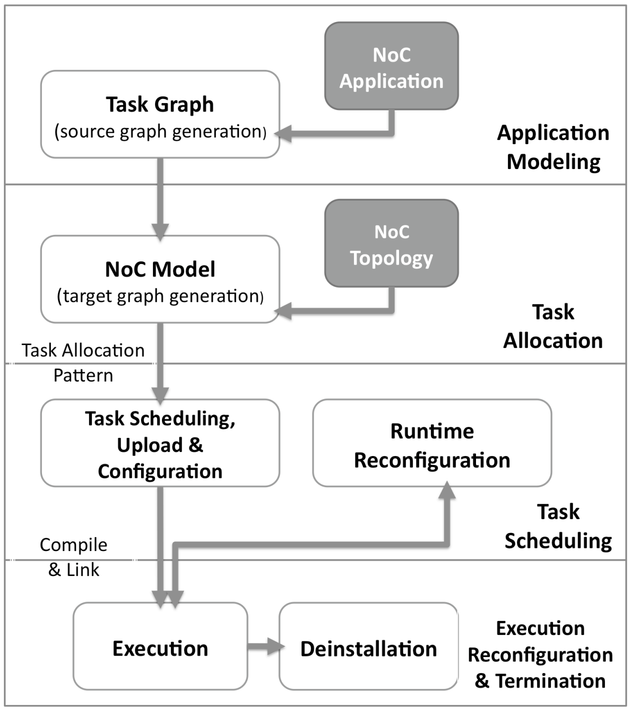


Figure 4.15: Future work:dynamic scheduling of tasks.

and average network RTT. Crossbar has the lowest RTT thanks to the absence of intermediate hops. Spidergon has an average RTT similar to Mesh and Ring while having shorter paths. This indicates that Spidergon channel buffers are in general better exploited.

4.6 Conclusions and Extensions

In this article we presented the methodology and tools we use to perform high-level analysis of NoC architectures.

In particular we presented TGFF , an open-source tool that allows to generate complex application task graph. Then we discussed about SCOTCH , an embedding and partitioning tool used to map the generated task graph onto a selected NoC.

Finally, we presented OMNET++ and out cycle-accurate high-level NoC simulator. Here we discussed a case-study considering three theoretical task graphs and a real Mpeg4 decoder.

Future related work will focus on improving:

- the SCOTCH partitioning tool to consider also non-minimal paths or implement complex cost functions, e.g. for optimizing system power consumption by minimizing total edge dilation;
- our OMNeT++ model to consider runtime task migration and reconfiguration, or optimize buffer size at the routers or at the incoming or outgoing network interface.

Part III

Transport Protocols for NoC

Chapter 5

Data Transfer Protocols

5.1 Introduction

Networks-on-Chip (NoC) have been proposed to address the increasing impact of communication on multi-core systems-on-chip (SoC) [BM02, DT01, HJK$^+$00]. With the NoC paradigm various on-chip cores (processors, memories...) exchange data by accessing a network of optimized links and routers through network interfaces (NI) as shown in Fig. 5.1. The NIs decouple the design of the cores from the design of the network, implement the NoC communication protocols, and improve performance by providing elasticity between inter-core communication tasks and intra-core computation tasks thanks to their storage capabilities: as shown in Fig. 5.1 input and output queues are used to temporarily store the incoming and outgoing messages. While messages are the units of transfer from the network clients (processors and memories) to the networks, in the network interface a single message is typically broken down into a sequence of small packets for routing purposes; packets may be further segmented in flow control digits (flit) for more efficient allocation of network resources such as link bandwidths and queue capacities [DT04, DYN03].

The correct operations of a network requires to efficiently handle deadlock situations which may arise due to the circular dependencies on the network resources that are generated by in-flight messages. A variety of methods has been proposed in the literature to either avoid or recover from deadlock [DT04, AP95]. Most of

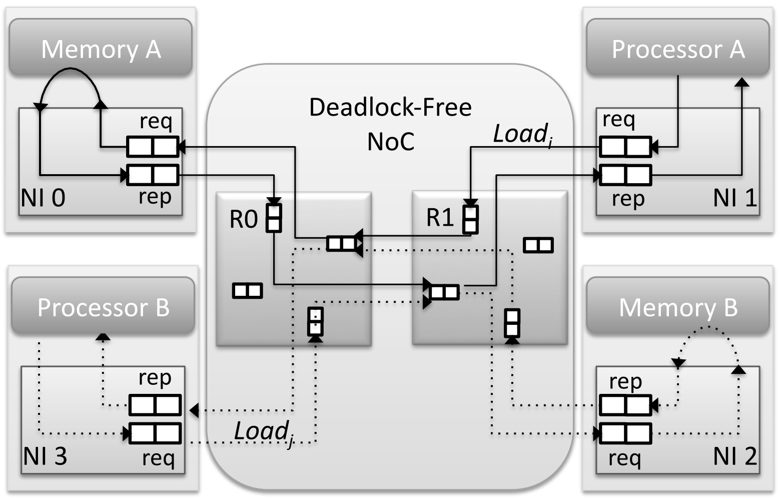


Figure 5.1: Message-dependent deadlock in a shared-memory request-response paradigm.

these protocols assume the *consumption assumption* where the packets of a message traversing the network are always consumed by the destination core once they reach its corresponding network interface [SP03a]. However, deadlock may be caused also by dependencies that are *external* to the network, i.e. dependencies that are *internal* to a core. In fact, in real SoC systems and multiprocessor systems a core typically generates new messages in response to the reception of a previous message. These dependencies between messages can generate a different type of deadlock that is commonly referred as *message-dependent* (or *protocol*) deadlock [D.K97,SP03a,HGR07]. Message-dependent deadlock occurs at a level of abstraction that is higher than the routing-level deadlock, which is addressed by deadlock-free routing algorithms such as dimension-order routing [DT04, DYN03]. [1]

Figure 5.1 shows a simple example of a message-dependent deadlock that may occur due to the dependence between the messages that are received by (sent from) a memory core. The network interface NI 0 receives packets for a memory load

[1]We focus on addressing message-dependent deadlock while assuming the use of a deadlock-free routing algorithm. Notice that message-dependent deadlock is different from *application-level deadlock* which is out of the scope of this paper.

(or store) request message addressed to Memory A and in reply sends packets with a response message that includes the requested data (or the acknowledgment of a store operation). Since the input and output queues of NI 0 have necessarily limited storage capacity, a long sequence of requests may cause a backpressure effect into the NoC. For instance, the packets of a series of load request messages $Load_i$ from Processor A may not be fully stored within NI 0 and, instead, may have to wait for several clock cycles in the East queue of Router 0. Then, let's assume that Processor B sends a series of load request messages $Load_j$ to Memory B. Even if Memory B can immediately serve a first subset of these requests, the packets of the corresponding response messages will not be able to reach Processor B because they will be blocked as they attempt to access the East Queue of Router 0. On the other hand, when Memory A will be finally able to serve the request messages $Load_i$, the packets of its response messages will not be able to reach Processor A because they will be blocked as they attempt to access the West Queue of Router 1, which are occupied by some of the packets of the load request messages $Load_j$. In summary, even if the NoC uses a deadlock-free routing algorithm, the dependencies across the messages inside the memory cores cause a circular dependency involving NI 0, Router 0, Router 1, and NI 1 which leads to a deadlock.

Related Work. Various solutions for message-dependent deadlock have been proposed in the literature. Dielissen *et al.* solve this problem by guaranteeing sufficient storage space for each possible pair of communicating elements [DRKR03]. Anjan *et al.*, instead, add timers into the router's output ports to detect deadlock occurrences and move the blocked packets into specialized queues to guarantee progress [AP95]. Song *et al.* propose a protocol-recovery protocol motivated by the observation that in practice message-dependent deadlocks occur very infrequently even when network resources are scarce [SP03a]. These three approaches, however, are meant for parallel computing systems and are not expected to scale well to NoC applications.

The message-dependent deadlock problem in NoC for shared-memory architectures has been addressed by introducing two physically-separated networks for the two message types (load and store requests) [MM05] or two logically-separated network (virtual networks) [CGL+08]. These solutions may be difficult to scale to future multicore SoCs where the increasing number of heterogeneous cores and message types is likely to grow, thus leading to more complex dependencies among packets.

The ÆTHEREAL [GRK+05] and FAUST [DBL05] NoCs use *credit-based (CB) end-to-end flow control protocols*. Similar to the credit-based flow control mechanisms that operate at the link level between a pair of interconnected routers [DT04, DYN03], a CB end-to-end flow control protocol uses credits to inform a sender NI about the current storage capacity of the queue in the receiving NI. As discussed in Section 5.3, the sender NI keeps track of this capacity with a credit counter that is initialized with a value equal to the size of the corresponding queue and is dynamically updated to track the number of available packet slots in the queue. Hence, the sender continuously transmits only a subset of the message packets that is guaranteed to eventually arrive inside the NI, thus avoiding a message-dependent deadlock. Notice that for a given SoC a core that may send messages to N different cores needs N credit counters while if it can receive messages from M different cores it needs M different queues.

Contributions. We build on the CB approach to develop *Connection then Credits* (CTC), an end-to-end flow control protocol that allow us to handle the message-dependent deadlock while simplifying the design of the network interface, which is based on the same micro-architecture regardless of the number of communications that its core may require. This micro-architecture uses a single credit counter together with an output queue for sending all the possible outgoing messages and a single pair of data-request queues that is shared across all possible incoming messages. On the other hand, as explained in Section 5.4, CTC requires the completion of a handshake procedure between any pair of cores that want to communicate before the actual message transfer starts. This procedure is used to initialize the credit counter in the sender NI based on the current available space in the data queue of the receiver NI. While this necessarily adds a latency overhead to the transfer of

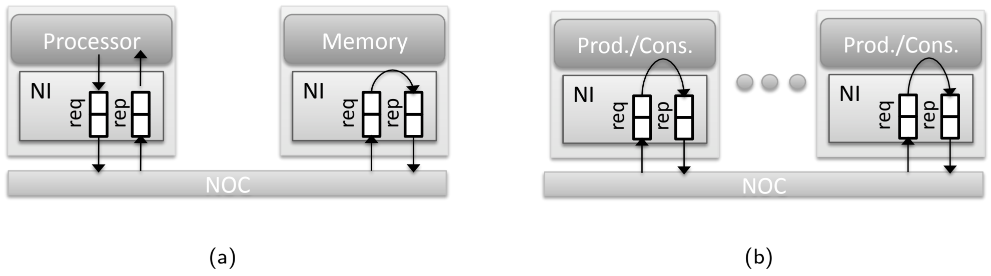


Figure 5.2: Message dependency in shared memory (a) and message passing (b) communication paradigms.

the message, the penalty in performance is limited when large messages need to be transferred as it is shown by the simulation results that we report in Section 5.5.

5.2 Message-Dependent Deadlock

There are two main communication paradigms for exchanging data among the processing cores of a system-on-chip and they are associated to two corresponding programming models: *shared memory* and *message passing*.

In a shared-memory paradigm the processing cores communicate via data variables that are defined in the same logical memory space and are physically stored in one or more memory cores. As shown in Figure 5.2(a), a processor accesses a memory through either a load or a store request by specifying the memory address and the size of the data block to be transferred. In the case of a load request the addressed memory replies by sending the values of the requested block of data (typically a cache line) to the processor, which saves them in its local cache memory. In the case of a store request the memory receives new values for a block of addresses, which typically correspond to a line in the processor's local cache, and it replies by generating a short ACK message to confirm their correct delivery. Shared memory is the most used paradigm in current multi-core SoCs.

In the message passing paradigm, which is illustrated in Figure 5.2 (b), the processing cores communicate by sending/receiving data that are pushed directly from a core to another (*peer-to-peer* communication): the sending and receiving cores are commonly referred as the *producer* and *consumer*, respectively. By having dedicated logical addressing space for each processing core and providing direct communication among their physical local memories, message passing avoids the issues of shared-memory coherency and consistency [HP06], thus potentially reducing the communication latency of each data transfer. This paradigm is particularly suited for data-flow and stream processing applications that consist of chains of processing cores such as the video processing pipeline [HGR07].

The correct implementation of shared memory and message passing paradigms in a system-on-chip require an underlying NoC with communication protocols that guarantee the correct transfer of each message and, particularly, the absence of deadlocks. As discussed in the Introduction, even if the NoC relies on deadlock-free routing algorithms, message-dependent deadlock may arise due the dependencies among the messages "inside a core", which are shown in Figure 5.2: e.g. the dependence between a load request and response in a memory for the shared memory paradigm and the causality dependency between the consumption and production of data in a core for the message passing paradigm. For both paradigms, the dependencies between pairs of messages may get combined, thus leading to *message dependency chains* [PS01]. Indeed, the causality relations among pairs of messages can be modeled as a partial order relation $\prec$ over the set of all possible messages that are transferred in the network. Message dependency chains depend on the chosen communication paradigm and the characteristic of the given application [HGR07].

As for routing-dependent deadlock, the message-dependent deadlock problem can be addressed with either avoidance or recovery strategies. The relative advantages of the various techniques based on these two approaches depend on how frequently deadlocks occur and how efficiently (in terms of resource cost and utilization) messages can be routed while guarding against deadlocks [SP03a].

The introduction of a *Virtual Network* (VN) for each type of message transfer guarantees the solution of the message-dependent deadlock by satisfying the consumption assumption [SP03a, CGL+08]: the input and output queue of each router and each NI in the network is replicated and assigned to a single specific message class (e.g. two classes in case of memory request and response messages). This solution "cuts" the causality dependency between messages in the network at the cost of a higher buffer requirement and more complex router and NI design.

Stream processing applications implemented with a pipeline of processing cores, where each core produces data for the next consumer core, lead to a dependency chain of message requests $request_1 \prec \cdots \prec request_n$ where n is the number of cores in the pipeline. For example, Figure 5.3 shows the task graph of the Video Object Plane Decoder (VOPD) application that can be implemented by mapping each of the tasks on a distinct processing core. The resulting pipeline has 12 stages and, therefore, it leads to 12 different types of request messages if all the communications are implemented by peer-to-peer message passing. Multi-core SoCs for embedded products simultaneously support an increasing number of applications such as the VOPD. This translates into the presence of complex communication patterns among the cores, which simultaneously run multiple threads of computation to implement the multiple tasks of the various applications. The implementation of the communication requirements among these cores with a NoC requires new solutions for the message-dependent protocol. In fact, a solution based on virtual networks does not scale as the number of distinct message types that travel on the network continues to grow. Furthermore, the length of the dependency chains depends on the given application and is difficult to predict. Similarly, as the number of processing and memory cores continues to grow also for multi-core SoCs that are based on a shared-memory architecture, the implementation of cache-coherent protocols becomes harder and scaling the number of virtual channels becomes impractical.

5.3 Credit Based (CB) Protocol

A different approach to the solution of the message-dependent deadlock is based on the use of an *end-to-end* flow control protocol that guarantees that a sender NI does not ever inject more packets in the network than the number of packets that the corresponding receiver NI can eject. The Credit Based (CB) end-to-end flow

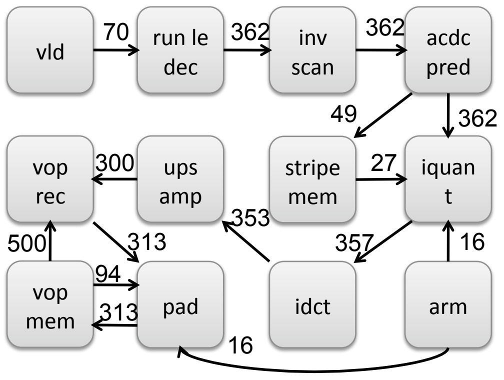


Figure 5.3: The MP Video Object Plane Decoder (VOPD) task graph.

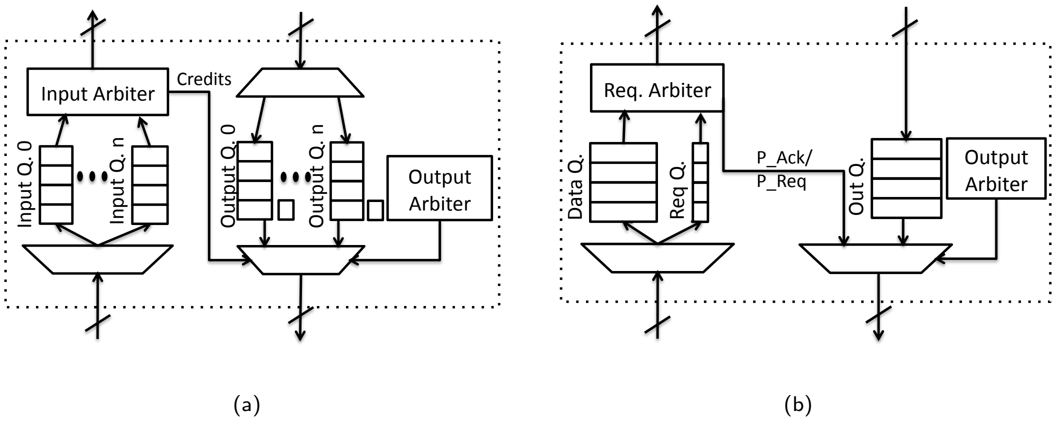


Figure 5.4: Network Interface implementations: (a) credit based and (b) CTC.

control protocol is a simple implementation of this idea that is used in [DBL05] and [GRK+05]. With a CB protocol, the sender NI maintains a detailed knowledge of the number of packet slots that the receiver NI has still available through the exchange of peer-to-peer transmission credits. A credit can be associated to either a packet or to a packet flit depending on the desired level of granularity. What is important is the guarantee that no fragment of a message can remain blocked in the network due to the lack of space in the NI input queue, with the potential risk of causing a deadlock situation. Hence, the sender NI can continue to inject flits in the network only if it has still enough credits as proofs that the receiver NI will eject these flits. Dually, the receiver NI must send a credit back to the sender NI for each flit that its core has consumed, thereby generating an empty slot in its input queue.

Generally a single consumer core can be addressed by multiple producers. Also a producer can address multiple consumers and for each of these the producer needs a separated credit counter. In fact, differently from credit-based flow control mechanisms that operate at the link level between a pair of interconnected routers [DT04, DYN03], here peer cores may be separated by multiple hops in the network. Also all packets generated by the peer cores arrive at the same NI's input port. Figure 5.4 (a) shows the simplest way to address this issue. Each NI is provided with multiple and independent input and output queues and credit counters: one for each possible sender NI (input queue) and receiver NI (output queue and credit counter) that may communicate with this NI. A generic NI n_d, upon the reception of a flit from NI n_s, saves the incoming data into the s^{th} input queue. When a flit is read from the s^{th} input queue, an end-to-end credit is sent back to n_s. In turn, n_s updates the d^{th} credit counter upon the reception of a credit.

When a NI forwards a message to the connected core it generally removes a multiple flits from its input queue. Hence during a single clock period multiple credits may be generated. To avoid this issue a single end-to-end credit message can convey more that one single credit per packet. The amount of credits K associated to a single credit-message is a fixed parameter of the system. Note that the size Q_{in} of each input queue must be set accordingly to K. In particular considering the set of peers $P_c = \{n_i \dots n_j\}$ that can possibly communicate with n_c, the n_c's input queues should be sized as:

$$Q_c = Max(K + RTT(n_c, n_i), i \in P_c) \tag{5.1}$$

where RTT is the round-trip time function, which depends on the distance between the NIs.

The choice of which queue must be used is made by the *Input Arbiter*. On the output side, instead, the *Output Arbiter* selects the queue that is used to forward the flit or to send a credit. The selection of the input and output queues is made on packet basis to avoid the delivering/reception of flits of different packets, e.g. according to a round robin policy.

Note that the CB end-to-end flow control protocol differs from the virtual network approach for a number of reasons: first in VN all the queues, including those in the routers must be replicated while in the CB protocol only the queues of the NI must be replicated. Moreover using VN the number of queues per channel depends on the message-dependencies that, in turn, depend on the given application. Instead, for the CB protocol this number varies with the number of producer cores addressing each single consumer core.

5.4 Connection Then Credits (CTC) Protocol

Adding a dedicated input and output queue for each possible source/destination of messages, as required by the CB flow control protocol, forces engineers to design a specific network interface and relative arbiters for each node of the network. This is the case particularly for multi-mode SoCs where the same core can be addressed by different other cores depending on the mode selected by the user.

As an alternative to CB, we present the Connection Then Credits (CTC) flow control protocol. CTC rationalizes and simplifies the design of NIs while guaranteeing the absence of message-dependent deadlock.

CTC regulates the exchange of messages between two peer NIs by first introducing a handshake procedure called *Connection*. A CTC-message is a fixed amount of data to be exchanged between the two NIs. As shown in Figure 5.4 (b) a CTC NI is composed by only *two* input queues and *one* single output queue independently from the number of possible peers that can require a connection with this NI. The first queue, called the *data-queue* is used for storing incoming data flits. The *request-queue* instead is used for the incoming transactions requests. When a producer NI n_s needs to initiate a connection towards a consumer peer NI n_d, it first sends a request packet[2] called P_REQ (packet-request) to n_d to signal its request to communicate. A P_REQ packet also indicates the total size of the message

[2]Note that P_REQ and P_ACK are actually *messages* composed by one single packet. When referring to these two messages we use the two words without distinction.

