Lecture Notes in Electrical Engineering 184

Umit Y. Ogras · Radu Marculescu

Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures

Springer

Lecture Notes in Electrical Engineering

Volume 184

For further volumes:
http://www.springer.com/series/7818

Umit Y. Ogras · Radu Marculescu

Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures

Springer

Umit Y. Ogras
Intel Corporation
Hillsboro, OR
USA

Radu Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA
USA

ISSN 1876-1100 ISSN 1876-1119 (electronic)
ISBN 978-94-007-3957-4 ISBN 978-94-007-3958-1 (eBook)
DOI 10.1007/978-94-007-3958-1
Springer Dordrecht Heidelberg New York London

Library of Congress Control Number: 2012956161

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This book is based on an exciting and intense period of collaboration while we were trying to understand the fundamental aspects of communication-based design of multiprocessor systems-on-chip (MPSoC). While the core material is largely based on first author's EDAA award winning Ph.D. thesis, the overall structure and contents has been updated to reflect our deeper and broader understanding of the role of the "network" in MPSoC design that only the intervening years and prototyping experience made it possible.

In terms of contents, the book provides a system-level view on various modeling and optimization issues for future MPSoCs. As such, it can be used as an advanced introduction in the area of multicore design and optimization where communication happens via the network-on-chip approach. At the same time, we hope to see this book helping readers overcome the abundance of available information (not always well structured) researchers face nowadays and stimulate new research in this exciting area.

We would like to express our gratitude to many close collaborators and entities that make this endeavor possible. First, we would like to thank our many colleagues at Carnegie Mellon University who directly influenced some of our ideas and provided valuable feedback over the years. In particular, we thank Drs. Jingcao Hu, Paul Bogdan, Hyung Gyu Lee, Chen-Ling Chou, Nicholas H. Zamora, Jung-Chun Kao, Radu David of the System Level Design group and Siddharth Garg, Natasa Miskov-Zivanov, Puru Choudhary and Prof. Diana Marculescu of the Energy Aware Computing group, also Professors Rob A. Rutenbar, Hui Zhang, and Shawn Blanton. In general, the CSSI environment at CMU proved to be the true 'home' for many of our intellectual endeavors.

Outside CMU, we had great collaborators that make this adventure exciting. In particular, we would like to acknowledge many discussions we had with Professors Alberto Sangiovanni-Vincentelli and Jan Rabaey of UC Berkeley during the GSRC years, as well as Li-Shiuan Peh of MIT, Natalie Enright-Jerger of University of Toronto, Petru Eles of Linkoping University, Naehyuck Chang of Seoul National University, Ran Ginosar of Technion, and Dr. Peter Feldmann of IBM who contributed to various stages of our research.

We have also received valuable feedback from our industrial mentors who monitored this research and constantly interacted with us over the years. In particular, Drs. Mike Kishinevsky, Yatin Hoskoke, and Stefan Rusu of Intel Corp., as well as Dr. Jim Holt from Freescale helped us shape some of our ideas and offered valuable insight on many practical aspects of this work.

We acknowledge the generous support from the Semiconductor Research Corporation, National Science Foundation, and Gigascale Systems Research Focus Center, one of five research centers funded under the Focus Center Research Program, that made this research possible.

Finally, we would like to thank our dear families (Rukiye, Osman, and Zeyneb Ogras, Diana, Andrei, Victor, and Cristina Marculescu, as well as our parents and families overseas) for unconditional love and support over the years. None of this would have been possible and meaningful without them.

Contents

Abbreviations

ALU	Arithmetic logic unit
APCG	Application characterization graph
ASIC	Application-specific integrated circuit
CPU	Central processing unit
CTG	Communication task graph
DCT	Discrete cosine transform
DLL	Delay locked loop
DSM	Deep-submicron
DSP	Digital signal processor
FIFO	First-in first-out
FLOPS	Floating point operations per second
FPGA	Field-programmable gate array
FPU	Floating point unit
GALS	Globally synchronous locally asynchronous
HDL	Hardware description language
IDCT	Inverse discrete cosine transform
IP	Intellectual property
ITRS	International roadmap for semiconductors
JPEG	Joint photographic experts group
MPEG	Moving Picture Experts Group
NI	Network interface
NoC	Network-on-chip
P2P	Point-to-point
PE	Processing elements
QoS	Quality of service
RM	Routing matrix
RTL	Register transfer level
SoC	System-on-chip
VC	Virtual channel