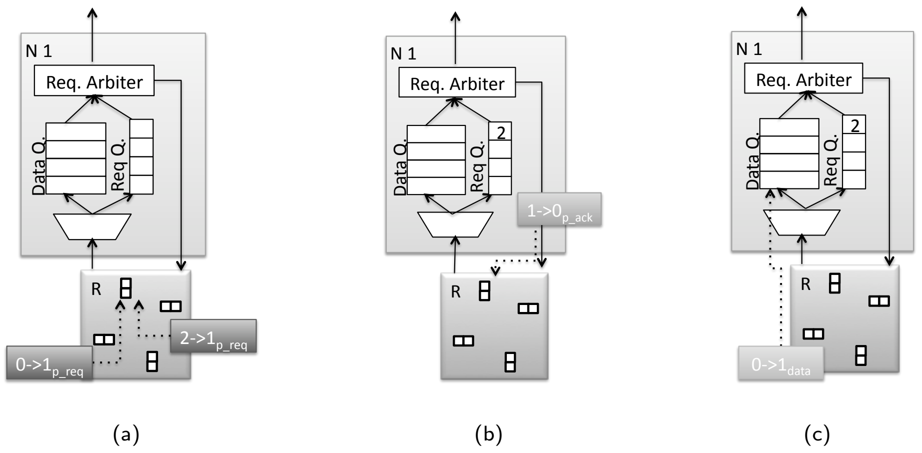


Figure 5.5: The CTC transaction-definition procedure: (a) a the consumer NI receives multiple P_REQ, (b) the consumer selects one requests and generates the relative P_ACK, (c) the selected producer NI starts sending the flits.

to be delivered and some additional information that can be used by the NI (i.e. for priority decisions). Upon the reception of a P_REQ, n_d stores the request in the request-queue together with the other requests previously received and not yet processed. When the core associated to n_d is available for processing a new request (i.e., the data queue has enough free space to accept a new message) it generates an acknowledge packet called P_ACK that is forwarded to the source of the given request. A P_ACK is similar to the credit packet in the CB flow control. The difference is that the first P_ACK sent by n_d actually *initializes* the output credit counter of n_s so that it can generate and send a specific amount of data. Upon the reception of credits the producer first generates a *header* flit used to open a path along the routers of the NoC, then it forwards the data flits and decreases the CTC-counter by one unit for each sent data-flit.

Figure 5.5 shows an example of the CTC protocol operations: at first two producer NIs, n_0 and n_2, address the consumer NI n_1 with two P_REQ messages indicating the size of the transaction they want to initiate. In Figure 5.5 (b) n_1 selects the peer n_0 to initiate the connections while it stores the other request in the request-queue. Then, N_1 generates a P_ACK message to initialize n_0's credit counter. Finally, Figure 5.5 (c) shows the data-packet generated by n_0 that reach the data-queue of n_1.

The NI frees a number of flits in one single clock whenever it forwards a message to the connected core. Hence, as for the CB case, a single P_Ack conveys K credits per single message. Differently from the CB flow control, however, once a P_Req is accepted, the consumer NI generates a chain of consecutive P_Ack packets so that the producer's credit counter is initialized with the maximum amount of credits that its consumer can offer. Instead, recall that in the CB protocol the credit counters are initialized at start up time. In both end-to-end flow controls each time the consumer frees K slots in the input data-queue, it continues to generate a new P_Ack message until the sent credits are sufficient to store all the flits of the requested transition (whose size was specified in the P_Req message). Finally, to avoid throughput degradation, data-input queue should be sized accordingly to Equation 5.1 as function of the number of credits per P_Ack and the maximum round trip time between the consumer NI and the producer addressing it.

To guarantee the consumption assumption of P_Req messages, the request-queue must be sized accordingly to the maximum number of requests that a NI can receive. For this reason each CTC producer is limited to have one single outstanding P_Req at time. Hence the length of the request-queue must be equal to the number of producers addressing the given NI.

CTC defines three message dependencies:

P_Req→ P_Ack: the request-queue is sized accordingly to the number of possible producer-peers addressing the given NI. CTC limits each node to have at most one outstanding P_Req at time. Hence the consumption of all injected P_Req is guaranteed.

P_Ack→data: P_Ack packets are always consumed by a network interface that updates the output credit counter and then deletes them.

data→ P_Ack: the credit mechanism ensures that no more data-flits than those allowed by the output credit counter can ever been injected. Hence all data

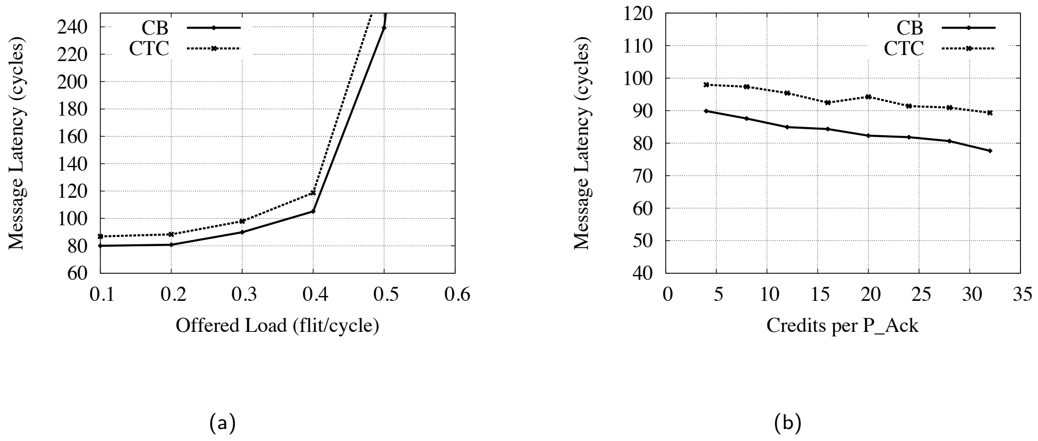


Figure 5.6: Message latency as function of (a) the injection rate and (b) the number of credits associated to a P_Ack packet.

flits injected in the NoC will, eventually, be consumed by their addressed NI.

Thanks to the CTC protocol design, these three dependencies are independent from the applications that is run by the cores, which, therefore, can be considered protocol-deadlock free.

5.5 Analysis and Simulation

To analyze the characteristics of the CTC protocol and compare the performance of a CTC-based NoC versus a CB-based NoC we developed a *C++* system-level simulator that allows us to model various NoC topologies, routing and flow-control protocols as well as the traffic scenarios injected by various types of cores. We use the simulator to compare the two end-to-end protocols for the case of the Vopd application whose task graph is shown in Figure 5.3.

In the CB-based NoC the NI of each core has a number of input queues equal to the number of incoming streams (see Figure 5.3). For both the CTC-based and the CB-based NoC, the size of each data input queue is set uniformly based on Equation 5.1 and no virtual channels are used.

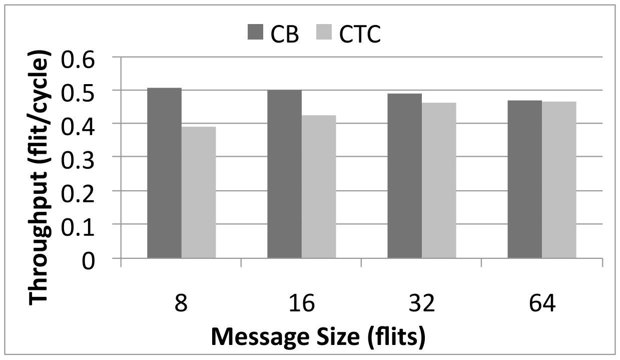


Figure 5.7: NoC Throughput as function of the message size when $K = 32$.

Figure 5.6 (a) shows the average peer-to-peer message latency as function of the average offered load when K is fixed to 32 credits (results are similar for the other credits values). As expected, the CTC protocol gives a higher latency due to the handshake procedure. Nevertheless, the difference between the two protocols remains under 10% up to the saturation point, which is around 0.4.

Figure 5.6 (b) shows the average peer-to-peer latency as function of the number of credits K per P_Ack packet when the offered load is lower than the saturation threshold. Clearly, by increasing the value of K the performance of the system also improves: PEs can inject more flits per P_Ack thus reducing the number of control packets (credits and headers). Conversely, increasing K also requires bigger input queues that must support the additional amount of flits received per P_Ack sent.

Figure 5.7 reports the throughput comparison as function of the message size. As expected, the performance of the CTC-based NoC increases with the messages size because it reduces the rate of connections-per-flits that must be set up. The throughput of the CB-based NoC, instead, decreases as the size of each message increases because this effectively augments the number of times in which a P_Ack packet preempts the data packet. Therefore, CTC represents a valid proposition for message-passing applications such as video stream processing that present large inter-core message transfers.

Finally, we analyze the amount of storage used by the two alternative flow control protocols. As discussed in Section 5.3, for avoiding throughput degradation, both

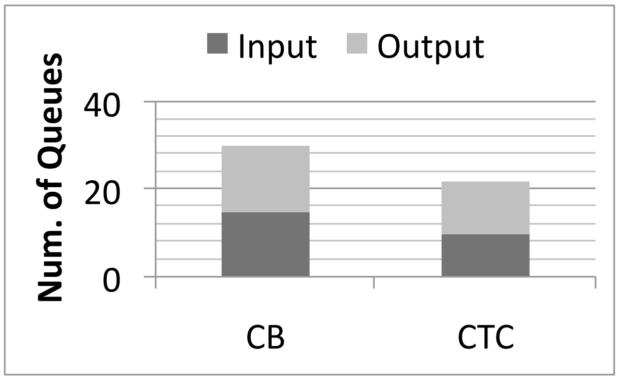


Figure 5.8: Breakdown of the aggregate number of input and output queues in the NoC NIs for the VOPD application.

CB-based and CTC-based NoCs need to size their input data-queues accordingly to the maximum round-trip time between the two communicating cores. For a CB-based NoC each input queue must be sized accordingly to Equation 5.1. For a CTC-based NoC, instead, only the data-queue must have this size while the request-queue must be as large as the *number* of distinct producer cores that can send message to its core. Notice that in order to provide a single interface design for each possible core, this number can be over-estimated without a major loss of area because the request-queue has a negligible size compared to the data-queue. Figure 5.8 shows the breakdown of the aggregate number of data-queues used in the network interfaces for the two approaches to support the VOPD application (where only the nodes with incident arrows are actually instantiated with input queues). The CTC-based NoC uses a total of 22 data queues, including both input and output, while the CB-based NoC needs 30 data queues. Assuming that the length of each data queue is the same in the two NoCs, CTC allows to save up to 35% of storage space for this particular case study. This translates directly in a reduction in area occupation and is expected to lead also to a reduction in overall NoC power dissipation.

5.6 Conclusions

Message-dependent deadlock is a destructive event that, even if rare [SP03a], must be properly addressed to guarantee the correct behavior of a network. The credit based (CB) end-to-end flow control protocol solves this problem by using multiple dedicated input queues and output registers in the network interfaces. This increases the complexity of the network interface design. Further, since the number of these queues depends on the number of distinct communications that its particular core may have, the same network may present interfaces that have different micro-architectural structures.

We proposed the Connection Then Credits (CTC) end-to-end flow control protocol as an area-efficient solution to the message-dependent problem that is characterized by a simpler and more modular network interface architecture. CTC-supporting network interfaces use one single input data queue and one output credit counter. Hence, the overall number of queues per network interface remains equal to two, the total amount of storage is reduced and the overall network-interface design becomes independent from the communication requirement of the particular core, thus increasing its reusability. On the other hand, any new communication between a pair of peer nodes requires the preliminary completion of a handshake procedure to initialize the output credit counter on the producer side (after the connection has been established CTC works in a way similar to the original Credit Based flow protocol). This procedure necessarily increases the latency of a message transfer and it also reduces the network throughput for small messages.

In summary, the choice between CB versus CTC may be seen as a case of typical "performance versus area" tradeoff. From this perspective, experimental results show that for a video processing application the latency penalty remains under 10% while the savings in terms of the overall area occupation of the network interfaces reaches 35%. Therefore, we believe that CTC is an effective solution of the message-dependent deadlock problem for throughput-driven stream processing applications.

Chapter 6

Network Interface Enhancement

A further development of the Spidergon architecture comprehends the capabilities' enhancement of each Network Interface. More specifically each NI should be capable to support not only one single communication paradigm (shared memory or message passing) at a time both of them at once. This fact implies that a network interface should be capable of supporting a PE (initiator) core connected on the same NI together with a SE (target) and also a message passing (called also streaming or CTC) core.

This kind of improvement induces a number of new issues related to arbitration, channel contention, and in a particular way connectivity.

6.1 Network Plug Switch

The solution proposed by the AST research Lab relies on a new network module, called *Network Plug Switch* (NPS).

As depicted in Figure 6.1 a NPS is a middleware module to be placed in between a *Spidergon* router and the PE/SE/CTC modules installed on a node. The optimal arbitration of the channels depends on the application to support. For our analysis we used a round robin policy similar to the one adopted in the standard *Spidergon* routers.

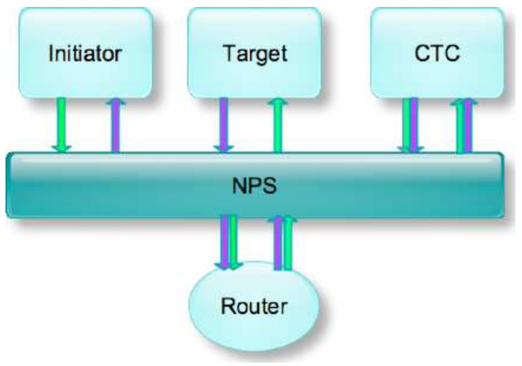


Figure 6.1: A node using a NPS module to interconnect a PE, SE and CTC module to the underlying router.

Integrating the PE/SE/CTC cores on a single node introduces also an issue on the protocol deadlock[1] problem.

Protocol-deadlock-free CTC traffic in fact is to be mixed with the shared memory traffic generated by the PE/SE cores that instead may generate ciclic dependencies. As described in Section 5.2 in fact in absence of a specific end-to-end flow control, Request packets traveling from an initiator towards a target and the Replay ones going in the opposite direction should travel on different and independent queues.

To archive this we must either introduce separate and independent networks or integrating the queues in a single network and split them in a number of virtual networks (VN).

A NPS node can be designed using a number of possible configurations. Figure 6.2 depicts those that in our point of view represent the most interesting and worth to be analyzed. In particular as shown in setting $S1$ to $S4$ a NPS node can be configured using a single NPS module interconnecting all the cores. Then this configuration can use one single router with two virtual networks, as in the case of scenarios $S3$ and $S4$, or can be enhanced by using two separated routers as in the case of $S1$ and $S4$. Moreover an additional channel can be added for the CTC node as in the scenario $S1$ and $S2$

[1]see Section 5.2

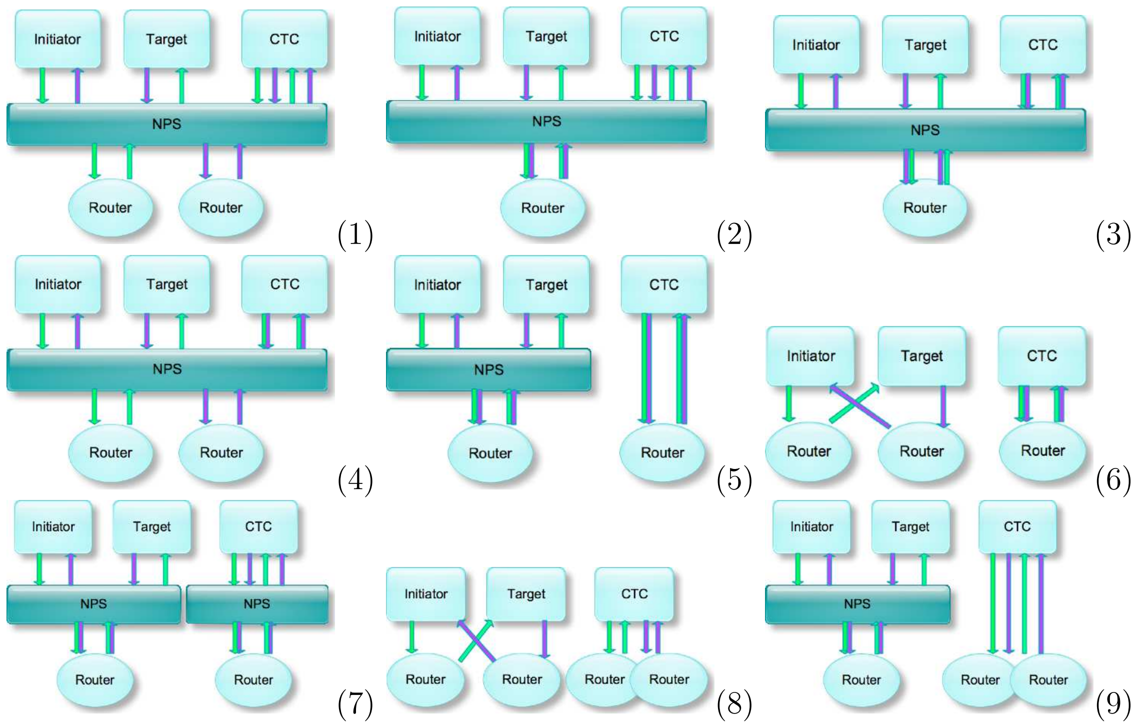
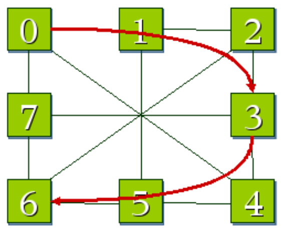


Figure 6.2: Possible Node setting: (1) NPS node with two CTC I/O channels and two routers with no VNs, (2) NPS node with two CTC I/O channels and one router with two VNs, (3) NPS node with one CTC I/O channel and one router with two VNs, (4) NPS node with one CTC I/O channel and two router with no VNs, (5) NPS node restricted to the MS traffic and independent network from CTC traffic, (6) system with no NPS, MS traffic traveling on two independent networks , (7) system based on two NPS nodes, one switching MS traffic and one switching CTC traffic, (8) system based on four independent physical networks CTC node with two I/O channels, (9) NPS node restricted to the MS traffic and double independent networks from CTC traffic.

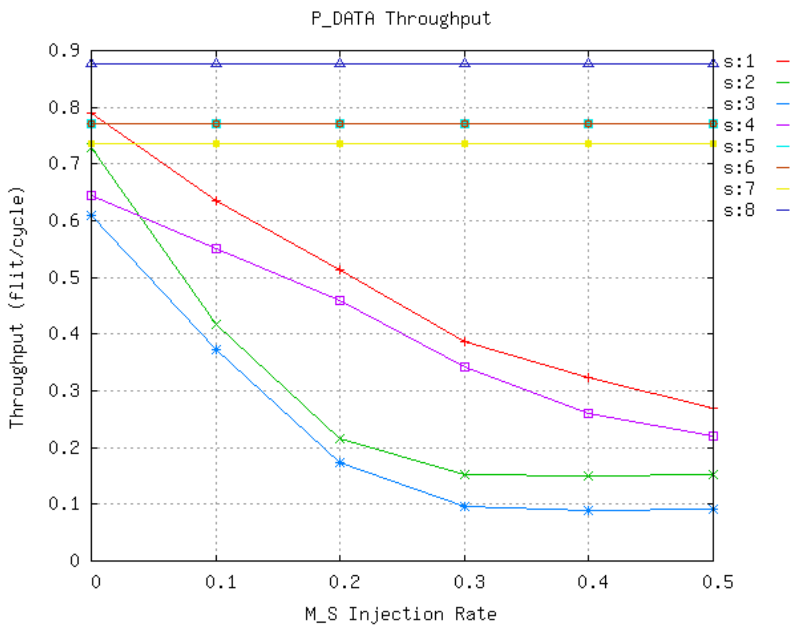
Scenarios $S5$, $S7$ and $S9$ use the NPS module to interconnect only the Master/Slave cores while the CTC core is connected to the NoC with virtual networks ($S5$), an additional NPS module ($S7$) or two separated routers ($S9$).

Scenarios $S6$ and $S8$ instead do not use a NPS module at all and hence they can be used as reference configurations.

6.1.1 Simulation and Analysis



(a)



(b)

Figure 6.3: The (a) traffic pattern used to test the contention between P_Data and P_Ack packets on the (b) output port of node three.

CTC protocol together with the NPS module requires a deep study to understand the role of each system parameter and to understand the difference between each possible scenario described in Figure 6.2.

Our study is first based on a simple traffic model shown in Figure 6.3(a): a stream of data between the node 0 the node 3 and the node 6. This model allows us to emphasize the critical point of node three represented in Figure 6.3(b). Here in fact the CTC protocol is more sensitive to the system parameters because of the contention between the P_Ack and the P_Data messages to control the output channels.

Our analysis is composed of four tests each one analyzing a single aspect of the given scenarios. In all tests the simulation parameters are those reported in the Table 6.1.1;

CTC inj.rate	**Stream Len.**	**Credit per P_Ack**	**Input Q.Size**
1.0 flit/producer	64 flit	Parametic	Min.: 8 or 12 flits
Routing Algo.	**MS Req. Size**	**MS Rep. Size**	**MS inj.rate**
Zero AFirst	L:1 flit S:10 flit	L:10 flit S:1 flit	Parametic

Table 6.1: Simulation parameters

The minimal queue size indicated in the table is function of the round trip time between the stream source and destination. In the considered case the distance between two peers is always two hops; when the NPS module is used the $RTT = 12$ clock otherwise $RTT = 8$ clock.

Test 1:"no noise" CTC parameter Analysis

The Max Packet Size (MPS) and the Credit Size (K) of the CTC protocol are two key parameters. The MPS varies between 0 and 16 with zero indicating no restrictions (as well as 16). The credit size K indicates how much a credit counter is to be incremented upon the reception of a P_Ack message.

To have a first understanding of the effects of these parameters we run a number of simulations varying each time one of the two parameters. The 3D graphs in Figure 6.4. reports the throughput of P_Data flits measured on the sink node 6.

Figure 6.4 reveals that there is not a specific couple $< MPS, CreditValue >$ that is optimal for all configurations.

In scenarios $S1$, $S8$ and $S9$, where the CTC module is separated from the MS ones and is provided with two independent channels connected (directly or through a NPS) to two disjoint routers, the best configurations are obtained setting MPS to the maximum value: zero (no restriction). In these cases, as long as the input queue is correctly sized, the value of K is not relevant. P_Ack packets in fact travel on a network different form the one used by the P_REQ packets and can be delivered

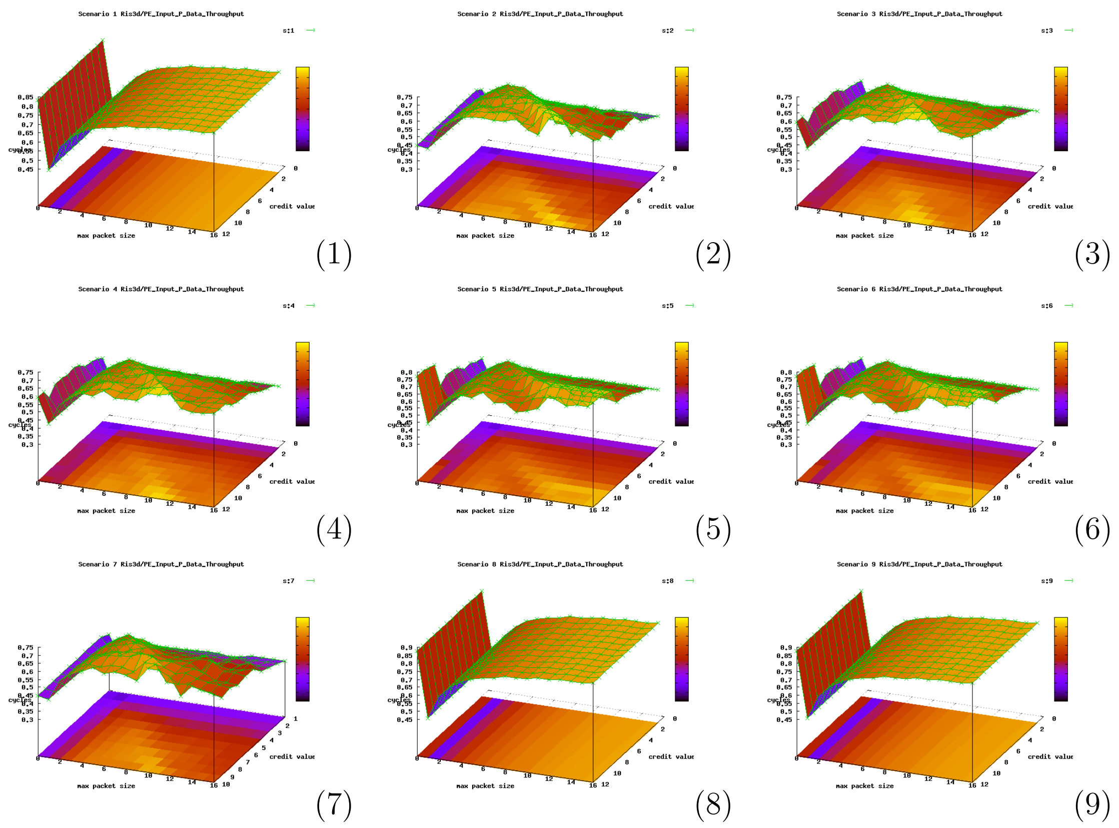


Figure 6.4: Throughput of P_Data flit received the the CTC PEs nodes in the different scenarios and with respect to the varying max packet size and credit counter.

in a minimal delay. Hence at node three the contention shown in Figure6.3(b) is avoided simply because packets do not share any queue or channel.

In the other scenarios the maximum throughput is obtained by setting big K values (hence big input queues) while MPS ranges between ten and twelve flits. In these scenarios in fact P_Ack messages have to compete with P_Data ones to obtain the access to a channel toward/from the CTC module. Small packet sizes generate an high number of headers, while big ones generate a "starvation" side effect on the CTC module. Small value of K instead generate a high number of P_Ack decreasing the system performance.

Table 6.1.1 summarizes the best configurations and their throughput measured on each PE involved in the data stream. Figure 6.5 reports a comparison histogram

scenario	**credit**	**max packet size**	**throughput**	**difference %**
1	*	0	0.8311	5.2%
2	11	11	0.7457	16.1%
3	11	11	0.7457	16.1%
4	11	11	0.7457	16.1%
5	11	0	0.7713	12.0%
6	11	0	0.7713	12.0%
7	10	10	0.7196	17.9%
8	*(1)	0	0.8768	0.0%
9	*(1)	0	0.8768	0.0%

Table 6.2: Best performing configurations for the no-noise (no MS traffic) test and their difference from the best performing scenario. The value in parenthesis indicates the value used in the following tests.

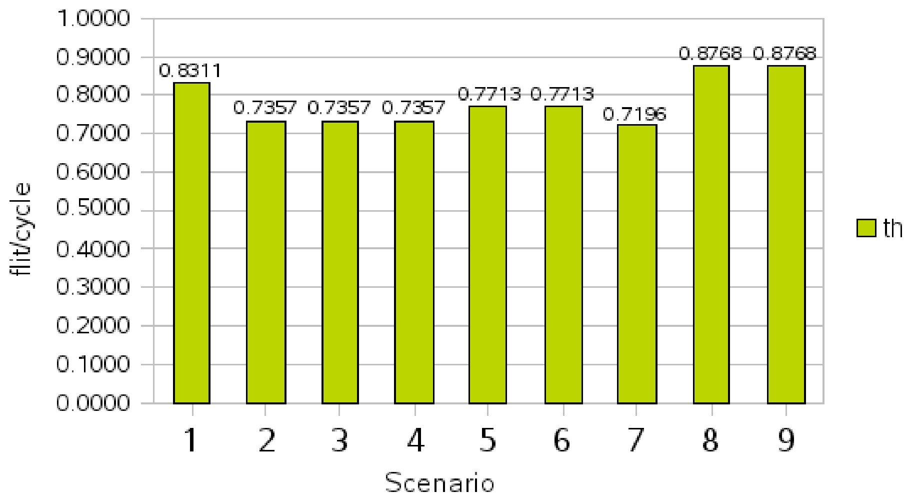


Figure 6.5: Test1: throughput comparison of the considered scenarios using their best performing configurations.

of the measured throughput. As expected scenarios with four dedicated routers have the best performance (scenarios $S8$ and $S9$).

In absence of Master/Slave traffic we notice that the NPS module introduces a 5% of throughput degradation. The use of two separated channels for the CTC nodes gives a 10% improvement when also two routers are used (scenarios $S8$ and $S9$).
In the case of one single router, using two channels for the CTC node (scenario $S7$) the additional channels do not improve the performance of the system as the contention between CTC messages is simply moved from the CTC node to the underlying router.

Test 2: "no noise" CTC resistance to MS traffic

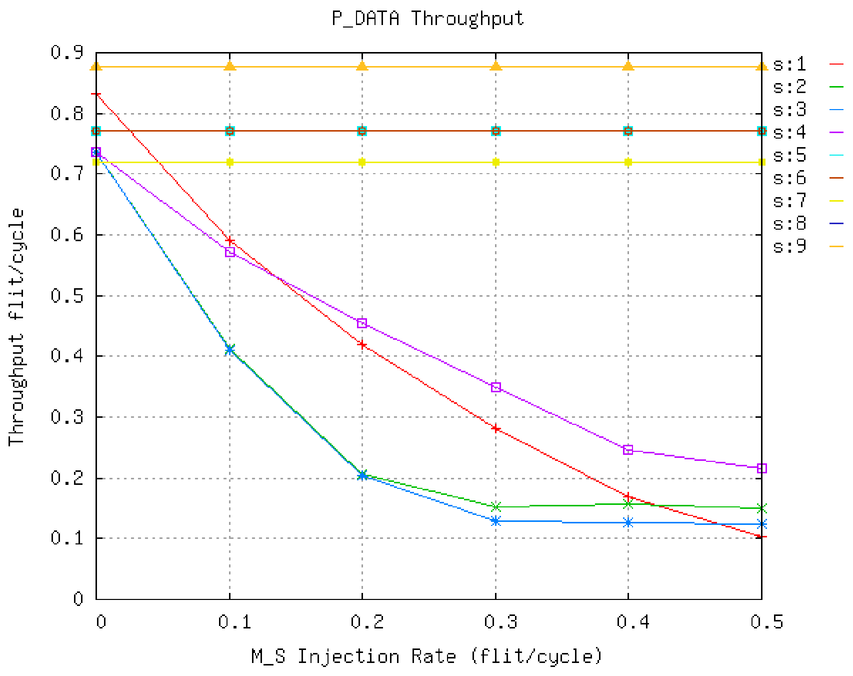


Figure 6.6: Performance degradation adding Master/Slave traffic to the "no noise" best setting.

Figure 6.6 reports the throughput degradation measured when adding Master/Slave traffic to the scenarios of Figure 6.2 configured with the "no noise" best setting of Test 1. The Master/Slave is generated by each node of the NoC following a random uniform distribution.

As expected in the cases where CTC traffic travels along disjoint channels the performance are not affected by the noise-traffic. Among these, the cases where CTC control messages travel along different networks have again the highest throughput.

In line with the previous test, when only one router is used performances are reduced by a a 10%. When also a CTC-dedicated NPS is added another 5% is lost.

Considering scenarios $S1$ through $S4$, as expected the cases with one single router are very sensitive to the M/S noise-traffic. Adding the CTC channels brings small improvements. It's interesting to note the difference of performance between scenarios $S1$ and $S4$. Scenario $S4$ adopting only one CTC channel performs better than $S1$, the case with two CTC channels.

In the previous pages we showed the benefits of an additional channel for the CTC protocol. This poor performance of $S1$ versus $S4$ is to be due to its $< MPS, CreditValue >$ configuration. In case of M/S traffic hence the best configurations seen in Test 1 must be revised.

Test 3: "noisy" CTC parameter Analysis

Test 3 is similar to Test 1: we varied the combination $< MPS, CreditValue >$ ranging from 0 to 16. In Test 3 we added a fixed noise traffic whose injection rate is of 0.2 $flit/cycle$ on each node. Figure 6.7 shows the throughput measured in all the considered scenarios.

Table 6.1.1 summarizes the best setting found for the $< MPS, CreditValue >$ couple when noise traffic is added while Figure 6.8 graphically reproduces the different performances.

We notice that the difference between the best and worst scenario now grows up to 75%. As expected the best scenarios are those who have four independent networks so that the Master/Slave and CTC traffic are orthogonal. The worst case is $S3$ when CTC and MS traffic are mixed in a single physical network.

Results for scenarios $S5..S9$ are essentially the same of those in Test 2. Scenarios $S1..S4$ instead are those sensitive to the MS traffic.

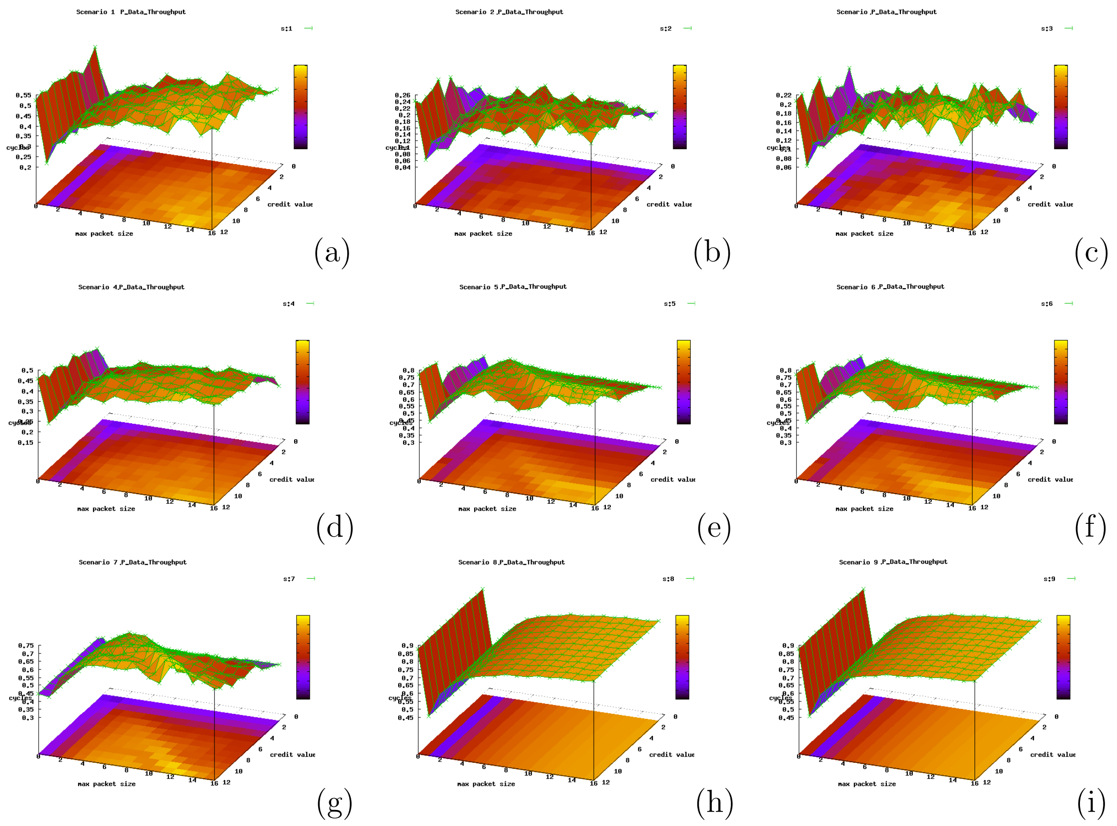


Figure 6.7: Throughput of P_Data flit received the the CTC PEs nodes when "noisy" Master/Slave traffic is used with respect to the different scenarios to the varying max packet size and credit counter.

Considering scenarios $S1$ and $S4$ we notice that the throughput difference is around 7%. This quantifies the gain offered by the additional CTC channel. Scenarios $S2$ an $S3$ show that reducing the number of routers available to the system can degrade the performance by up to 40%.

Test 4: "noisy" CTC resistance to MS traffic

Figure 6.9 shows the loss of performance of the considered scenarios when we use the $< MPS, CreditValue >$ configuration obtained from Test 3 and reported in Table 6.1.1.

As in Test 2 scenarios $S5$ through $S9$ are orthogonal to MS traffic and their

scenario	credit	max packet size	throughput	difference %
1	12	13	0.5405	38.4%
2	12	13	0.2486	71.6%
3	6	13	0.2187	75.1%
4	12	14	0.4830	44.9%
5	12	0	0.7714	12.0%
6	12	0	0.7714	12.0%
7	11	11	0.7357	16.1%
8	*(1)	0	0.8768	0.0%
9	*(1)	0	0.8768	0.0%

Table 6.3: Best performing configurations for the noisy (with MS traffic) test and their difference from the best performing scenario. The value in parenthesis indicates the value used in the following tests.

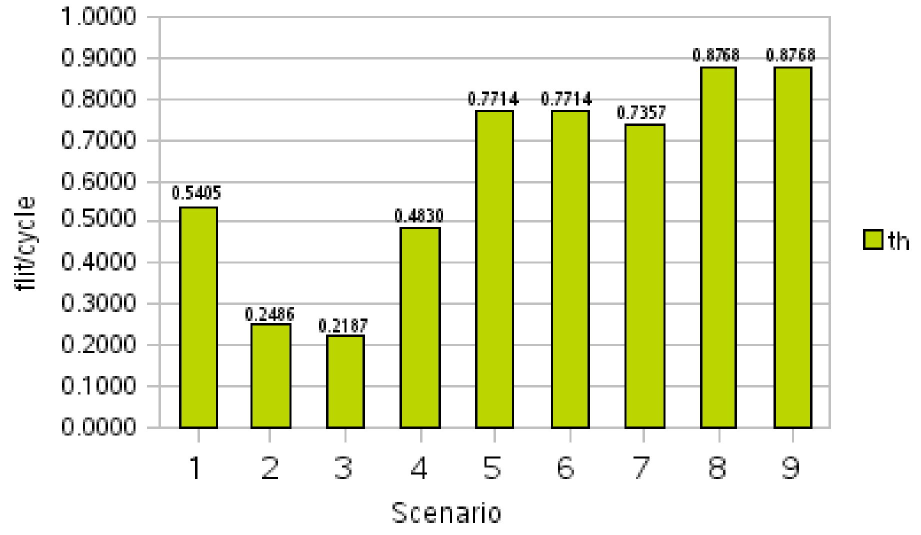


Figure 6.8: Test1: throughput comparison of the considered scenarios using their best performing configurations.

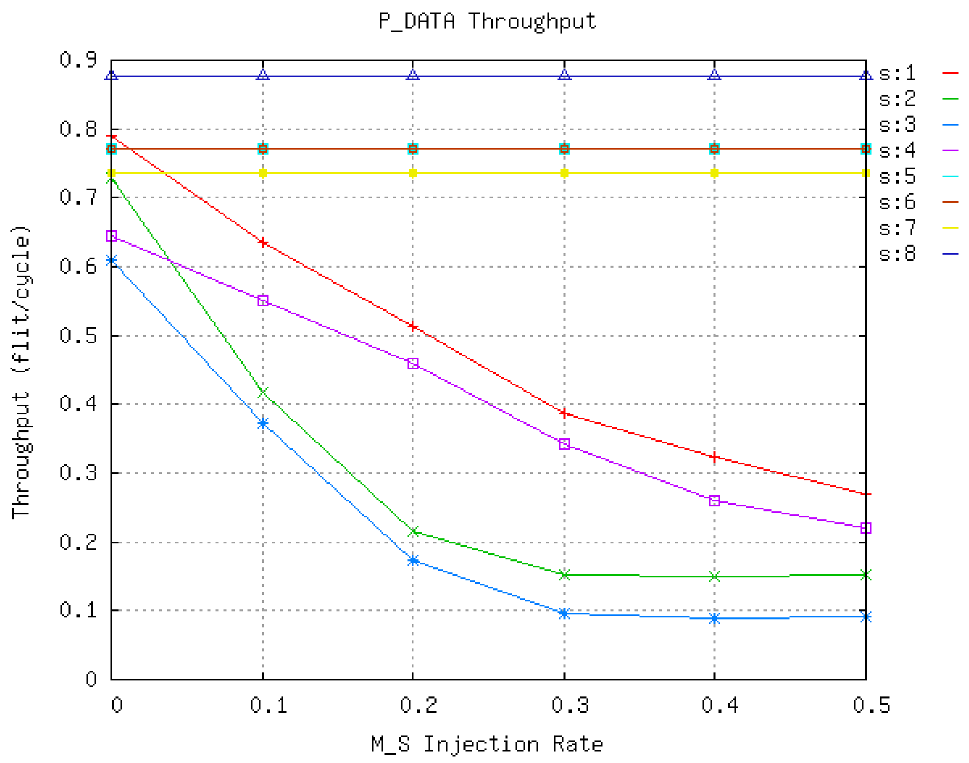


Figure 6.9: Performance degradation adding Master/Slave traffic to the ”noisy” best setting.

difference of throughput is due to the different number of routers, CTC channels and NPS modules.

Scenarios $S2$ *and* $S3$ are particularly sensitive to the noise because of the single router used to handle the two different traffics.

Finally Scenario $S1$ performs better than scenario $S4$ confirming that in Test 2 the reason that made it perform worse than scenario 4 was the bad $< MPS, CreditValue >$ setting (in Test 2 scenario used an unrestricted packet size whereas in Test 4 is has a finite and fixed value).