VFI	Voltage–frequency island
VLE	Variable length encoding
VLSI	Very large-scale integration

Abstract

Design space exploration for systems-on-chip (SoCs) has focused traditionally on the computational aspects of the problem at hand. However, as the number of components on a single chip and their performance continue to increase, the communication architecture plays a major role in reducing the area and power consumption, as well as improving the overall performance of such multiprocessor systems. As a result, a shift from computation-based design to communication-based design becomes mandatory. Toward this end, the network-on-chip (NoC) communication architecture emerged recently as a promising alternative to the classical bus and point-to-point communication architectures.

In this book, we address a few fundamental research problems related to modeling, analysis, and optimization of multiprocessor systems-on-chip (MPSoCs) where communication happens via the NoC approach. More precisely, we present new system-level models, algorithms, software tools, as well as hardware prototypes that can be used to help the design and optimization of application-specific NoCs. The discussion gravitates around the following key contributions to the state-of-the-art:

- We discuss a new mathematical model for on-chip routers and then use this model for NoC performance analysis. Our performance analysis approach can be used not only to obtain fast and accurate performance estimates, but also guide the NoC design process within an optimization loop.
- We present a methodology to automatically synthesize an NoC architecture which is a superposition of a few long-range links and a standard mesh network. The few application-specific long-range links we insert induce small world effects and significantly increase the critical traffic workload at which the network transitions from a free to a congested state. This way, we can exploit the benefits offered by complete regularity and partial topology customization.
- We consider flow control algorithms specifically developed for NoCs and propose a predictive closed-loop flow control mechanism which can control the packet injection rate at traffic sources in order to regulate the total number of packets in the network. This predictive flow control algorithm enjoys the

simplicity of the switch-to-switch algorithms, while directly controlling the traffic sources, very much like the end-to-end algorithms. Consequently, this approach can lead to significant reductions in the average and maximum packet latency.

- The increase of energy consumption and synchronization among processing cores that operate at multiple voltage and frequency levels are major issues in the design of NoCs for multicore SoCs. To this end, we present a design methodology for partitioning an NoC architecture into multiple voltage-frequency islands (VFIs) and assigning supply and threshold voltage levels to each VFI. In order to compensate for run-time workload and parameter variations, we further discuss an online feedback control mechanism that can dynamically adjust the operating voltage and frequency around the static values found by the proposed partitioning algorithm.
- Finally, we present hardware prototypes to support the theoretical findings discussed in this book. Besides demonstrating the feasibility of our newly proposed algorithms, prototyping enables us to evaluate the area, power consumption, and performance figures of our designs accurately.

In summary, the relentless scaling down of CMOS technology will soon enable multicore platforms consisting of thousands of communicating IP blocks integrated on a single chip. Successful integration of these blocks on the same silicon substrate relies on designing truly scalable communication architectures. The most promising solution to date is given by the structured communication approach via the NoC-based architecture. In this book, we present new system-level models, algorithms, and tools meant to support a communication-centric design methodology for future multicore systems.

Keywords Multiprocessor systems-on-chip (MPSoCs) · On-chip communication · Network-on-chip · Performance analysis · Flow control · Energy and power consumption · Voltage-frequency islands · Dynamic power management · Router design · Traffic modeling · Architecture customization · Long-range links

Chapter 1
Introduction

To alleviate the complex communication problems that arise as the number of on-chip components increases, network-on-chip (NoC) architectures have been recently proposed to replace global interconnects. This chapter first provides a general description of NoC architectures. Then, it describes a generic synthesis flow for NoCs starting from the application specification through tape-out and applications. Finally, it addresses the interactions among these research problems and put the NoC design process into perspective.