Part IV

Communication Issues

Chapter 7

Communication Link Systems

7.1 Overview

In the case of SoCs for embedded applications designers use standard industrial CAD-tool flows for the synthesis of a platform-specific NoC and must cope with an increasing number of timing-closure exceptions due the differences in size across its heterogeneous processing cores. This problem becomes particularly hard when using nanometer technology processes [P$^+$07] as the impact of global interconnect wires raises exponentially the number of *wire exceptions*, i.e. timing-closure violations due to the delay of a global wire exceeding the target clock period T_{clk} [CSV02, HMH01].

A method to fix wire exceptions is *wire pipelining*, i.e. the insertion of sequential elements (or clocked buffers) to pipeline long wires in shorter segments whose delays meet T_{clk} [CMSSV99, Coc02, LPP04, LZKC02, Sch02].

By providing one or more extra clock periods to traverse long distances, wire pipelining trade-offs latency for throughput. As proposed by Jalabert *et al.* [JBMM04], the use of latency-insensitive protocols [CMSSV99] in NoC design allows channels to be pipelined to an arbitrary degree, thus decoupling T_{clk} from the worst-case channel delay.

Latency-insensitive protocols are implemented using *relay stations*, clocked repeaters of unit latency and twofold storage capacity. Relay stations can be used instead of regular flip-flops to enable arbitrary wire pipelining between two routers

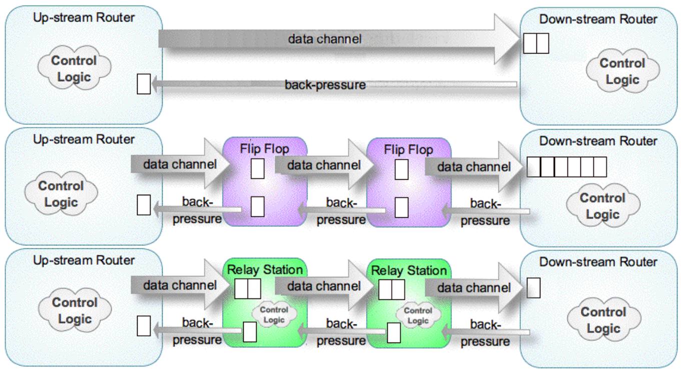


Figure 7.1: Alternative ways to pipeline NoC channels.

in a NoC (Figure 7.1). Further, when combined with flit-buffer flow control methods [DT04], relay stations can store flits in the presence of persistent congestion because they actively process the flow-control signals. Hence, their use effectively increases the total storage capacity of the channel, thereby opening the way for interesting design optimizations.

We study in detail the interaction between wire pipelining and NoC flow-control methods and we propose *distributed flit-buffer flow control* as a technique that combines the simplest form of ack/nack protocol with the distribution of relay stations on the NoC channels for both buffering and wire pipeline purposes. We show how this approach provides NoC designers with both better options to optimize performance/area trade-offs and precious flexibility to complete efficiently the NoC physical design stage.

Pullini *et al.* studied the interaction between wire pipelining and flow-control focusing on providing fault-tolerant communication on the NoC channels, a goal that is outside the scope of this paper [PABB05]. Hu *et al.* proposed an algorithm for optimal buffer sizing in packet-switched or virtual-cut-through NoCs [HOM06]. Ogras *et al.* proposed a technique to improve the performance of a Mesh NoC by incrementally inserting additional long pipelined channels [OM06b]. Methods to optimally size queues in on-chip global communication channels are presented in [LK03, CXSP04].

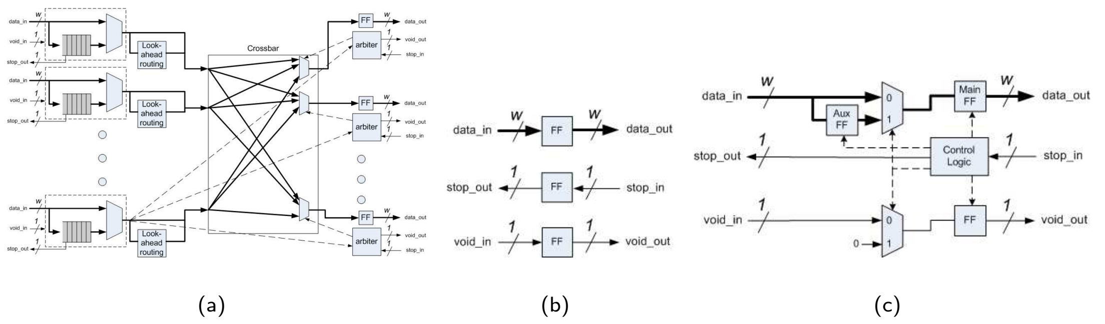


Figure 7.2: NoC components: (a) router; (b) FF-repeater; (c) RS-repeater.

7.2 Basic NoC Components

We summarize here the main characteristics of the three basic NoC components used in the rest of the paper.

The **router** is the key component of a packet-switched NoC. Figure 7.2(a) shows the basic structure of a router implementing a XY-routing algorithm supported by wormhole flow-control. Solid lines show the data plane while dashed lines show the control plane. A crossbar switch separates the input from the output part. Each input port is equipped with a *look-ahead routing* module and a *by-passable queue* of size Q and parallelism W (*flit width*). Each output port has W *output registers* to store the forwarded flit and an *arbiter* to allocate the port among competing input worms. With look-ahead routing each router pre-computes the output port for the next downstream router: the information is carried in the worm head-flit, which can now be forwarded directly to the output port (if available). This leads to better performance by reducing the router critical path and allowing the routing task to be executed in parallel to arbitration. Without congestion a flit traverses the router in one clock cycle. In case of congestion, the flits of a worm that loses the arbitration are temporarily stored in the queue. When the queue gets filled, *back-pressure* is triggered according to the given low-level flow-control mechanism.

An **FF-repeater** is the simplest type of channel repeater (Figure 7.2(b)). It consists of a number of flip-flops (FF) equal to the flit width W plus two FFs: one for the *void* signal distinguishing valid flits from void ones and one for the *stop* signal carrying back-pressure information backward on the channel. At each clock cycle a FF-repeater samples a new flit and makes it available on the output ports *without* processing the void/stop signals. Thus, each flit spends *exactly* one cycle on each FF-repeater without the possibility of being stored. Hence, in the case of a persistent congestion the flits of a worm end up being stored in the router queues while the channel FF-repeaters are empty. The overall channel latency is equal to the number of channel repeaters K.

An **RS-repeater** is a more complex repeater based on the relay station (RS) circuit that was first proposed for latency-insensitive design [CMSSV99]. Relay stations are used implement a latency-insensitive protocol, but when used in NoC design enable also a distributed implementation of flow control.

Figure 7.2(c) shows the structure of a relay station: it consist of a battery of W *main FFs* in parallel with W *auxiliary FFs* plus one FF for the void signal, one additional FF that is used both to sample the stop signal and to implement the two-state finite-state machine governing the flow-control mechanism.

At each clock cycle a RS-repeater samples a new flit into its main flip-flops to make it available on its output port. However, if it samples also the asserted *stop_in* value then it goes in a stalling state to: (a) keep the present flit on the main FF (to make it available again on the output port in the next cycle), and, (b) sample any newly-arrived valid flit in the auxiliary FF while asserting *stop_out* so that the upstream node will be stalled too. Hence, in the case of a persistent congestion a channel of K RS-repeaters contains $2 \cdot K$ flits.

In Section 7.4 we study in detail the effective ratio between the area of an NoC based on RS-repeater and an equivalent NoC based on FF-repeater as a function of the flit width W. Meanwhile, as a *rule of thumb* we assume that this ratio is equal to two.

7.3 Wire Pipelining & Flow Control

Flow-control methods can be classified based on their granularity of channel bandwidth allocation and of buffer allocation [DT04]. The basic unit of bandwidth and storage allocation is a *flit (flow control digit)*. Packets are divided in sequences of flits. Differently from packets, flits carry no routing and sequencing information. Flit-buffer flow control allocates both bandwidth and buffers in units of flits. This has three advantages: it (a) reduces the storage required for correct operation of a router, (b) provides stiffer back-pressure from a point of congestion back to the source of a flit stream, and (c) enables more efficient use of storage. Since these advantages match well the characteristics of on-chip communication, flit-buffer flow control methods are seen as a promising solution for NoC, where typically the size of a flit matches the parallelism of a channel. The two main high-level flow-control methods are *wormhole flow control* and *virtual-channel flow control.*

These need to be supported by one of three main low-level flow control mechanisms that provides buffer management and back-pressure, namely: *on/off, credit-based* and *ack/nack* [DT04]. In our analysis we focus on the combination of wormhole flow control with each of these low-level mechanisms.

Besides allocating the NoC bandwidth and storage resources, flow-control methods should provide good performance by guaranteeing a high bandwidth for the transmission of a stream of flits in the presence of possible *intervals of stalling cycles* caused by congestion in the downstream nodes. The choice of the flow-control strategy has consequences on the design of the network components and, particularly, on the size of the flit-buffering queues in the routers. If we want to avoid dropping flits, the correct operation of a particular flow-control method sets a constraint on the minimum size Q_{min} of Q. Once this constraint is met, raising the value of Q leads generally to better performance. This, however, varies depending on the network traffic as discussed in Section 7.5. Also, in practice, raising it beyond a certain value leads to diminishing returns.

On/Off

is a simple flow-control mechanism that minimizes the amount of back-pressure signaling in exchange for larger queue size. The upstream node has a single-bit state register that switches between *on* and *off* states based on the last back-pressure signal received from the downstream node. The latter sends an *off* signal back whenever the number of free slots in its flit-buffering queue goes below a threshold F_{off} and sends an *on* signal whenever it goes above a threshold F_{on}.

Hence the minimum size Q_{min} depends on the number of flits that can be received during the time T_{rt} from the instant when an *off* signal leaves the downstream node until the instant when the downstream node has received the last flit transmitted by the upstream node before stalling due to the processing and reception of signal *off*. In a synchronous NoC, if the latency of the channel is K clock cycles, then $T_{rt} = (2 + 2 \cdot K) \cdot T_{clk}$. Hence, for protocol correctness, $Q_{min} = 2 + 2 \cdot K$.

However to obtain a maximum sustainable bandwidth to optimize the bandwidth we must consider also the dual case when the downstream node sends an *on* signal upstream to resume transmission. In this case at least T_{rt} cycles must pass by before a new flit arrives downstream. Meanwhile, in order to have sufficient flits to forward to the next hop, the downstream queue must be able to contain as many additional flits for a total size

$Q_{min} = 2 + 4 \cdot K.$

Credit Based

is a flow control mechanism where the upstream node has a counter to track the number of available free slots in the downstream queue. The counter state is decremented whenever a flit is transmitted and incremented whenever a credit signal arrives from the downstream node, which in turns sends the credit whenever it has succeeded in forwarding a flit from its queue to the next hop. With respect to on/off, credit-based flow control requires more "back-pressure signaling", but smaller queues. Specifically, for protocol correctness, a $Q_{min} = 1$ is enough. However, such

Rep.	**On/Off**		**Credit Based**		**Ack/Nack**	
type	Q_{min}	S	Q_{min}	S	Q_{min}	S
FF	$2+4K$	$2+5K$	$2+2K$	$2+3K$	$1+2K$	$1+3K$
RS	2	$2+2K$	2	$2+2K$	1	$1+2K$

Table 7.1: Queue size and channel total storage per flow-control/repeater type.

small size leads to the insertion of void flits (*bubbles*) at every hop because only one credit is available on each channel at any given time. Hence, only one flit can be forwarded per each round-trip of this credit and the higher is the value of $K > 1$ the lower the performance. To avoid bubble insertion the queue must be sized based on the round-trip latency. $T_{crt} = (2 + 2 \cdot K) \cdot T_{clk}$, which leads to $Q_{min} = 2 \cdot (1 + K)$.

Ack/Nack

does not require any state in the upstream node to indicate buffer availability in the downstream node. Instead flits are optimistically sent whenever they become available: if the downstream node has a slot available in the queue it accepts the flit by sending an *ack* signal, otherwise it drops it and sends a *nack* signal. Ack/nack flow control mechanism is traditionally considered inefficient both in terms of storage (it requires that each transmitted flit be held waiting for an acknowledgement) and bandwidth (due to the potential retransmissions) [DT04].

Further, since it is based on acknowledging the reception of each specific flit, it works well for a channel of unit latency. But, if the channel contains K FF-repeaters it becomes suboptimal with respect to credit-based, where an acknowledgment denotes the successful forwarding of a generic flit. t Still, ack/nack was effectively used to implement fault-tolerant *Go-Back-N* protocols [PABB05] with routers having *output queues* of size $Q_{min} = 1 + (2 \cdot K)$.

What are the best combinations?

The first row of Table 7.1 summarizes the requirements on the queue size for the three flow control methods as well as the corresponding values for the *channel total* *storage* S. The value of S is obtained by adding the queue size and the amount of storage provided by the channel repeaters, which is independent from the flow control method. In the case of a channel containing K FF-repeaters, each repeater provides storage for one flit, thus resulting in K flit buffers distributed on the channel. For FF-repeaters, the credit-based flow control method is the best choice in terms of sustainable bandwidth per unit of storage.

The second row of Table 7.1 shows the corresponding numbers for the case when the repeaters are implemented as relay stations. Since a RS-repeater can store up to two flits, the distributed storage on a channel of K RS-repeaters is equal to $2 \cdot K$. But, the fact that a relay station contains the logic implementing the low-level flow control mechanism makes it possible to reduce the size of the downstream queue to a minimum value that is *always* as if $K = 0$. In particular, ($Q_{min} = 1$) is sufficient when combining RS-repeaters with the ack/nack flow control mechanism. Ack/nack signalling naturally matches the *stop_in*/*stop_out* signalling proposed for latency-insensitive protocols [CMSSV99] and, indeed, we will show that it is the best design choice when using RS-repeaters.

In summary, *independently from the chosen flow-control mechanism, for any value of* $K > 0$ *the value of the channel total storage* S *that is needed for a correct behavior when using RS-repeaters is always smaller than the value needed when using FF-repeaters.*

This result, which is reached with an analytical model based on the rule of thumb that the RS-repeater area is twice the flip-flop area, is validated by our experiments with the semi-custom design of many channel subsystems for various flit widths (Section 7.4). While queue sizes larger than Q_{min} generally benefit the network performance, the optimal size depends on the network topology and application traffic [CXSP04, HOM06]. Still, as shown by the system-level experiments of Section 7.5, the ability of working with a lower Q_{min} gives an important advantage to a NoC that employs RS-repeaters instead of FF-repeaters.

(a)

W	FF	RS	RS/FF
16	735	2348	3.19
32	1390	4100	2.95
64	2699	6514	2.41
128	5318	13903	2.61
256	10557	21316	2.02
512	21034	37599	1.79

(b)

W	$K = 1$			$K = 2$			$K = 3$		
	RS	FF	ratio	RS	FF	ratio	RS	FF	ratio
16	29k	34k	0.84	38k	43k	0.88	48k	54k	0.89
32	42k	52k	0.81	58k	66k	0.89	75k	84k	0.89
64	66k	88k	0.75	92k	116k	0.80	118k	159k	0.74
128	123k	151k	0.81	178k	217k	0.82	234k	296k	0.79
256	203k	284k	0.72	288k	408k	0.71	374k	531k	0.70
512	363k	545k	0.67	513k	788k	0.65	664k	1006k	0.66

Table 7.2: Area [um^2] as function of W: (a) FF-repeater vs. RS-repeater and (b) FF-system vs. RS-system.

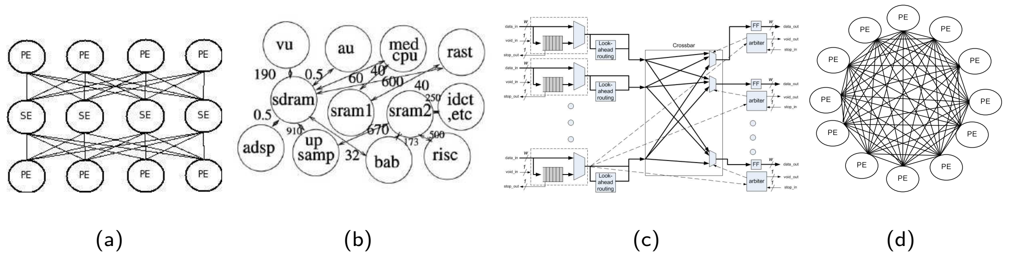


Figure 7.3: Application task graphs: (a) 4-Rooted Tree Forest (4RTF), (b) MPEG4 decoder, (c) VOPD decoder, and (d) random uniform traffic (URT).

7.4 Area Occupation Analysis

We completed VHDL parameterized designs for the NoC components presented in Section 7.2 and synthesized many versions of them with a $90nm$ industrial standard-cell library.

Table 7.2(a) reports the area occupation of a RS-repeater versus a FF-repeater as function of the flit width W varying from 16 to 512 bits. The target clock frequency was set equal to $2Ghz$ and met by all repeaters under all configurations. Each output port was loaded with a wire capacitance that was previously characterized by considering an optimally-buffered wire implemented in an intermediate metal level.

Generally the higher is the flit width the lower is the ratio of the RS-repeater area over the FF-repeater area. The ratio goes from 3.19 (for $W = 16$) to 1.79 (for

$W = 512$). This is not surprising since the additional overhead due to the flow-control logic becomes less important with respect to the fact that a RS-repeater has twice the number of FFs as an equivalent FF-repeater [1].

Still, based on these results one may think that the rule of thumb of considering this ratio equal to two is justified only for channel with large width.

Before drawing this conclusion, let's consider what happens when the repeaters are instanced as part of a long channel in a NoC.

We instanced a 5×5-port version of the router of Table. 7.2 (a) and we connected four of its five output ports to as many long repeated channels (while we assume that the fifth port is used for the local connection). This subsystem corresponds exactly to one "tile" of a 2D-Mesh NoC and, therefore, its area is a good estimate of the overall NoC area.

For instance, in a 4×4 mesh the subsystem would account for $\frac{1}{16}$ of the NoC area.

Indeed, we considered two subsystems:

- *RS-subsystem*: a router plus 4 channels pipelined using RSs.
- *FF-subsystem*: a router plus 4 channels pipelined using FFs.

However, for both subsystems we used the same router implementing ack/nack flow control. This is advantageous for the FF-subsystem since a credit-based router would have a larger area than an equivalent ack/nack router because it needs an additional counter at each output to store status information on the outstanding credits.

[1] The fact that for very-high values of W, i.e. 512 bits, the ratio goes below two can be explained as follows: both RS- and FF-repeaters are equipped with an array of output buffers to drive the wire load capacitance. The auxiliary FFs inside the RS-repeater, instead, do not need them because they are directly connected to the local multiplexers. Therefore, for high-values of W the area of the main FFs and the buffer dominates the area of a RS-repeater and it is comparable to the area of an equivalent FF-repeater.

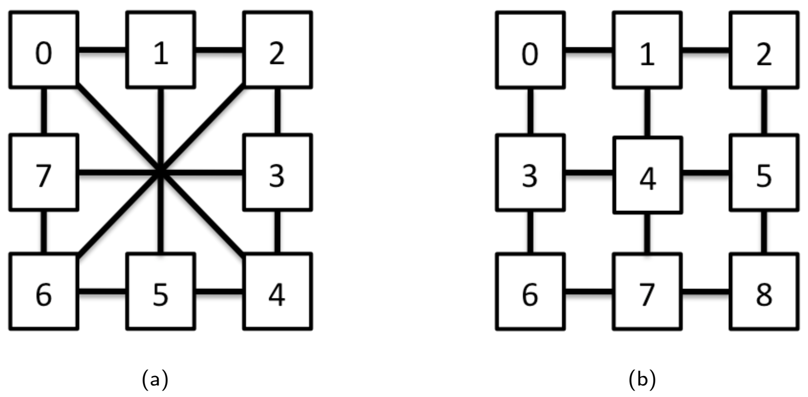


Figure 7.4: NoC topology examples: (a) a 8-node *Spidergon* and (b) a 3×3 2D Mesh.

Table 7.2(b) reports the area occupation of a RS-subsystem versus a FF-subsystem as function of the flit width W, which varies from 16 to 512, and the number K of repeaters on each channel, which varies from 1 to 3. The input queues of the router are set to the minimum value Q_{min}, i.e. 1 for the RS-subsystem and $2+2 \cdot K$ for the FF-subsystem. As the value of W grows, the area ratio approaches the theoretical limit of $\frac{2}{3}$, which is obtained from the analytical model by dividing the corresponding values of S from Figure 7.1. In conclusion, *under every condition the RS-subsystem is always significantly smaller than the FF-subsystem.*

7.5 System-Level Simulations

For our system-level simulations we considered two NoC topologies (Figure 7.4): a 2D Mesh, which broadly represents a class of NoCs that have been proposed for various general-purpose chip multiprocessors and Spidergon, an NoC architecture aimed at SoC for embedded applications [CLM+04]. Spidergon is a bidirectional ring with an even number of nodes enriched by "across" bidirectional channels between opposite nodes.

We built an event-driven simulator with detailed models of the parameterized NoC components (routers, FF-repeaters, RS-repeaters) and high-level models We built detailed models from the parameterized NoC components (routers, FF-repeaters, RS-repeaters) in the Omnet++ event-driven network simulator [OMN] and we combined them with high-level abstractions of the processing elements (PE) and memory elements (ME) that are on the chip. Each node in both the 2D Mesh and Spidergon contains either a PE or an ME attached to the local port of the router via a network interface that performs the operations of fragmenting packets into flits (and vice versa).

The 2D Mesh uses 5×5 router implementing the well-known XY-routing algorithm [DT04], while Spidergon uses 4×4 routers implementing a discrete minimal routing algorithm that forwards the incoming flits along the across channels if their destination is "closer" to the opposite half of the ring, and sends them along the ring otherwise. Wormhole flow control is used in both NoCs.

We simulated the two NoCs with four different traffic patterns taken from the literature (Figure 7.3):

1. the *4-Rooted Tree Forest (4RTF)* models a scenario where 4 MEs are uniformly shared as communication targets by 8 PEs: Each PE initiates a communication by sending either a load or a store request to a given ME that replies with either data or an acknowledgement. This is a typical scenario in many embedded applications where a shared memory bank becomes a central hot spot of the NoC [BCG+07];

2. a central memory hot spot is present also in the *MPEG decoder* SoC where various PEs exchange data by means of three memories (SDRAM, SRAM1 and SRAM2) [BJM+05];

3. PEs in the *Video Object Plane Decoder (VOPD)* SoC, instead, exchange data via point-to-point communication [BJM+05]. In this case, as in the following, a communication initiated by a PE is not followed by a reply from the target node;

4. in the *uniform random traffic (URT)* case, each node is a PE that communicates with every other PE in the system.

In this work we compare the different systems using the following performance metrics:

- *packet latency*: time taken by a packet to enter the network, traverse it, and reach the destination;
- *round-trip time*: time elapsed from the transmission of a request packet and the reception of the corresponding replay packet;
- *bandwidth*: number of flits reaching a node per time unit.

7.5.1 Bandwidth Analysis

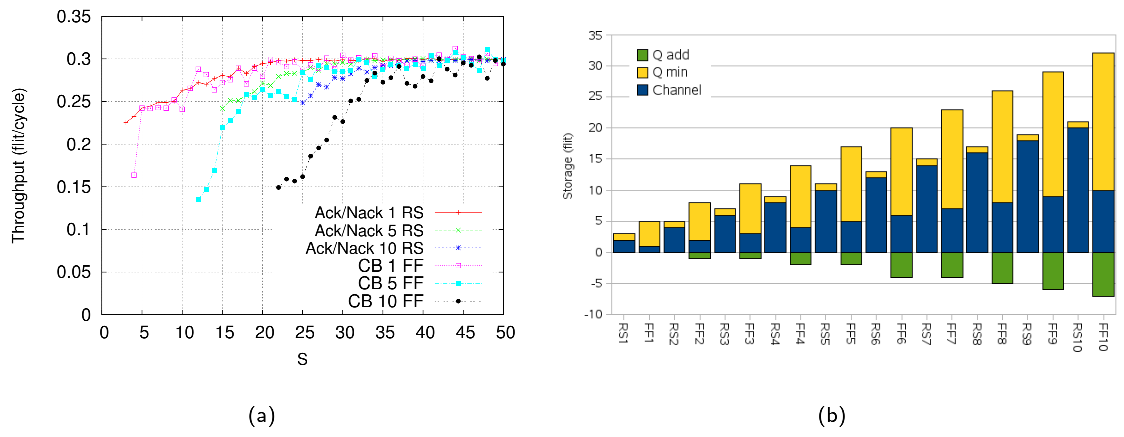


Figure 7.5: Results with *Spidergon* supporting the 4RTF traffic: (a) average bandwidth and (b) breakdown of channel total storage.

We report experimental results only for the 12-node *Spidergon* NoC supporting the 4RFT traffic pattern because the results for the other topology/traffic combinations are similar. Figure 7.5(a) shows the average bandwidth as function of the channel total storage S. Each PE has a fixed injection rate that is higher than what the NoC can sustain if the router queue sizes are kept at the minimum value Q_{min}. From this graph it is clear that the amount of storage available on a channel significantly influences the system performance: as the storage increases, more flits can be stored in the routers' queues reducing the channel contentions (with wormhole switching single packets are stored along multiple routers). Hence, the saturation threshold is raised and the performance of the NoC improved.

The bar diagram of Figure 7.5(b) reports the breakdown of the channel total storage that is required to obtain maximum bandwidth in a non-saturated NoC as function of the number K of channel repeaters. In particular, the sequence of points on the x-axis corresponds to 20 different design scenarios for K that varies from 1 to 10. For each value of K there are two bars: one corresponding to the RS-repeaters and one corresponding to the FF-repeaters. In each design scenario K repeaters are uniformly distributed on each NoC channel. Each bar includes up to three components:

- the blue (dark) part is the amount of storage provided by the repeaters, i.e. K for FF-repeaters and to $2 \cdot K$ for RS-repeaters;
- the yellow (light) part is the size Q_{min} of the router's input queues that is necessary to correctly support the given flow control, as explained in Section 7.3. This is always 1 for a RS-system and it is equal to $2 + (2 \cdot K)$ for a FF-system;
- the green (grey) part is the additional amount of storage Q_{add} that queues must have to reach the maximum bandwidth. Notice how for FF-systems under the analyzed traffic scenario, Q_{add} is lower than zero indicating that the maximum bandwidth can be reached with less storage than the one provided to satisfy the Q_{min} optimization constraint. In other words, the network can tolerate the insertion of a certain amount of bubbles per hop. Since bubbles increase the worm length, the worm can lock more channels during the time it passes through the NoC. Still, when the storage is high enough, the impact of those bubbles on the channel occupation is reduced.

In all the scenarios the RS-system reaches the maximum bandwidth using an amount of storage smaller than the corresponding FF-system, with an improvement that goes from 40% in case of small values of K down to 15% for K equal to ten. Notice, however, that for the foreseeable future it is expected that wire pipelining will be limited to the insertion of few repeaters, i.e. less than 5, even in large chips.

7.5.2 Latency Analysis

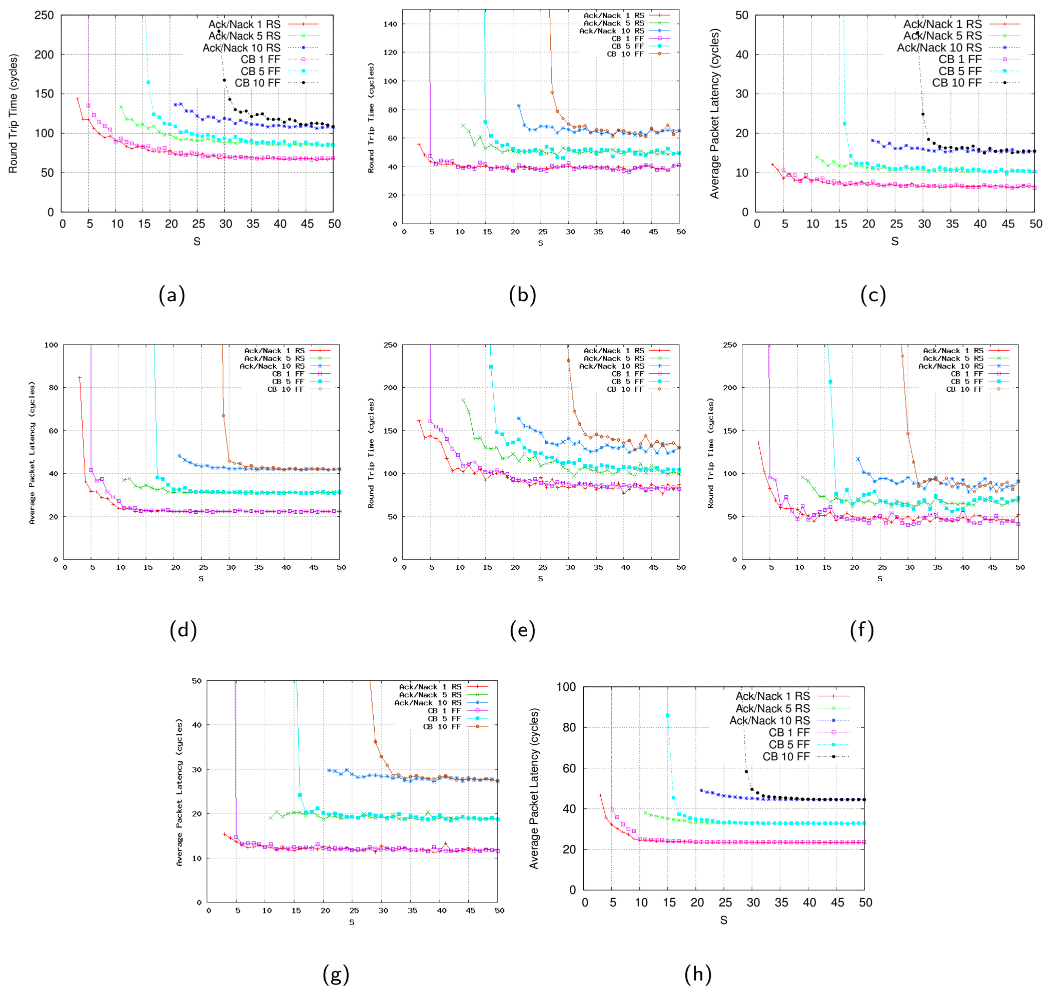


Figure 7.6: Comparing RS-repeaters with Ack/Nack vs FF-repeaters with Credit Based: Round trip time for *Spidergon* with (a) 4RTF, (b) MPEG4, and 2D Mesh with (e) 4RTF, (f) MPEG4. Average packet latency for *Spidergon* with (c) VOPD, (d) URT and 2D Mesh with (g) VOPD, (h) URT.

The charts in Figure 7.6 compare the RS-system and the FF-system with respect to the round-trip delay or packet latency for *Spidergon* and Mesh in function of the considered traffic patterns as we vary the channel total storage S.

As a theoretical exercise we let vary S up to 50 flit slots, but the curves reach a *minimal latency* value much earlier than that. The RS-system performs better than the corresponding FF-system for a given S and requires a smaller value of S to meet a given maximum latency constraint. In Figure 7.6(a), for instance, when $K = 5$ the RS-system does not exceed a latency of 17 cycles using 35% less storage than the FF-system, while for a fixed $S = 17$ it delivers 12% less latency. Further, for the case FF-repeaters if the queues have size $Q < Q_{min}$ then the credit-based flow control creates many bubbles that increase dramatically the average NoC latency.

Next, we measured the channel total storage S_{delay} that is required to stay within 10% of the above-mentioned minimal latency. This value depends on the specific NoC, the application task graph, the PEs' injection rate, and the number of repeaters K. The bar diagrams in Figure 7.7 report S_{delay} for all design combinations. Generally, higher values of S are needed than for bandwidth optimization (Figure 7.5(b)). Again, RS-systems shows better performance than corresponding FF-systems, e.g. requiring up to 30% less storage in the case of Spidergon/4RTF. Reaching the minimal latency with a lower storage amount indicates that the given resources are better exploited. Indeed, this is the case for the RS-system where the storage deployed on the NoC can be used also to buffer the flits traversing the channels.

Experiments with Non-Uniform Repeater Deployment

In the previous experiments we made the simplifying assumption that K repeaters are uniformly distributed on each NoC channel. This is useful to gain an understanding of the various aspects of the problem. In real SoC designs, however, repeaters will likely be inserted only on some long channels. Since at this time we do not have access to a complete design of a real SoC, we created a case study based on a 12-node *Spidergon* NoC with URT. We then randomly distributed a total of $K_{tot} = 18$ repeaters along the channels, with the constraint of having at most $K_{max} = 2$ repeaters on a single channel. Specifically, for each of the following experiments we

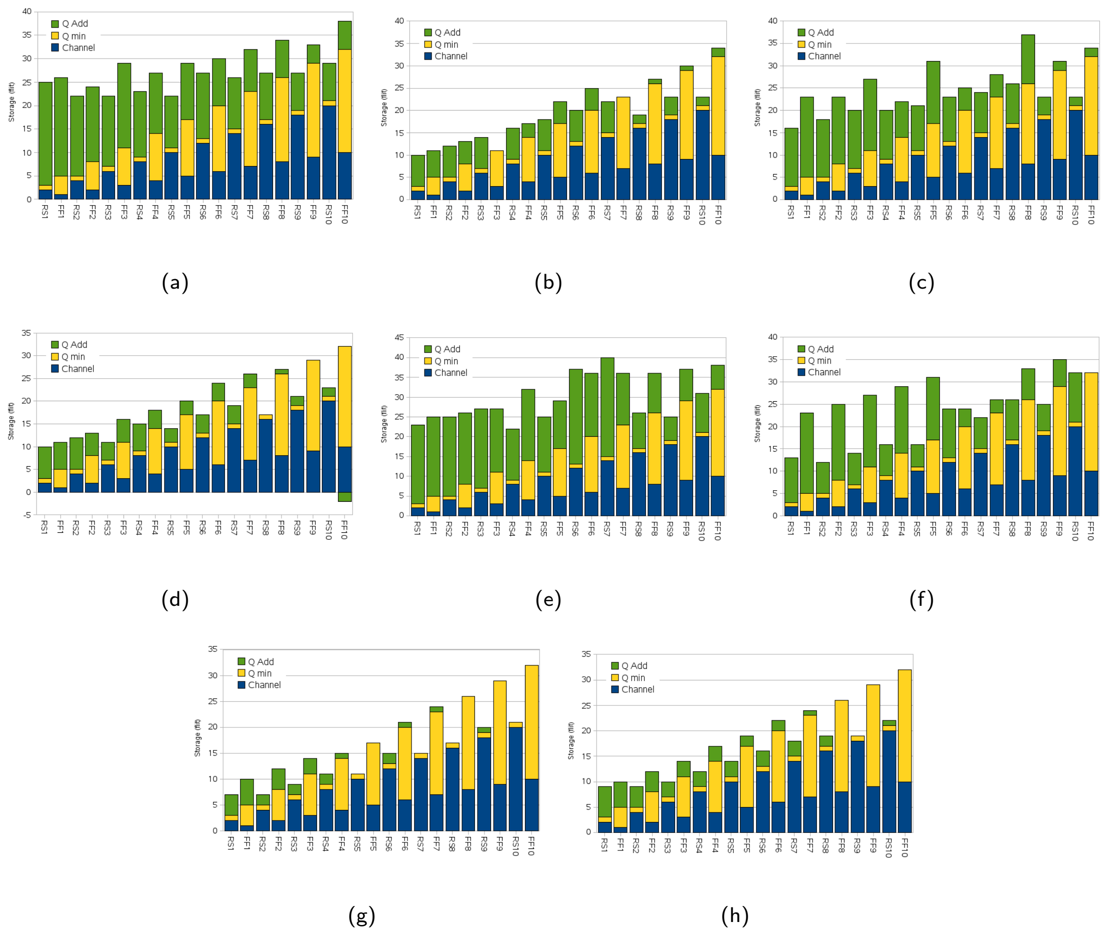


Figure 7.7: RS-repeaters with ack/nack vs FF-repeaters with credit based: breakdown of channel total storage required to obtain the minimal latency in a non-saturated NoC for: *Spidergon* with (a) 4RTF, (b) MPEG4, (c) VOPD, (d) URT, and 2D Mesh with (e) 4RTF, (f) MPEG4, (g) VOPD, (h) URT.

set $K_{tot} = 9$ and $K_{max} = 2$, before generating 15 distinct random configurations.

Figure 7.8(a) shows the results obtained when each channel has a fixed amount of storage $S = 8$ slots, which are distributed between the repeaters and the downstream queue depending on its value of K and the type of repeater. In all these scenarios the RS-systems slightly outperforms the FF-system. The cases where the difference between the two systems is more evident are when repeaters are deployed along the busiest channels. In these cases in fact flip flop use the available storage as repeater while relay stations use the memory slots also as storage buffers.

Here we set $Q = 6$ in the FF case and $Q = 2$ in the RS case. In this case FF systems outperformed the RS-based in a significant way. As seen also in FF in fact had the advantage to have a deep queues even in the cases where the channels did not need them.

Figure 7.8(b) considers the case where all routers, regardless of the number of repeaters on the channels, have the same fixed input queue set to $Q = 2$ for RS systems and $Q = 6$ for FF ones. In this case, the FF-systems outperform the RS-systems significantly at the price of 200% additional queue size. Indeed, as seen for the analysis of Figure 7.6(b), FF-systems sensitively improve their performance as Q increases that is right the case where channels are not segmented by any signal repeater.

In Figure 7.8(c) all routers have the same input queues of 6 flits. Here the RS-systems clearly outperforms the FF-systems as they can use the RS-repeaters also as buffering unit. The RS-system area, however, is bigger by K flit slots. Figure 7.8(d) considers the case where each router has all input queues set as Q_{min}, the minimal queue defined by Table 7.1. For RS we chose $Q = 2$ as this setting gave better performance than the simple minimal queue. Here it is clear that RS outperforms again FF-based systems while requiring a reduced amount of storage S.

Finally, in Figure 7.8(d), the routers' queues are set at the minimum value, i.e. $Q_{min} = 2 + (2 \cdot K)$ for FF-systems and $Q_{min} = 2$ for RS-systems.

Here the RS-systems clearly outperform the FF-systems, which are very sensitive to small values of Q as shown by Figure 7.6, While RS-systems behave essentially like in Figure 7.8(b) (in fact the setting is the same), FF-systems have a major performance degradation due to the reduction of the queue to the functional minimum.

Finally, the bar diagrams in Figure 7.9 report the results of a similar experiment as in Figure 7.8 but using the URT application instead of MPEG4. The trends are the same, but the difference of performance between FF- and RS-systems are more marked. The reason is that packets have a longer average path length (2.09 hops

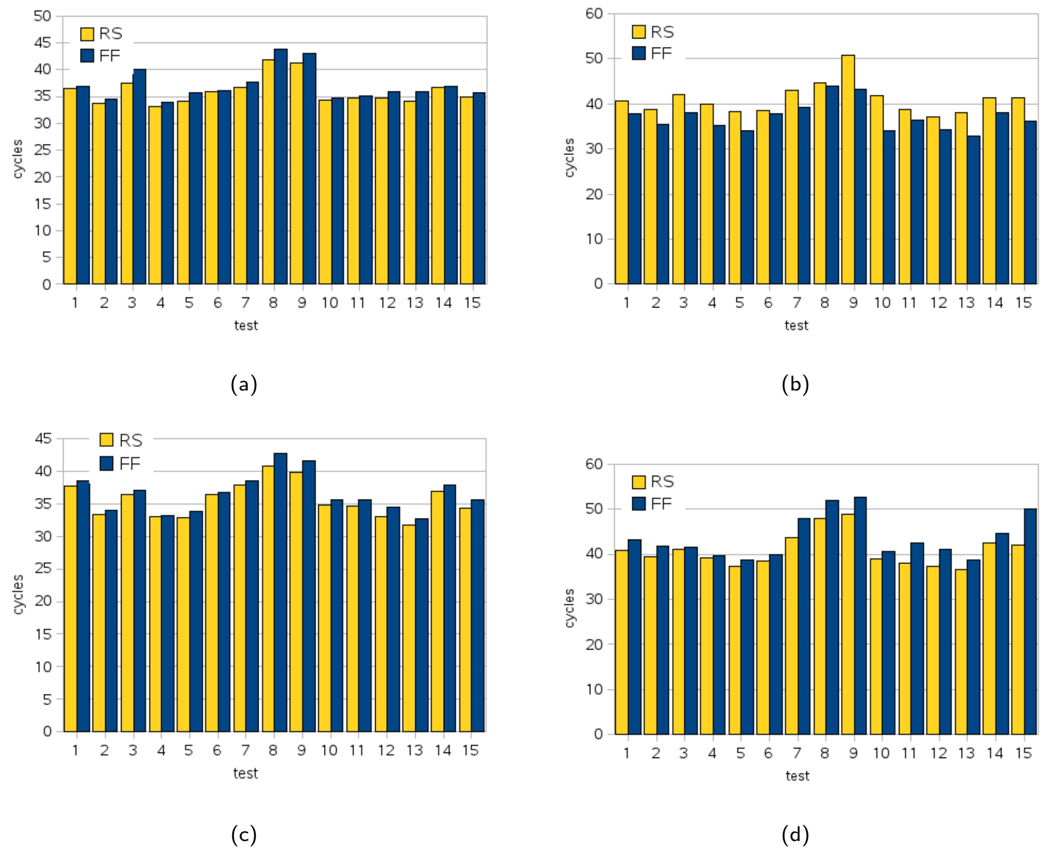


Figure 7.8: Round trip time of a *Spidergon* NoC with MPEG4 and various scenario of repeaters' random deployments, with: (a) channel storage is set to $S = 8$, (b) $Q = 6$ for FF-systems and $Q = 2$ for RS-systems, (c) $Q = 6$ for both FF- and RS-systems, (d) Q changes depending on the value of K.