1.1 Network-on-Chip Architectures

Systems-on-Chip (SoCs) designed at nanoscale domain will soon contain billions of transistors [18]. This makes it possible to integrate hundreds of IP cores running multiple concurrent processes on a single chip. The design of such complex SoCs faces a number of design challenges. First, the richness of computational resources places tremendous demands on the communication resources. Consequently, the entire design methodology needs to change from computation-based design to communication-based design. Second, global interconnects can cause severe on-chip synchronization errors, unpredictable delays and high power consumption. As a consequence, exploiting dynamic voltage and frequency scaling in the context of globally asynchronous locally synchronous design styles are needed in order to mitigate such effects. Finally, increasing complexity, costs and tight time-to-market constraints require new design methodologies that favor design re-use at all levels of abstraction [14]. As a result, novel on-chip communication solutions that can effectively address all these design issues are in high demand.

Traditionally, two types of on-chip communication schemes, namely Point-to-Point (P2P) and bus-based communication, have been considered for implementing integrated systems. The P2P communication architecture can provide the utmost communication performance at the expense of implementing dedicated

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1_1,
© Springer Science+Business Media New York 2013

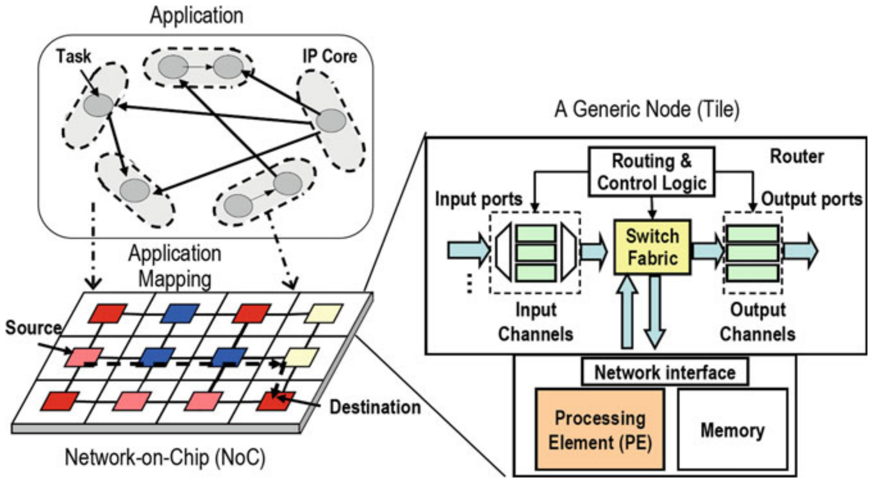


Fig. 1.1 Generic NoC architecture, application and its mapping to the NoC are illustrated. The anatomy of a node which consist of an on-chip router and processing element is also depicted on the *right-hand* side of the figure

channels among all the communicating IP cores. However, the P2P architecture lacks scalability due to the high complexity, cost and design effort required when the system needs to scale up. On the other hand, the bus-based architecture can connect a few tens of IP cores in a cost efficient manner by reducing the design complexity and eliminating the dedicated wires and specialized interfaces required by the P2P communication architecture. However, the bus-based architecture still lacks scalability, both in terms of power consumption and performance, when it comes to systems involving more than a few communicating cores. Indeed, from an implementation standpoint, a bus-based design involving say tens of cores would clearly provide very low performance figures due to its limited bandwidth capabilities. Moreover, the large capacitive load of the bus drivers would result in large delays and energy consumption in the interconnected wires [20]; this makes the bus-based solution inappropriate for implementing a complex designs.

In contrast to these methods, the Network-on-Chip (NoC) approach represents a promising solution to the on-chip communication problems [4, 8, 12, 13]. As shown in Fig. 1.1, modern SoC architectures consist of heterogeneous IP cores such as processing cores, graphics engines, video processors, embedded memory blocks, I/O devices, etc. Each such processing element (PE) is attached to a local router which connects the core to the neighboring nodes via a NoC type of interconnect. More precisely, when a source node sends a packet to a destination node (see Fig. 1.1), the packet is first generated and transmitted from the local processor to the router attached to it via a network interface (NI). The NI is meant to enable seamless communication between the various cores and the network. Then, the packet is stored at the input channels and the router starts servicing it. This service time includes the time needed to take a routing decision, allocate a channel and traverse the switch fabric. After being serviced, the packet moves to the next router on its path and the process repeats until the packet arrives at its final destination. As a result, the communication among various cores is achieved by generating, processing, and forwarding packets through the network infrastructure rather than by routing global wires. Not surprisingly, the network communication latency depends on the characteristics of the target application (e.g., inter-task communication volume, deadlines), computational elements (e.g., processor speed, memory), and network characteristics (e.g., network bandwidth, buffer size).