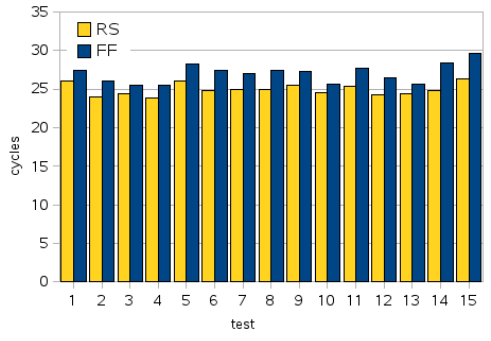
with respect to 1.17 hops) in the URT pattern, thus enhancing the impact of channel pipelining.

We define the *latency speed-up* as:

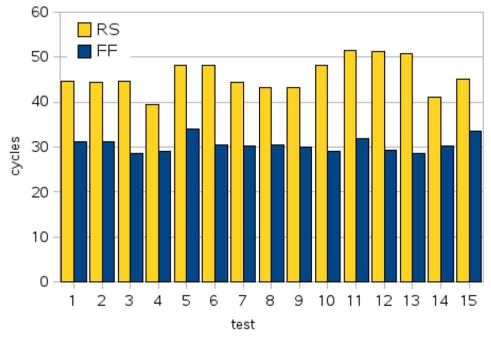
$$LS = \frac{L_{FF}}{L_{RS}}$$

We estimate the ratio of the area of a FF-subsystem and RS-subsystem as

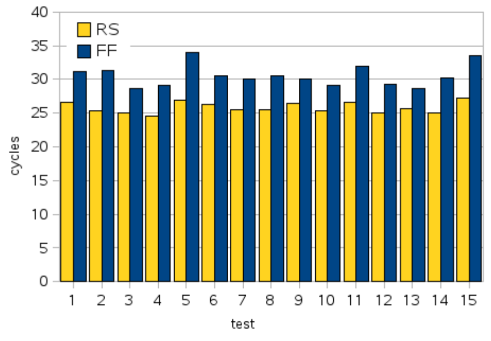
$$AR = \frac{\sum_{0 \leq i < C} Q_i(K_i) + K_i}{\sum_{0 \leq i < C} Q_{RS} + (2 \times K_i)}$$



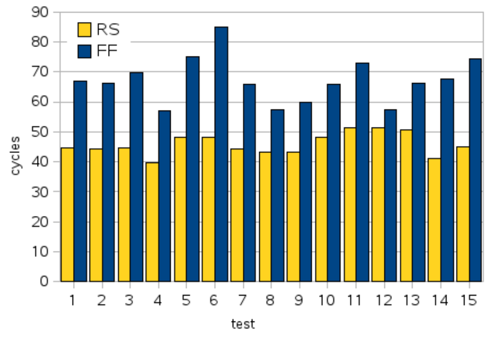
(a)



(b)



(c)



(d)

Figure 7.9: Average packet latency of a *Spidergon* NoC with URT and various scenario of repeaters' random deployments, with: (a) channel storage is set to $S = 8$, (b) $Q = 6$ for FF-systems and $Q = 2$ for RS-systems, (c) $Q = 6$ for both FF- and RS-systems, (d) Q changes depending on the value of K.

where C is the number of channels in the network, K_i is the number of repeaters on the $i-$th channel, $Q_i(K_i)$ is the router queue size as function of the FF-repeaters deployed on the channel and Q_{RS} is the size of the router queues in a RS-system.

Table 7.3 reports the results on latency speedup and area savings obtained by using RS-repeaters instead of FF-repeaters for various queue sizes: specifically we vary Q_{RS} between 1 and 3 for a RS-system while we consider $Q_{FF} \in [1, 2, 4, 6, Q_{min}]$ for the corresponding FF-system. Recall that Q_{min} depends on the number K of repeaters segmenting the channel feeding the queue. First, notice how for low values of $Q \in [1, 2]$ the latency speedup is huge (higher than $100\times$) with relatively minor

Q_{FF}	Q_{RS}					
	1		2		3	
	LS	AR	LS	AR	LS	AR
1	258.73	0.75	412.45	0.50	518.18	0.38
2	151.67	1.25	219.41	0.83	298.15	0.63
4	7.83	2.25	11.46	1.50	16.03	1.13
6	0.41	3.25	0.67	2.17	0.84	1.63
Q_{min}	0.93	1.75	1.55	1.17	1.87	0.88

Table 7.3: Packet latency speed-up and area gain of RS-systems vs. FF-systems for *Spidergon* with URT as function of the router input queue size Q.

area penalty (and indeed with a 25% gain when $Q_{RS} = 1$ and $Q_{FF} = 2$). This confirms again the bad consequences of having queues with size $Q < Q_{min}$ in an FF-system, as explained in Section 7.2. For $Q_{FF} = 4$, the RS-system outperforms always the FF-system both in terms of penalty and area occupation. Instead, for $Q_{FF} = 6$, the FF-system has better performance than any RS-system but at much higher costs in terms of area. Finally for $Q_{FF} = Q_{min}$, we have that a RS-system with $Q_{RS} = 1$ is 8% slower while being 75% smaller. Also, it is enough to set $Q_{RS} = 2$ to obtain both a gain of 55% in performance and 17% in area. In conclusion, *to use RS-repeaters (a) often leads to a gain in both area and performance while (b) a loss in either figure is always accompanied by a major gain in the other.*

"Fat" Relay Stations

In our final experiment we use "fat" RSs, which are obtained by replacing the main and auxiliary FFs with queues that in case of back-pressure can store more than two flits. For a fixed channel total storage S, if K fat RSs provide B_{RS} units of flit buffering each, the size of the downstream queue is reduced to $B = S - (K * B_{RS})$. The experimental results in Figure 7.10 confirm the theoretical result given in [CXSP04]: when using RS-repeaters, the particular distribution of flit buffering on the channel does not really affect the performance. In other words, comparing this result with the previous experiments shows that the critical performance efficiency is obtained by using RS-repeaters that are flow-control-aware, while increasing their size beyond the minimum $B_{RS} = 2$ doesn't have a big impact. In fact, keeping the size of the router queue to a small value and distributing minimum-size RS-repeater is the strategy that will provide the highest flexibility from a physical design perspective. However, once the number of RS-repeaters that are necessary for wire pipelining a given channel is determined, designers do have the freedom to decide whether and where to add flit buffering.

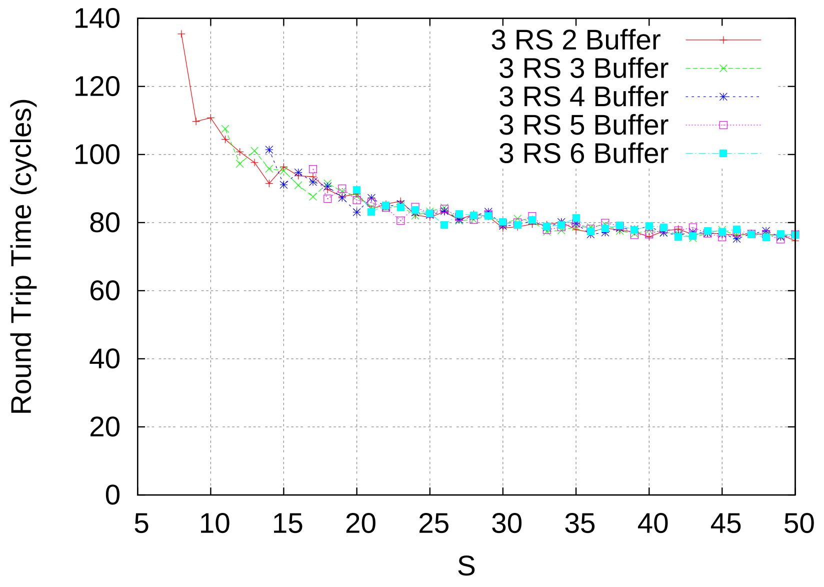


Figure 7.10: Round Trip Time of a "fat" Relay Station in function of S.

7.6 Conclusions

Given the particular constraints imposed by nanometer technologies, we propose *distributed flit-buffer flow control* for NoC design as a method that combines the simplest form of ack/nack protocol with the distribution of relay stations on the channels. Relay stations act both as clocked repeaters to pipeline the channels and as flit buffers to enable a distributed implementation of flow control. Consequently, they provide precious flexibility during the physical design of the NoC by allowing designers to use smaller routers and to manage long wires better. Experimental results, including semicustom implementations and system-level simulations, show that across many scenarios:

- for an equivalent amount of storage capacity a RS-based NoC performs better than a FF-based NoC;
- a RS-based NoC needs less storage capacity to deliver the same performance as a FF-based NoC.

Future work includes the study of the proposed approach in combination with virtual-channel flow control.

Chapter 8

Link Management

8.1 Introduction

Packet-switched networks-on-chip (NoC) have been proposed as an alternative solution to standard bus-based interconnects to address the global communication demands of future chip-multiprocessors (CMP) and system-on-chip (SoC) [BM02, DT01, GvMPW02, HJK+00]. While these communication demands continue to grow as more cores are integrated on a chip, the on-chip power-dissipation budget is expected to remain very limited due to packaging constraints. Hence, the challenge is not only to design NoCs that can deliver high-bandwidth at low latency for inter-core communication, but also to make sure that this is done in a very power-efficient way [O+07].

In a packet-switched NoC the network nodes are connected by optimized point-to-point communication channels that are organized in a regular topology and shared by multiple packets flowing from one processing core to another [MB06]. A NoC node is either a network interface, which is used by a processing cores to inject/eject packets to/from the network, or a router. Network interfaces and routers implement the packet routing and flow-control protocols. In the route to their destinations packets need to compete for NoC resources (e.g. channel bandwidth and router buffers) on a hop-by-hop basis.

Many NoCs proposed in the literature rely on *worm-hole (WH) flow control*, a

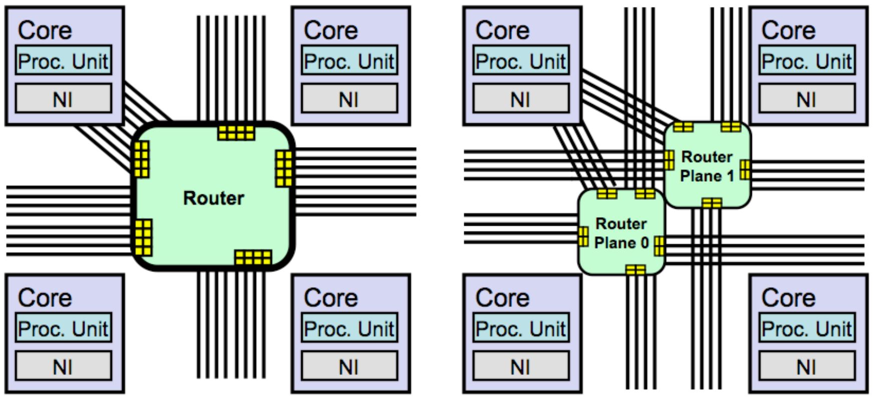


Figure 8.1: A standard NoC (left) and a Multi-Plane NoC for $p = 2$ (right).

mechanism to allocate these resources in units of *flits* (flow-control digits) rather than packets. A packet is decomposed in a sequence of flits and when the *head flit* arrives at a node, it must acquire three resources before it can be forwarded to the next node along its route: a *virtual channel* that holds the state needed to coordinate the handling of the flits for the entire packet, one flit buffer and one flit of channel bandwidth [DT04]. The subsequent *body flits* only need to acquire the last two resources while the last body flit (the *tail*) releases the virtual channel. By associating several virtual channels (channel state plus flit buffers) with a single physical channel, *virtual channel (VC) flow control* avoids the blocking problem (caused by the fact that a channel is owned by a packet, but buffers are allocated on a flit-by-flit basis) and optimizes the use of the channel bandwidth. Peh and Dally showed that a VC router can deliver 25-40% throughput improvement over a wormhole router [PD01].

Both WH and VC flow controls are appealing for NoC design because they require less buffering space in the routers than packet-buffer flow control methods. Indeed, in comparison with macro-level networks, NoCs must be designed while keeping in mind that, in a chip, buffers are generally more expensive resources than wires, both in terms of area and, to a certain degree, power. VC flow control aims at improving performance by investing in more flit buffering space to better exploit the available channel bandwidth. After noticing that the packet energy/delay in an

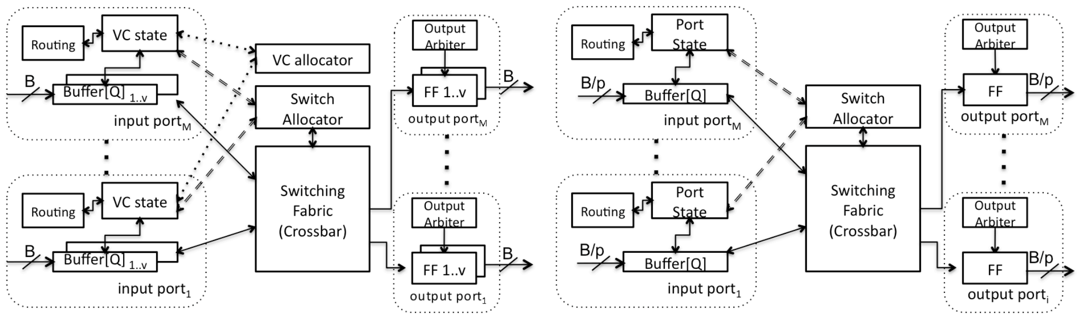


Figure 8.2: Block diagrams of a VC router (left) and the simpler router used in the MP NoC (right).

NoC is dominated largely by contention at intermediate routers (thus resulting in a high router-to-channel energy/delay ratio) Kumar *et al.* have recently proposed *Express Virtual Channels (EVCs)* [KPKJ07]. This is a novel flow control and router microarchitecture design that allows packets to virtually bypass the entire pipeline of intermediate routers along pre-defined virtual express paths between pairs of nodes and, therefore, to approach the energy/delay of a dedicated wire interconnect.

Contributions. Instead of building NoCs with complex VC routers, we propose the *Multi-Plane* (MP) approach to NoC design. We start from a standard NoC and we split it into a number p of parallel sub-networks (the *planes*) such that the aggregated point-to-point channel parallelism and router buffering capacity remains constant. Fig. 8.1 shows an example of a MP NoC with $p = 2$ where the standard network (both router and wires) are split into two parallel and independent network planes. In the sequel, we study the cases of MP NoCs with $p = 2$ and $p = 4$ and we compare them with a single plane NoC without VCs as well as single-plan NoCs with two and four VCs. We present a complete comparative analysis that includes: (a) the application of analytical models for router area and delay, (b) the use of the power/performance ORION simulator [WZPM02], (c) the register-transfer level (RTL) design, logic synthesis and technology mapping of various routers, and (d) an extensive set of system-level simulations. Combined our results show that MP NoCs represent a simple and effective alternative to virtual channels for low-power NoC design.

8.2 Related Work

Balfour and Dally presented a comprehensive comparative analysis of NoC topologies and architectures in [BD06b], where they also discuss the idea of duplicating certain NoC topologies, such as Mesh and *CMesh*, to improve the system performance. Our work differs from this analysis because instead of duplicating the NoC we actually *divide* it in a number of sub-networks while keeping the overall amount of wire and buffering resources constant. For NoCs, Kumar *et al.* presented the full-custom design of a VC router that can deliver a single-cycle no-load latency at 3.6GHz clock frequency while achieving a peak switching data rate of 4.6Tbits/s per node [KKS+07]. Ogras *et al.* extended the VC concept by adding extra physical *long-range links* [OM06a]. Matsutani *et al.* proposed slow-silent VCs to reduce both static and dynamic power consumption caused by VCs [MKWA08]: while relying on voltage and frequency scaling to reduce dynamic power leakage, they also use a power-gating technique to reduce the standby power of the VCs when they are not active. They show that 58.2% of total power is saved by using slow-silent VCs under the assumption of Uniform traffic and four VCs at 45 million flits per second per core.

In the RAW processor four separate and independent NoCs are used: two NoCs are statically routed and two are dynamically routed [TKM+02]. The reason for implementing physically separated networks and using different routing schemes is to accommodate different types of traffic in general purpose systems. In our approach we do not build heterogeneous networks. Instead, we partition a single NoC in multiple ones to achieve power efficiency and area advantages by simplifying the router design.

Noh *et al.* proposed a multi-planes-based design for a VC-enabled router [NNJC06]: the internal crossbar switch is replaced with a number of parallel crossbars (planes) that increase the flit transfer rate between input and output queues. The result of this design is a router with a simpler hardware design that performs better than a single-plane router with a larger number of VCs. Differently from our approach,

Terms	Definitions
B	flit width of single-plane NoC
Q	size of router input buffer
p	number of physical channels
v	number of virtual channels
b	flit size of a plane in multi-plane NoC ($= \frac{B}{p}$)

Table 8.1: NoC parameters used in our comparative study.

they mantain the flit-width constant as they scale the number of additional lanes.

In summary, to the best of our knowledge this is the first article that proposes *multi-plane partitioning* for NoC design.

8.3 Multi-Plane Partitioning of a NoC

Fig. 8.2(left) shows the block diagram of a classic 5-port VC router that can be used in a 2-D Mesh network. Each input/output port is connected to a physical channel that as a data parallelism of B bits which matches the flit size. For a VC router supporting v virtual channels, each input port is equipped with: (1) a routing logic block that determines the destination port for each packet based on the information contained in the head flit and the specific routing algorithm (e.g. XY routing); (2) a set of v queue buffers, and (3) a VC control block that holds the state needed to coordinate the handling of the flits of the various packets. Each queue buffer has a size Q corresponding to the number of incoming flit that can be stored when they cannot be forwarded because the destination port is busy forwarding another packet. Every output port is equipped with an *output arbiter* that manages the allocation of the port across the multiple packets arriving at the input ports by scheduling the forwarding order among these. Typically NoC routers do not have buffers on the output ports. If VC flow control is used (i.e. $v > 1$), the output arbiter manages the allocation of every VC associated to that output, while a *VC allocator* block arbitrates the matching between input and output VCs. The packet forwarding happens over a *switching fabric* that connects every input to every output port.

This is configured dynamically on the basis of the input routing and the output arbitration. Thank to the limited number of ports, the switching fabric of an NoC router is typically realized as a *crossbar*, a structure that has also the advantage of being regular, thus simplifying the layout.

The basic idea of our proposed multi-plane approach to NoC design is to partition the resource of a traditional NoC in p parallel simpler sub-networks (the *planes*) that do not use VC flow control and such that the aggregated point-to-point channel parallelism and router buffering capacity remains constant. Hence, for each physical channel in the original NoC having a bit parallelism B, the p-plane MP NoC will have p independent physical sub-channels with parallelism $b = B/p$. The p independent channels are part of p parallel meshes, each containing a simpler router.

In this scenario, the processing unit (PU) in each core is connected to p routers, one per plane, through a network interface (NI). For instance, Fig. 8.1 (right) shows the interface of a PU with the multi-plane NoC, for $p = 2$. The NI manages the PU access to the NoC in terms of reading/writing data from/to the memories, sending/receiving the packets, and handling the back-pressure received from the NoC in case of congestion. The PU generates/consumes the actual data transfer requests. When a PU needs to transmit a packet, its NI chooses which plane to use based on a simple allocation policy. The packet is forwarded towards the destination over the assigned plane. More than one packet can be sent simultaneously, each on a different plane. In the same way more packets can be received at the same time on different planes. However, the length of a packet, i.e. the number of flits, is inversely proportional to the number of planes due to the reduction of the flit width.

While the idea of MP NoCs can be applied to various NoC topologies, we focus on 2-D Meshes with dimensional (XY) routing [DT04] because they lead to simple and efficient VLSI implementations. For the same reason, each router in a MP NoC implements a simpler worm-hole (WH) flow control mechanism. Fig. 8.2(right) shows the block diagram of the router of a p-plane MP NoC. Essentially, this can be seen as a simplification of the previous VC router where WH flow control is used instead of VC flow control (in the sequel we denote this design choice by setting the parameter $v = 1$) and where each input port has a single queue buffer of size Q and width b.

Fig. 8.1 summarizes all the most important NoC parameters that are used in the following comparative analysis.

8.4 Model-Based Comparative Analysis

In this section we present a comparative analysis of a MP NoC with respect to traditional single-plane NoCs from the perspectives of area occupation, maximum logic delay, and power dissipation. Since the number of channel wires remains constant across the various NoC designs that we consider, we focus on analyzing the impact of the various router designs as we vary the parameters b, v and Q of Table 8.1. This analysis is based on the analytical model for router delay proposed in [Peh01a] and those for area and power used in the ORION tool [WZPM02].

8.4.1 Area Model

Following the modeling approach used for ORION, we approximate the area of the entire router as the sum of the area of the input buffers ($Area_{buffer}$) plus the area of the crossbar ($Area_{xbar}$), while considering the area of the arbitration and routing logic negligible. In particular, assuming that the channel bit parallelism (the *phit* width) is equal to the flit width B, and that a buffer is able to store Q flits, we have:

$$Area_{router} = \#ports \cdot Area_{buffer} + Area_{xbar}$$

If we implement the buffer as a register file of SRAM cells, we have:

$$Area_{buffer} = length_{bitline} \cdot length_{wordline}$$

and if we consider two bitlines per bit and a wordline per buffer location:

$$length_{bitline} = Q \cdot (height_{memorycell} + 2 \cdot pitch)$$

$$length_{wordline} = B \cdot (width_{memorycell} + 4 \cdot pitch)$$

Similarly, considering the implementation of the crossbar as a matrix of horizontal and vertical wires using pass transistors at the junction point between an input and output, we have:

$$Area_{xbar} = length_{inline} * length_{outline}$$

with:

$$length_{inline} = \#ports \cdot B \cdot (width_{xbarcell} + pitch)$$

$$length_{outline} = \#ports \cdot B \cdot (height_{xbarcell} + pitch)$$

In summary:

$$Area_{router} = \alpha \cdot B + \beta \cdot B^2$$

where α, β include all the parameters depending on the technology and on the design, i.e. cell sizes, wire pitch, buffer depth and number of ports.

According to this model the total area of the router depends quadratically on the flit size B (due to the crossbar layout). This relation remains valid as we scale the technology process (assuming classic CMOS scaling). The number n of ports in a router mainly depends on the NoC topology (it is five for a regular 2-D Mesh). The buffer depth Q can be dimensioned as a trade off of power/area versus performance, i.e. the deeper the buffer the better the performance of the network, but the higher the overhead in terms of power dissipation and area. The value of B must be dimensioned to guarantee the target line rate on the router-to-router channels. Recall that the *line rate* is the amount of data transferred in a clock cycle, i.e. $L = B \cdot f_{clk}$, where f_{clk} is the frequency of the clock at which the NoC operates.

We saw that the dependency of the router area is quadratic with respect to B. To reduce B means to reduce the line rate, with a degradation of the overall NoC capacity of transferring data.

	1 plane w/ $size_{flit} = B$	p planes w/ $size_{flit} = B/p$
$Area_{xbar}$	$M \cdot B^2$	$M \cdot p \cdot (\frac{B}{p})^2$
$Area_{buffer}$	$N \cdot B$	$N \cdot p \cdot \frac{B}{p}$

Table 8.2: Router area occupation as a function of the number of planes and flit size.

As explained in Section 8.3 in our approach each router with flit size B of a single-plane NoC is replaced with p parallel and independent routers, each with flit size b. Table 8.2 reports the analytical formulas to characterize the area occupation of a router as a function of the number of planes and flit size. Comparing the set of p routers of the p-plane NoC with a router of the single-plane NoC, we note that the aggregate line rate, i.e. the number of wires, of the two NoCs is the same. The area due to the buffer remains the same, but the crossbar area is reduced by a factor $p^2/p = p$ (Table 8.2).

8.4.2 Power Model

We used the accurate model that is available in ORION also for estimating the router power dissipation. This model breaks down the power dissipation along the main router components: the switching activity of the buffers and the crossbar, and the leakage (static power), which becomes increasingly more important with the scaling to nanometer technology processes.

Table 8.3 collects the power dissipation values obtained using Orion for a 5×5 router with $B = 256$, which we take as our reference design, for different values of input storage Q. The dissipation of a router in case of multiple planes is obtained by multiplying the power dissipation of the single router times the number of planes p. We considered NoC with $p = \{1, 2, 4\}$ and compared it with a single plane NoC with two and four virtual channels ($p = 1, v = \{2, 4\}$). In addition we report data for three different technology processes: 100, 70 and 50nm. This analysis shows that adding more planes gives an overall power saving while using VCs leads to an

Q	2			4			8			16			32		
Tech $[nm]$	100	70	50	100	70	50	100	70	50	100	70	50	100	70	50
$p = 1$		1			1			1			1			1	
$p = 2$	0.68	0.74	0.84	0.68	0.75	0.85	0.69	0.76	0.86	0.71	0.77	0.87	0.74	0.81	0.90
$p = 4$	0.68	0.62	0.77	0.68	0.63	0.78	0.69	0.65	0.80	0.71	0.68	0.83	0.74	0.73	0.87
$p = 1, v = 2$		N/A		1.50	1.53	1.62	1.43	1.52	1.59	1.45	1.48	1.54	1.40	1.42	1.45
$p = 1, v = 4$		N/A			N/A		1.48	1.52	1.59	1.46	1.48	1.53	1.41	1.42	1.45

Table 8.3: Power dissipation ratio (w.r.t. 1-plane WH reference NoC) for different values of p, v and different technologies, with $B = 256$.

higher power consumption.

8.4.3 Delay Model

Li-Shiuan Peh has proposed an accurate hierarchical model for estimating the critical path delay for WH and VC routers and comparing the two architectures [Peh01a]. The critical path of each main block of a router is modeled as a sequence of logic gates and measured as a multiple of the basic delay of an inverter feeding 4 other inverters (the fanout-of-4 unit τ_4). Following Peh's method we quantify the advantages of having a MP NoC with small-flit VC-free routers over a traditional NoC with large-flit VC routers under the assumption of using XY-dimensional look-ahead routing [M.G96]. In particular, the latter allows computing the routing destination one hop in advance, thus removing the routing logic from the critical path. We also assume that routers have zero-load latency of one clock cycle.

In a 5×5 WH router, the allocation of the switching fabric is performed in $9\tau_4$. Regarding the crossbar traversal, the time needed by the data to propagate from the input to the output grows accordingly to the size of the crossbar itself, depending on the flit-size (Section 8.4.1). In a 5×5 VC router, the allocation of the switching fabric is performed in $12\tau_4$ if $v = 2$ or $15\tau_4$ if $v = 4$. However, this stage follows the VC allocation, that takes $14\tau_4$ if $v = 2$ or $18\tau_4$ if $v = 4$. Since the wire delay, which accounts for the main part of the propagation time in the crossbar, is difficult to model, we follow Peh's approach [Peh01a] and assume that the crossbar traversal is about $20\tau_4$ for any flit-size.

In summary, a WH router has a critical path of $29\tau_4$. Using 2 VCs, makes the critical path grow up to $46\tau_4$, with a maximum clock frequency reduction of roughly 37%. Using VC leads to a clock frequency reduction of 45% (critical path $53\tau_4$) with respect to the WH router.

Notice that this analysis is conservative for two reasons:

- using MP logically results in an shorter critical path even without considering VCs (but just large-flit WH routers) because a WH router crossbar traversal decreases with the area reduction of the crossbar itself;
- the lower is the crossbar traversal time, the higher is the weighted advantage of using simple and fast logic for arbitration, i.e. WH as opposed to VC.

8.5 Synthesis-Based Comparative Analysis

We validated the model-based comparative analysis of the previous section with a set of experiments based on the logic synthesis and technology mapping of a set of routers (and small NoCs) that we obtained by varying the values of the parameters of Table 8.1. In order to derive the RTL designs we took advantage of the NoC Emulator (NoCem) [Ope]. In particular, the NoCem router is the combination of a simple buffer-less switching fabric equipped with arbitration and a bidirectional channel for each port, containing the storage and the logic needed to manage VC flow control and buffer accesses. We performed two series of synthesis experiments using Synopsys Design Compiler and a $90nm$ standard-cell library:

1. we synthesized the channel/buffering logic and the switching fabric separately and then we combined 2.5 channels (because of bidirectionality) with the switching fabric to estimate the router's area and power;
2. we synthesized a 2×2 Mesh of routers and divided the measured values by four.

Q	2	4	8	16	32
$p = 1$	1	1	1	1	1
$p = 2$	1.05	1.06	1.04	1.05	1.06
$p = 4$	1.19	1.14	1.11	1.12	1.12
$p = 1, v = 2$	N/A	1.37	1.22	1.13	1.07
$p = 1, v = 4$	N/A	N/A	1.62	1.37	1.21

Table 8.4: Area occupation ratio (w.r.t. 1-plane 256-bit reference NoC) for different values of p and v, with $B = 256$.

We cross-checked the two series of experiments to confirm that they returned the same trends. The experimental results for area, power and delay are presented next.

Area. Table 8.4 reports the are occupation of our reference design, i.e. 5×5 router with $B = 256$, for different values of input storage Q. We considered a p-plane NoC and multiply the area of a router with flit size B/p by the number of planes. We considered NoC with $p = 1, 2, 4$ and compared it with a single plane NoC with two and four virtual channels ($p = 1, v = \{2, 4\}$). The VC allocation algorithm implemented in the router is a simple round-robin, which offers the smallest possible area occupation. Since we used static allocation of buffers to VCs with input storage capacity Q for a VC-router, the storage capacity per single VC is $Q_{VC} = Q/v$. This gives a more fair comparison because all the NoC have the same number of physical I/O wires and the same amount of storage resources, i.e. the total number of flip-flops in the input buffers. On the other hand, this does not allow us to apply all the values of Q to the VC-routers because a size of at least $Q_{VC} = 2$ is needed per each VC buffer.

The results do not completely validate our previous analysis based on the ORION area model (Section 8.4.1). The main reason is that an RTL-designed crossbar is generally synthesized as a collection of multiplexers, and it scales linearly with B, even though no layout has been taken into account. In particular, the use of $p > 1$

Q	2	4	8	16	32
$p = 1$	0.52	0.53	0.52	0.52	0.52
$p = 2$	0.59	0.57	0.56	0.56	0.56
$p = 4$	0.70	0.68	0.66	0.64	0.63
$p = 1, v = 2$	N/A	0.73	0.64	0.60	0.56
$p = 1, v = 4$	N/A	N/A	0.88	0.71	0.64

Table 8.5: Area occupation ratio (w.r.t. 1-plane 256-bit reference NoC) for different values of p and v, with $B = 128$.

planes gives a disadvantage in terms of area consumption. Meanwhile, it is not possible to neglect the area contribution due to the management of the VCs, which has a noticeable impact particularly for small values of Q. Indeed, when the amount of storage that is possible to reserve at each input router is small ($Q < 8$), it is convenient to organize this storage on a multi-plane fashion instead of sharing it among multiple VCs. Table 8.5 shows the same trends for a 128-bit NoC.

Power. The reports of the logic synthesis tool present power dissipation trends similar to those for the area occupation discussed above. Specifically, in most cases the use of multiple planes increases the power consumption of the overall system by less than 15%. The use of VCs, instead, leads to an increment of more than 30% in terms of power dissipation for low values of Q. This increment decreases when Q grows. Hence, these results confirm the trend about the use of VCs: they help improving the NoC performance at the price of higher power dissipation of the NoC. On the other hand, the results disagree with the estimations obtained through the ORION model, which suggest that the use of more planes is effective from the power-dissipation viewpoint.

Delay. Through the synthesis of the RTL NoCEM router we also collected data about the critical path for all the proposed configurations. The critical path of a router depends inversely on B and v. In the MP NoC configurations reducing B by a factor of $p = \{2, 4\}$ leads to a delay improvement of $1 - 10\%$. Further, a MP NoC with $p = 2$ planes can run at a clock frequency that is 15-25% higher than an NoC with $v = 2$ VCs while a MP NoC with $p = 4$ planes can run at a clock frequency that is up to 25-35% faster than an NoC with $v = 4$ VCs. These results generally confirm the speedup trends estimated by the analytical model of Section 8.4.3.

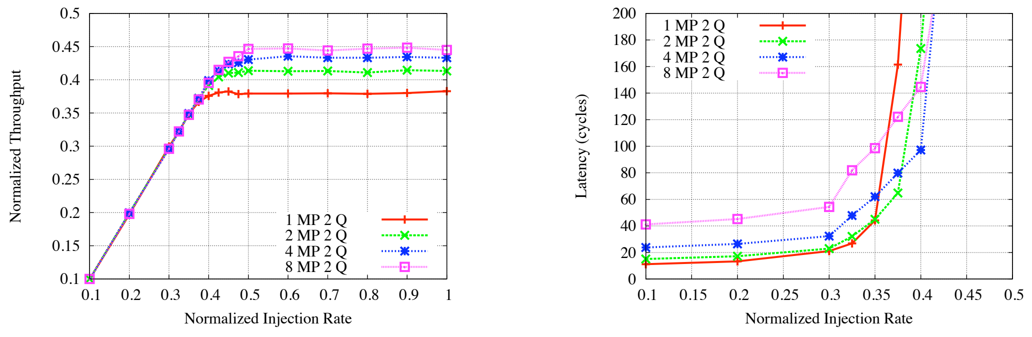
8.6 System-Level Simulations

We developed an event-driven simulator including detailed models of the parameterized NoC components, such as routers, NIs and high-level models for the PUs that are the sources of network traffic. Each NI is connected to one or more routers depending on the number of NoC planes. The flit width of the single-plane base configuration is set to $B = 256$ bits.

We analyzed the multi-plane approach under the Uniform, Tornado, Transpose and 4-HotSpot traffic patterns. In Uniform traffic each PU chooses a randomly peer and sends a packet of a fixed size. In Tornado traffic each $\langle x, y \rangle$ node exchange packets only with $\langle x + (\lceil k/2 \rceil - 1)\%k, y \rangle$ node where k is the number of nodes in the x dimension in our experiments. For Transpose traffic in a 2D Mesh instead each $\langle x, y \rangle$ node communicates only with the node $\langle y, x \rangle$. Finally, 4-HotSpot (4HS) defines four nodes who are uniformly selected by all the other nodes as destinations for their traffic.

Fig. 8.3 (a) shows the throughput per node measured assuming $p = \{1, 2, 4, 8\}$ under Uniform traffic. By partitioning a NoC in multiple planes we increase the parallelism of the system so that each NI can send and receive more than one (smaller) flit per clock period. Different packets assigned to different planes can be processed in parallel during a single period. With a high number of planes, i.e. $p = 8$, we reach a 17% improvement of the maximum throughput. The result for other traffic patterns are similar as shown in the first row of Table 8.6.

The improvement of throughput comes at the expense of latency: using p multiple planes, each NI has a channel width b that is p times narrower than the original system. As a consequence, each packet traveling on a multi-plane network is made of



(a) (b)

Figure 8.3: Throughput (a) and Latency (b) with $Q = 2$ under Uniform traffic.

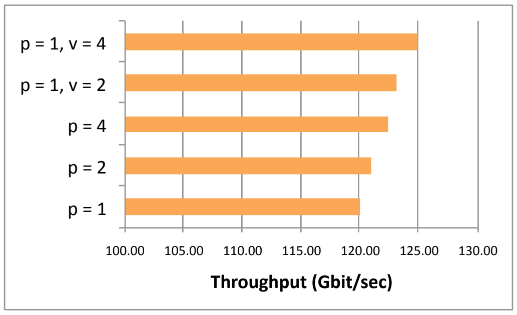
p times more flits than in the single-plane NoC, e.g. a packet of 1 Kbit is composed by 4 flits when $b = 256$ bits or 8 flits when $b = 128$ bits.

Fig. 8.3 (b) shows the latency measured under Uniform traffic pattern. Notice that under different injection rates the reference NoC with $p = 1$ has a latency lower than the MP NoCs only for low traffic loads. As the average load increase, the MP NoC can handle better the higher traffic volume, thus reducing the overall system latency and raising its maximum throughput. We obtained the same trends with the other traffic patterns but we omit the plots for brevity.

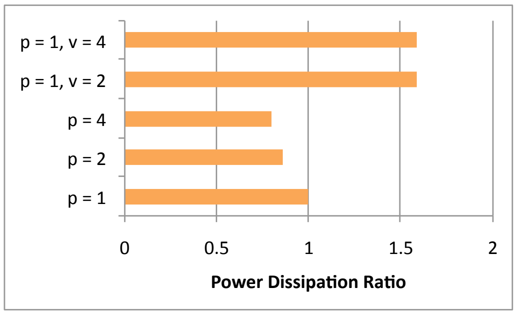
Table 8.6 reports the maximum throughput achieved by the MP NoCs as function of the input buffer size Q and the number of planes p, for all the proposed traffic patterns. Generally duplicating the number of planes delivers a throughput improvement. We also notice an improvement for small values of Q, i.e. shorter than the packet length, which we set to $1Kbit$, i.e. 4 flit of 256 bit. Indeed, when the router input buffers are small and the injection rate is high, packets can wait in the NI output channel blocking all the following packets that are generated by the same NI. A MP NoC instead parallelizes the traffic on multiple independent networks thus reducing the *Head-of-Line* blocking. As Q increases, this effect is reduced, because more packets can be stored in the larger router's buffers. This is confirmed by the results of Table 8.6 for Uniform and 4-HotSpot. Instead, Transpose and Tornado

Traffic	Uniform				Transpose				Tornado				Hotspot			
p	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
$Q = 2$	0.38	0.41	0.43	0.45	0.45	0.48	0.50	0.50	0.44	0.47	0.48	0.49	0.29	0.31	0.32	0.32
$Q = 4$	0.43	0.44	0.45	0.45	0.47	0.50	0.51	0.52	0.44	0.47	0.48	0.49	0.30	0.32	0.32	0.33
$Q = 8$	0.47	0.50	0.48	0.47	0.47	0.50	0.51	0.52	0.44	0.47	0.48	0.49	0.32	0.33	0.33	0.33
$Q = 16$	0.52	0.53	0.54	0.50	0.47	0.50	0.51	0.52	0.44	0.47	0.48	0.49	0.34	0.35	0.34	0.34
$Q = 32$	0.54	0.57	0.57	0.56	0.47	0.50	0.51	0.52	0.44	0.47	0.48	0.49	0.36	0.36	0.36	0.34

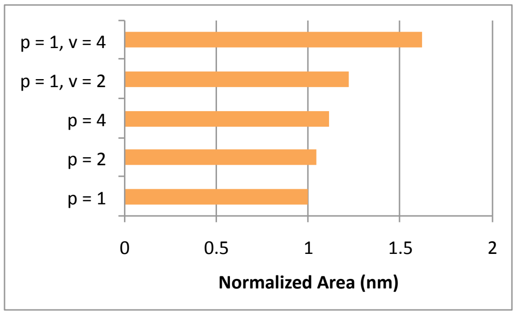
Table 8.6: Normalized throughput as function of Q and p under various traffic patterns.



(a)



(b)



(c)

Figure 8.4: Throughput (a) and Latency (b) with $Q = 2$ under Uniform traffic.

are less sensitive to the variations of Q and do not present an improvement of the throughput. Finally, these results suggest that an improvement is always guarantee by splitting an NoC in a 2-plane NoC.

Finally we compared the MP NoC approach with respect to NoCs with virtual channels from a performance viewpoint by using the performance gain reported in the literature for the latter.

Graph in Figure 8.4 compare throughput and power and area consumption of MP and VC networks when $Q = 8$ slots with 50nm technology. As Figure 8.4(a) shows, MP can actually improve the performance of a system. Nevertheless VC-approach can achieve a higher improvement in line with [PD01] where, Peh shows how an NoCs with four virtual channels ($v = 4$) gives a throughput improvement of up to 40% with respect to a WH NoC. As depicted in Figures (b) and (c) this higher improvement comes at the price of a wider area and increased power consumption.

Hence Multi-Planes can be a energy and cost efficient approach to improve the performance of the system when the SoC is characterized by tight power and area constraints.

8.7 Conclusions

We proposed multi-plane (MP) networks-on-chip as a power-efficient communication infrastructure for multi-core architectures and a possible alternative solution to NoCs based on the use of virtual channels (VC) routers.

First we showed that, independently from the input buffer size, partitioning a single-plane NoC without virtual channels into a MP NoC with small VC-free routers offers important advantages in terms of both throughput and power dissipation with limited loss in terms of latency. From an area occupation viewpoint, we showed that the results depend mainly on the design style of the router switching fabric: if this is implemented as a crossbar the smaller routers in the MP NoC have combined smaller area than the bigger router in the single-plane NoC, while they may be slightly worse if implemented through the logic synthesis of standard cells.

NoCs with VC routers provide higher throughput gains than equivalent MP NoCs but they also have larger power dissipation. Indeed, part of this performance loss can be recovered by trading power dissipation for clock frequency and running the MP NoC with a higher rate frequency because its critical path has a delay that is up to 25-35% smaller. Finally, the MP approach saves area with respect to using

VCs and is well suited for small sized input buffers, i.e. area occupation critical applications.

Future work includes investigating the implementation of traffic distribution policies in the network interface, the combination of heterogeneous routing algorithms across the planes, and the optimization of the MP NoC layout organization.

Chapter 9

Conclusions

In this Thesis we considered the analysis and simulation of protocols and architectures for Networks on Chip.

In Chapter 2 we introduced the Networks on Chip main characteristics and explained motivations making NoC a promising solution for future System on Chip interconnects.