1.2 Advantages of NoC Architectures

By eliminating the (ad-hoc) global wires, the NoC approach provides the following advantages:

- **Scalability**: Since the communication between different nodes is achieved by routing packets, a large number of cores can be connected without the need of using long global wires. Instead, the processing nodes can be connected to the communication infrastructure via short links and standard interfaces. Moreover, the network bandwidth scales with the number of cores in the design. Hence, the NoC paradigm provides a highly scalable communication architecture.
- **Design reuse**: Reuse is recognized as one of the most effective techniques to improve the design productivity [18]. The modularity of the NoC approach offers a great potential for re-using the network routers and other existing IP cores such as processor cores and multimedia codecs. The routers, the interconnect and the lower-level communication protocols can be designed, optimized and verified only once and reused subsequently in a large number of products. Likewise, many existing IP cores which have been designed with certain protocols in mind (e.g., CoreConnect, AMBA) can be reused in NoC designs using wrappers which can efficiently interface the existing bus-based IPs and the NoC communication infrastructure.
- **Predictability**: The structured nature of wires in NoCs facilitates well-controlled and optimized electrical parameters. In turn, these controlled parameters enable the use of aggressive signaling circuits which can reduce the power dissipation and propagation delay significantly [8]. In addition to this, coping with physical design problems such as cross-talk, and immunity to noise becomes easier due to this regularity.

1.3 A Generic NoC Synthesis Flow

Before surveying the NoC research, we describe a generic synthesis flow for NoCs starting from the application specification through tape-out. As shown in Fig. 1.2, the flow has three major steps: (1) Application modeling and optimization, (2) NoC

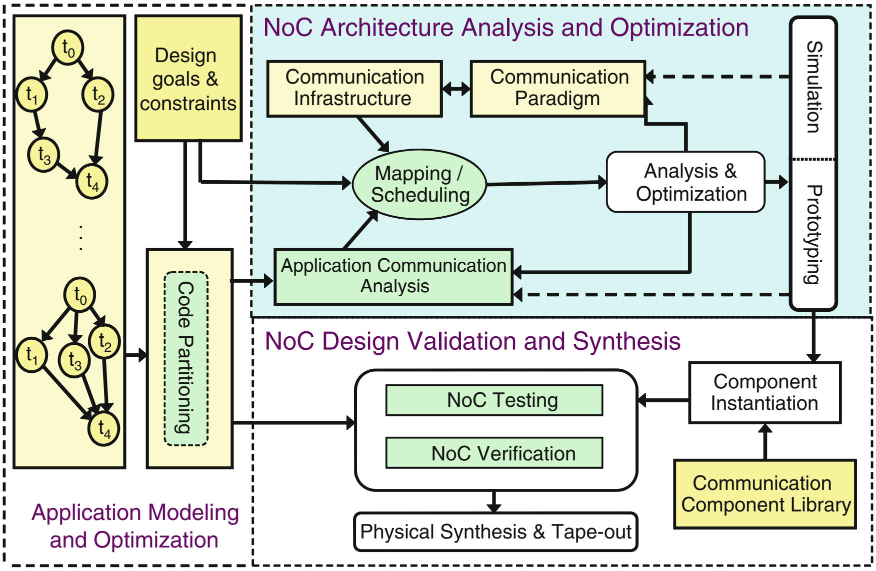

t_0
t_1
t_2
t_3
t_4
Design goals & constraints
Code Partitioning
Application Modeling and Optimization
NoC Architecture Analysis and Optimization
Communication Infrastructure
Communication Paradigm
Mapping / Scheduling
Analysis & Optimization
Application Communication Analysis
Simulation
Prototyping
NoC Design Validation and Synthesis
NoC Testing
NoC Verification
Component Instantiation
Communication Component Library
Physical Synthesis & Tape-out

Fig. 1.2 A generic NoC synthesis flow. It has three major steps: (1) Application modeling and optimization, (2) NoC architecture analysis and optimization, (3) NoC design validation and synthesis

architecture analysis and optimization, (3) NoC design validation and synthesis. These steps are detailed next.