In Chapter 3 we analyzed in detail the Spidergon NoC architecture. We compared it with two classic solutions such as 2D Mesh and Ring and the Crossbar-based Bus used in former Systems on Chip. Simulation-based test reported in this chapter show that the Spidergon NoC is a good tradeoff between cost and performance. In particular when used to support hot spot traffic, the Spidergon architecture can offers performances similar to those of the more expensive 2D Mesh and Crossbar interconnect. In this chapter we also introduced the aEqualized routing algorithm for the Spidergon NoC. AEqualized grants similar performances as the classical aLast and aFirst algorithms but reducing the resources actually used by the system by up to 15%.

In Chapter 4 we discussed the methodology and tools that we used to model and simulate the NoC systems considered in this Thesis. In particular we introduced the SCOTCH partitioning tool used to perform the mapping of a given application on the NoC, Metis, Nauty and Neato used to analyze the mathematical properties of the NoC topologies and finally OMNET++ used to model and simulate a given application on a specific NoC architecture.

In Chapter 5 and 6 we discussed the *message dependent* deadlock and proposed the Ctc end-to-end flow control to solve it. Ctc is a cost-efficient solution developed in collaboration with the ST Microelectronics research lab of Grenoble. We compared it to the well known end-to-end version of *Credit Based* (CB) flow control and we demonstrated that it can reach similar performances as CB with slightly higher delay but sensitively reduced implementation costs. Ctc hence is an interesting solution for the cost-constraint SoC wold.

In Chapter 7: we described the *Time Closure Exception* issue that can be triggered when long-linked NoC run at high clock frequencies. Here we proposed a pipelined-based solution based on the use of Relay Stations. We considered system-level and register-transfer lever (RTL) analysis used to compare both performance and actual chip-area of the proposed solution. Results indicate that RS-based pipeline is both energy and area efficient and also can grant better throughput and latency performances.

Finally in Chapter 8 we discussed a *Multi-Plane* (MP) approach as an alternative to virtual channels. A Multi-Plane based NoC is composed my a set of parallel networks supported by simple and very fast routers. We compared our approach with a VC-based NoC whose data parallelism is equal to the sum of the one of the MPs system. Simulation-based analysis indicate that our MP approach actually improves the performance of a NoC in terms of throughput and latency. The VC-based approach does actually grant better improvements at the cost of a much higher area and power consumption. Hence MP-based NoC are an interesting cost effective solution of NoC systems.

In conclusion this Thesis has produced a total of four published papers all in major international conferences and a book chapter of a well known scientific editor. At the time of typing other two conference-papers and two journal articles are in phase of review for eventual publication.

References

[AA] AMBA-AXI.

[AAZ03] A. Greiner L. Mortiez A. Adriahantenaina, H. Charlery and C.A. Zeferino. SPIN: A scalable, packet switched, on-chip micro-network. In *Design, Automation and Test in Europe (DATE'05)*, page 20070, Washington, DC, USA, 2003. IEEE Computer Society.

[amb] Website. Amba Bus, Arm.

[AP95] K. V. Anjan and T. M. Pinkston. DISHA: a deadlock recovery scheme for fully adaptive routing. In *IPPS '95: Proceedings of the 9th International Symposium on Parallel Processing*, pages 537–543, Washington, DC, USA, 1995. IEEE Computer Society.

[BC06] L. Bononi and N. Concer. Simulation and analysis of network on chip architectures: ring, spidergon and 2d mesh. In *Design, Automation and Test in Europe (DATE'06)*, pages 154–159, 2006.

[BCG+07] L. Bononi, N. Concer, M. Grammatikakis, M. Coppola, and R. Locatelli. Noc topologies exploration based on mapping and simulation models. In *DSD '07: Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 543–546, 2007.

[BCH95] J. C. Bermond, F. Comellas, and D. F. Hsu. Distributed loop computer networks: A survey. *J. Parallel Distrib. Comput.*, 24(1):2–10, 1995.

[BD06a] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *ACM/IEEE (SC—05) Conf. Supercomputing*, pages 187–198, 2006.

[BD06b] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. pages 187–198, 2006.

[BJM$^+$05] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. on Parallel and Distributed Systems*, 16(2):113–129, February 2005.

[BM02] L. Benini and G. De Micheli. Networks on chip: A new SoC paradigm. *IEEE Computer*, 49(2/3):70–71, January 2002.

[CGL$^+$08] M. Coppola, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and L. Pieralisi. *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*. CRC Press, Inc., Boca Raton, FL, USA, 2008.

[cis] Virtual component interface standard.

[CLM$^+$04] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra. Spidergon: A NoC modeling paradigm. In *Proc. 2004 International Symposium on System-on-Chip*, page 15, November 2004.

[CMSSV99] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli. A methodology for “correct-by-construction” latency insensitive design. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 309–315, San Jose, CA, November 1999. IEEE.

[Coc02] P. Cocchini. Concurrent flip-flop and repeater insertion for high-performance integrated circuits. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 268–273, 2002.

[CPC08] N. Concer, M. Petracca, and L.P. Carloni. Distributed flit-buffer flow control for networks-on-chip. In *The Proceedings of the Sixth International Conference on Hardware/Software Codesign & System Synthesis (CODES+ISSS)*, page 6, September 2008.

[CSB09] N. Concer, S.Iamundo, and L. Bononi. aEqualized a novel routing algorithm for Spidergon NoC. In *Design, Automation and Test in Europe (DATE'06)*, 2009.

[CSV02] L. P. Carloni and A. L. Sangiovanni-Vincentelli. Coping with latency in SoC design. *IEEE Micro*, 22(5):24–35, September/October 2002.

[CXSP04] V. Chandra, A. Xu, H. Schmit, and L. Pileggi. An interconnect channel design methodology for high performance integrated circuits. In *Conf. on Design, Automation and Test in Europe*, pages 21138–21143, 2004.

[DA93] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Trans. on Parallel and Distributed Systems*, 4(4):466–475, 1993.

[Dal90] W. J. Dally. Virtual-channel flow control. In *in Proc. Int. Symp. Comp. Arch.*, pages 60–68, Seattle, Washington, May 1990.

[DBL05] Y. Durand, C. Bernard, and D. Lattard. FAUST : On-chip distributed architecture for a 4G baseband modem SoC, in. In *Proceedings of Design and Reuse IP-SOC*, pages 51–55, 2005.

[D.K97] H. D.Kubiatowicz. *Integrathed Shared Memory and Message Passing Communications in the Alewire Multiprocessor*. PhD thesis, Massachusetts Institute of Technology. Boston, 1997.

[DRKR03] J. Dielissen, A. Rdulescu, Goossens K, and E. Rijpkema. Concepts and implementation of the philips network-on-chip. In *In IP-Based SoC Design*, 2003.

[DRW98] R.P. Dick, D.L. Rhodes, and W. Wayne. TGFF: task graphs for free. In *CODES/CASHE '98: Proceedings of the 6th international workshop on Hardware/software codesign*, pages 97–101, Washington, DC, USA, 1998. IEEE Computer Society.

[DT01] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the Design Automation Conference*, pages 684–689, June 2001.

[DT04] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, San Mateo, CA, 2004.

[DYN03] Jose' Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection Networks. An Engineering Approach*. Morgan Kaufmann Publishers, San Mateo, CA, 2003.

[E.W] E.Weisstein. Moore Graphs. `http://mathworld.wolfram.com/MooreGraph.html`.

[FM82] C.M. Fiduccia and R.M. Mattheyses. A linear time heuristic for improving network partitions. In *in Proc. Int. Conf. Design Automation*, pages 175–181, Washington, DC, USA, 1982. IEEE Computer Society.

[For02] M. Forsell. A scalable high-performance computing solution for networks on chips. *IEEE Micro*, 22(5):46–55, 2002.

[GDvM$^+$03] K. Goossens, J. Dielissen, J. van Meerbergen, P. Poplavko, A. Rădulescu, E. Rijpkema, E. Waterlander, and P. Wielage. Guaranteeing the quality of services in networks on chip. pages 61–82, 2003.

[GIP$^+$07] C Grecu, A Ivanov, P Pande, A Jantsch, E Salminen, U Ogras, and R Marculescu. An initiative towards open network-on-chip benchmarks. In *Proceedings of the The First International Symposium on Networks-on-Chips (NOCS)*, 2007.

[GJS76] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[GPIS04] C. Grecu, P.P. Pande, A. Ivanov, and R. Saleh. Structured interconnect architecture: a solution for the non-scalability of bus based SoCs. In *14th ACM Great Lakes symposium on VLSI, (GLSVLSI '04)*, pages 192–195, April 2004.

[GRK+05] O. P. Gangwal, A. Rădulescu, K.Goossens, S. González Pestana, and E. Rijpkema. Building predictable systems on chip: An analysis of guaranteed communication in the Æthereal network on chip. In Peter van der Stok, editor, *Dynamic and Robust Streaming In And Between Connected Consumer-Electronics Devices*, volume 3 of *Philips Research Book Series*, chapter 1, pages 1–36. Springer, 2005.

[GvMPW02] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *Conf. on Design, Automation and Test in Europe*, 2002.

[H+07] Y. Hoskote et al. A 5-GHz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5):51–61, Sept.-Oct. 2007.

[HG07] A. Hansson and K. Goossens. Trade-offs in the configuration of a network on chip for multiple use-cases. In *Proceedings of the The First International Symposium on Networks-on-Chips (NOCS)*, pages 233–242, May 2007.

[HGR07] A. Hansson, K. Goossens, and A. Rădulescu. Avoiding message-dependent deadlock in network-based systems on chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2007:Article ID 95859, 10 pages, 2007. Hindawi Publishing Corporation.

[HJK$^+$00] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist. Network on chip: An architecture for billion transistor era. In *18th IEEE NorChip Conference*, November 2000.

[HM03] J. Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *Design, Automation and Test in Europe (DATE'03)*, 2003.

[HMH01] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[HOM06] J. Hu, U.Y. Ogras, and R. Marculescu. System-level buffer allocation for application-specific networks-on-chip router design. *IEEE Trans. on Computers*, 25(12):2919–2933, December 2006.

[HP06] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2006.

[Hwa01] F. K. Hwang. A complementary survey on double-loop networks. *Theoretical Computer Science*, 263(1-2):211–229, 2001.

[IC] IBM-CoreConnect.

[JBMM04] A. Jalabert, L. Benini, S. Murali, and G. De Micheli. $\times$*pipesCompiler*: a tool for instantiating application-specific NoCs. In *Conf. on Design, Automation and Test in Europe*, February 2004.

[JMBM04] A. Jalabert, S. Murali, L. Benini, and G. De Micheli. xpipesCompiler: A tool for instantiating application specific Networks on Chip. In *in Proc. Design, Automation and Test in Europe Conf.*, Paris, France, 2004.

[JZH] D. Jayasimha, B. Zafar, and Y. Hoskote. On-chip interconnection networks: Why they are different and how to compare them. *blogs.intel.com*.

[KJM+02] A. Kumar, A. Jantsch, M. Millberg, J. Oberg, J.P Soininen, M. Forsell, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. In *in Proc. Symp. VLSI*, page 117, Washington, DC, USA, 2002. IEEE Computer Society.

[KK97] G. Karypis and V. Kumar. METIS: a software package for partitioning unstructured graphs, meshes, and computing fill-reducing orderings of sparse matrices (version 3.0.3). Technical report, University of Minnesota, Dept. Comp. Sci. and Army HPC Research Center, November 1997.

[KKS05a] M. Kim, D. Kim, and G. E. Sobelman. MPEG-4 performance analysis for CDMA network on chip. In *Proceedings, International Conference on Communications Circuits and Systems*, pages 493–496, Hong Kong China, 2005.

[KKS05b] M. Kim, D. Kim, and G. E. Sobelman. MPEG-4 performance analysis for cdma network on chip. In *in Proc. Int. Conf. Comm. Circ. and Syst.*, pages 493–496, Hong Kong China, 2005.

[KKS+07] A. Kumar, P. Kundu, A. Singh, L.-S. Peh, and N. Jha. A 4.6Tbits/s 3.6GHz single-cycle noc router with a novel switch allocator in 65nm CMOS. In *International Conference on Computer Design (ICCD)*, October 2007.

[Knu91] D. E. Knuth. *The Art of Computer Systems Performance Analysis.* Wiley Computer Publishing, 1991.

[KO] Karim and Faraydon O. Octagonal interconnection network for linking processing nodes on an SOC device and method of operating same. US Patent 7218616.

[KPKJ07] A. Kumar, L.S. Peh, P. Kundu, and N. K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *International Symposium on Computer Architecture*, pages 150–161, 2007.

[KPKJ08] A. Kumar, L.S. Peh, P. Kundu, and N. K. Jha. Toward ideal on-chip communication using express virtual channels. *IEEE Micro*, 28(1):80–90, 2008.

[KPP06] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26(3):10–23, May/June 2006.

[LBBP94] Y.W. Lu, K. K. Bagchi, J. B. Burr, and A. M. Peterson. A comparison of different wormhole routing schemes. In *MASCOTS '94: Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, pages 323–328, Washington, DC, USA, 1994. IEEE Computer Society.

[Lei06] T. F. Leighton. *Introduction to Parallel Algorithms and Architectures: Algorithms and VLSI.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[LK03] R. Lu and C.K. Koh. Performance optimization of latency insensitive systems through buffer queue sizing of communication channels. In *Proc. Intl. Conf. on Computer-Aided Design*, page 227, 2003.

[LPP04] X. Liu, Y. Peng, and M. C. Papaefthymiou. Practical repeater insertion for low power: what repeater library do we need? In *Proceedings of the Design Automation Conference*, pages 30–35, 2004.

[LRD01] K. Lahiri, A. Raghunathan, and S. Dey. Evaluation of the traffic performance characteristics of system-on-chip communication architectures. In *in Proc. Int. Conf. VLSI Design*, pages 29–35, 2001.

[Luk75] J.A. Lukes. Combinatorial solution to the partitioning of general graphs. *IBM Journal of Research and Development*, 19:170–180, 1975.

[LZKC02] R. Lu, G. Zhong, C.K. Koh, and J.Y. Chao. Flip-flop and repeater insertion for early interconnect planning. In *Conf. on Design, Automation and Test in Europe*, March 2002.

[LZL00] X. Liu, S. Zhang, and T J. Li. A cost-effective load balanced adaptive routing scheme for mesh-connected networks. In *MASCOTS '00: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 532, Washington, DC, USA, 2000. IEEE Computer Society.

[MB06] G. De Micheli and L. Benini. *Networks on Chips: Technology and Tools (Systems on Silicon).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[McK] B. McKay. Nauty users guide (version 1.5). Technical report, Australian National University, Dept. Comp. Sci.

[M.G96] M.Galles. Scalable pipelined interconnect for distributed endpoint routing. 1996.

[MKWA08] H. Matsutani, M. Koibuchi, D. Wang, and H. Amano. Adding slow-silent virtual channels for low-power on-chip networks. In *Proceedings of the The First International Symposium on Networks-on-Chips*, pages 23–32, April 2008.

[MM04a] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores onto noc architectures. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 20896, Washington, DC, USA, 2004. IEEE Computer Society.

[MM04b] S. Murali and G. De Micheli. SUNMAP: a tool for automatic topology selection and generation for NoCs. In *in Proc. Int. Conf. Design Automation*, pages 914–919, New York, NY, USA, 2004. ACM.

[MM05] S. Murali and G. De Micheli. An application-specific design methodology for STbus crossbar generation. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 1176–1181, Washington, DC, USA, 2005. IEEE Computer Society.

[MNL09] M.Petracca, N.Concer, and L.P.Carloni. Multi-planes vs. virtual channel: a comparative analysis. In *submitted for conference publication*, 2009.

[MRG$^+$04] M.Coppola, R.Locatelli, G.Maruccia, L.Pieralisi, and A.Scandurra. Networks on chip: A new paradigm for systems on chip design. In *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, page 15, Washington, DC, USA, 2004. IEEE Computer Society.

[MT02] J. Miller et al. M. Taylor, J. Kim. The raw microprocessor: A computational fabric for software circuits and general purpose programs, 2002.

[NMLa] N.Concer, M.Grammatikakis, and L.Bononi. High level tools for NoC study and simulation. *submitted for journal publication.*

[NMLb] N.Concer, M.Petracca, and L.P.Carloni. Wire pipelining on synchronous noc. *submitted for journal publication.*

[NNJC06] S. Noh, V.-D. Ngo, H. Jao, and H.-W. Choi. Multiplane virtual channel router for network-on-chip design. pages 348–351, Oct 2006.

[Nor] S.C. North. NEATO user's guide. Technical Report 59113-921014-14TM, AT&T Bell Laboratories, Murray Hill, NJ, USA, October.

[NTIJ04] J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch. *Interconnect-Centric Design for Advanced SOC and NOC.* Springer, New York, NY, USA, 2004.

[O$^+$07] J.D. Owens et al. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108, Sept.-Oct. 2007.

[OI] OCP-IP.

[OM06a] U. Y. Ogras and R. Marculescu. It's a small world after all': NoC performance optimization via long-range link insertion. *IEEE Trans. on Very Large Scale Integration Systems*, 14(7):693–706, July 2006.

[OM06b] U. Y. Ogras and Radu Marculescu. It's a small world after all: Noc performance optimization via long-range link insertion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(7):693–706, July 2006.

[OMN] OMNeT++ discrete event simulation system. available online at http://www.omnetpp.org/.

[Ope] Website. `www.opencores.org/`.

[P$^+$07] A. Pullini et al. Bringing NoCs to 65nm. *IEEE Micro*, 27(5):75–85, 2007.

[PABB05] A. Pullini, F. Angiolini, D. Bertozzi, and L. Benini. Fault tolerance overhead in network-on-chip flow control schemes. In *SBCCI '05: Proceedings of the 18th annual symposium on Integrated circuits and system design*, pages 224–229, 2005.

[PD01] L.S. Peh and W. J. Dally. A delay model for router microarchitectures. *IEEE Micro*, 21:26–34, 2001.

[Peh01a] L.-S. Peh. *Flow Control and Micro-Architectural Mechanisms for Extending the Performance of Interconnection Networks.* PhD thesis, Stanford University, 2001.

[Peh01b] Li-Shiuan Peh. *Flow Control and Micro-Architectural Mechanisms for Extending the Performance of Interconnection Networks.* PhD thesis, Stanford University, August 2001.

[PG$^+$05] P.P.Pande, C. Grecu, , M.Jones, A.Ivanov, and R.Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Trans. on Computers*, Dec 2005.

[Pin08] A. Pinto. A platform-based approach to communication synthesis for embedded systems. May 2008.

[PR96] F. Pellegrini and J. Roman. SCOTCH: A software package for static mapping by dual recursive bi-partitioning of process and architecture graphs. In *in Proc. Int. Conf. on High Perf. Computing and Networking*, pages 493–498, London, UK, 1996. Springer-Verlag.

[PS] Device Transaction Level (DTL) Protocol Specification PHILIPS Semiconductors.

[PS01] Timothy M. Pinkston and Jeonghee Shin. Trends toward on-chip networked microsystems. *Intl. J. High Performance Computing and Networking*, 3(1):3–18, 2001.

[Sch02] L. Scheffer. Methodologies and tools for pipelined on-chip interconnect. In *Proc. Intl. Conf. on Computer Design*, pages 152–157, October 2002.

[SCK07] K. Srinivasan, K. S. Chatha, and G. Konjevod. Application specific network-on-chip design with guaranteed quality approximation algorithms. In *Proceedings of the Design Automation Conference*, pages 184–190, 2007.

[SP03a] Y. H. Song and T. M. Pinkston. A progressive approach to handling message-dependent deadlock in parallel computer systems. *IEEE Trans. on Parallel and Distributed Systems*, 14(3):259–275, 2003.

[SP03b] Y. H. Song and T. M. Pinkston. A progressive approach to handling message-dependent deadlock in parallel computer systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):259–275, 2003.

[STAN04] D. Siguenza-Tortosa, T. Ahonen, and J. Nurmi. Issues in the development of a practical noc: the proteo concept. *Integr. VLSI J.*, 38(1):95–105, 2004.

[SVE07] Y. A. Sekercioglu, A. Varga, and G. K. Egan. Parallel simulation made easy with omnet++. In *Proceedings of the 15th European Simulation Symposium (ESS'03)*, pages 493–499, October 2007.

[T$^+$02] Michael Taylor et al. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, March/April 2002.

[TJ05] T.Bjerregaard and J.Sparso. A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip. In *In Proceedings of the 11th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 2005.

[TKM$^+$02] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The Raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 22(2), Mar-Apr 2002.

[TS04] T.Felicijan and S.Fourber. An asynchronous on-chip network router with quality-of-service (qos) support. In *In Proceedings IEEE International SOC Conference*, 2004.

[VCG] VCG user manual. `http://rw4.cs.uni-sb.de/users/sander/html/gsvcg1.html`.

[VSD97] A.S. Vaidya, A. Sivasubramaniam, and C. R. Das. Performance benefits of virtual channels and adaptive routing: an application-driven study. In *ICS '97: Proceedings of the 11th international conference on Supercomputing*, pages 140–147, New York, NY, USA, 1997. ACM.

[WZPM02] H.S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In *IEEE/ACM Intl. Symp. on Microarchitecture (MICRO-35)*, November 2002.