- **Application Modeling and Optimization**. One of the first questions to be answered is what are the target applications and their associated traffic patterns, as well as the interconnect bandwidth requirements for each node in the network (see left-most box in Fig. 1.2). In coherent shared memory architectures, the communication patterns depend primarily on the flow of external memory traffic and the on-die cache hierarchy and coherence protocol. On the other hand, non-coherent shared memory or message passing models depend more directly on the explicit communication patterns of various applications. Clearly, the application partitioning and overall system architecture (e.g., homogenous vs. heterogeneous cores, synchronous vs. asynchronous clocking, memory controllers, I/O devices, etc.) significantly impact the network traffic. Having a good model for the target application helps finding the best application-architecture combinations that satisfy various performance and energy constraints. Application models must be scalable and flexible enough for quick analysis. Furthermore, it is also crucial to capture the key behavior of the application in order to have confidence in the predicted results. If a software reference is used for the target application, or the application has software components, then a code partitioning step may be needed [2, 17]. This step is crucial to extract as much parallelism from the application as possible. Only after this step is completed, the concurrency provided by the NoC architectures can be fully exploited.

Finally, when the NoC platform is general purpose or it is likely to accommodate a large set of applications, random traffic models such as uniform traffic [11] may be used to suplement the standard benchmark.

- **NoC Communication Architecture Analysis and Optimization**. A good understanding of the traffic patterns and system requirements helps determining the optimal network topology. This has a huge impact on design costs, power, performance and helps designers choose an efficient routing algorithm and flow control scheme in order to manage the incoming traffic. In addition to performance, accurate power models are of crucial importance. Indeed, early and efficient floorplanning and accurate estimation of the area and power are necessary for optimizing the design and ensuring a quick backend convergence. These issues are shown in the upper right box in Fig. 1.2. At this point, communication bandwidth and network latency are the key performance metrics, while area, power, and reliability are the key cost metrics. Several formalisms (e.g., queueing networks, Petri nets, process algebra, etc.) can be used to carry out this analysis step. If the design constraints are not met, then the mapping process and/or architectural parameters are reiterated until the desired result is obtained. From this perspective, the performance analysis needs to be tractable in order to allow for cutting down the time spent in the design cycle. This is important to enable fast design space exploration and finding good quality solutions within a short (and predictable) time budget. Once the results obtained at this high level of abstraction are satisfactory and the design space is effectively pruned, more accurate evaluations of the candidate architectures are needed. Referring to the methodology in Fig. 1.2, we note that simulation and often prototyping are used for performance evaluation purposes. While this phase, especially prototyping, may take a significantly longer time, it is necessary to determine the final architecture and mapping that will be used during the synthesis stage.
- **NoC Design Validation and Synthesis**. After a particular architecture-mapping combination is selected, the final phase seeks to implement the NoC communication architecture by instantiating the components from a communication library and carrying out the synthesis process. This includes simulation and verification of the final design to ensure that user-defined design constraints and overall design goals are properly met. Finally, the synthesis, floorplanning and layout generation steps are performed before tape-out, as summarized in Fig. 1.2. Prototyping, verification and testing are also used to address early on various resilience and fault tolerance issues.

In summary, the NoC design is a multi-faceted process involving both application and communication architecture. This book is primarily focused on the second stage highlighted in Fig. 1.2, i.e. NoC communication architecture analysis and optimization. We also address NoC architecture and application modelling issues in Chap. 4. Other important issues such as NoC synthesis flow [5, 19], testing [1, 3], verification [7, 10], and low-level implementation [15] are addressed in the literature, as we detail next.

1.4 NoC Design Space and State of the Art

From a design methodology standpoint, designing on-chip networks differs significantly from designing large scale interconnection networks [6, 9]; this is mainly due to the different cost functions and constraints that need to be imposed in order to ensure the on-chip functionality. For example, the energy consumption is one of the major constraints in NoC design. Moreover, NoCs are far more resource limited compared to traditional networks. For this reason, NoCs should be implemented with minimum area overhead. Likewise, unlike large scale networks, NoCs can be applications-specific. Therefore, an ample portion of the design effort is likely to go into optimizing the network for a specific application or class of applications.

NoC design problems can be conceived as representing a 3-D design space [16], as shown in Fig. 1.3. In this representation, the level of randomness increases towards the tip of the *x*-, *y*-, and *z*- arrows. The *communication infrastructure* is depicted along the *x-axis* in Fig. 1.3. This dimension defines how nodes are interconnected to each other and reflects the properties of the underlying network. As shown in Fig. 1.3, examples include regular networks, small-world networks, networks with customized buffer sizes, or fully customized networks. The *communication paradigm* is represented by the *y-axis* in Fig. 1.3. This dimension captures the dynamics of transferring packets (e.g., deterministic vs. adaptive routing, QoS-based routing, etc.) through the network. Finally, the third

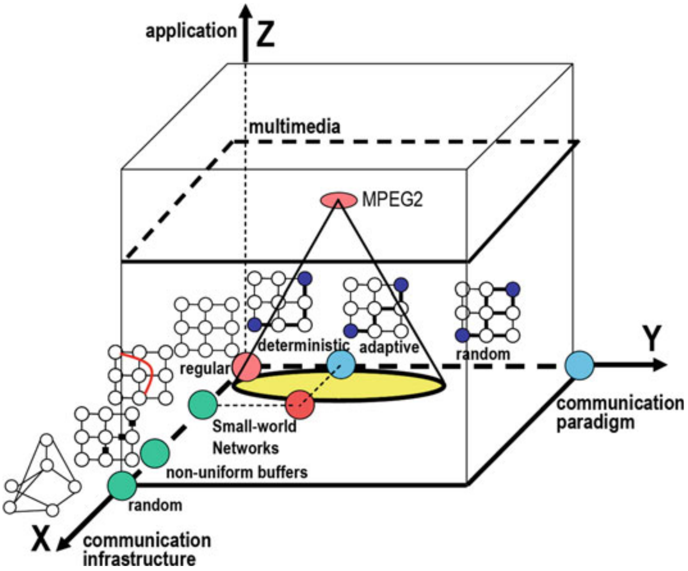


Fig. 1.3 The NoC design space in a 3D representation. As an example, optimizing the communication infrastructure (Sect. 2.3) in conjunction with the communication paradigm (Sect. 2.2) covers a region on the XY plane

dimension, *application mapping*, defines how different tasks of the target application get mapped to the network nodes (e.g., how various traffic sources and sinks are distributed across the network) in order to satisfy various design constraints. Although not depicted in Fig. 1.3, analysis, simulation and validation can be viewed as encompassing the entire design space; these steps are an integral part of work in any dimension and for research spanning multiple dimensions.

It is important to note that each axis represents a continuum rather than a discrete set of solutions. Also, any point inside the 3D representation in Fig. 1.3 actually denotes a possible design trade-off among various problems and techniques already discussed. For example, designing a routing algorithm while solving the topology customization problem corresponds to finding a legitimate point in the *x*-*y* plane. More generally, the shaded circle within the *x*-*y* plane shows a *range* of design solutions which correspond to various design trade-offs between the communication infrastructure and the communication paradigm. Indeed, by projecting a particular application (e.g., MPEG-2 in Fig. 1.3) onto the *x*-*y* plane, one can see that any solution within the shaded circle can implement the application at hand while pursuing different trade-offs. By the same token, by projecting up a particular point from the *x*-*y* plane onto the application plane, we can see a family of related multimedia applications (e.g., MPEG-2 video, audio) benefiting from such a particular implementation.

Design space exploration along each dimension has been considered to some extent without explicitly referring to such a classification. In the next chapter, we provide a detailed literature review following the three dimensional design space view.

References

1. Amory AM, Briao E, Cota E, Lubaszewski M, Moraes FG (2005) A scalable test strategy for network-on-chip routers. In: Proceedings of IEEE international test conference, November 2005
2. Ball M, Cifuentes C, Deepankar B (2004) Partitioning of code for a massively parallel machine. In: Proceedings of the international conference on parallel architecture and compilation techniques, September 2004
3. Bengtsson T, Jutman A, Kumar S, Ubar S, Peng Z (2006) Off-line testing of delay faults in NoC interconnects. In: Proceedings of EUROMICRO conference on digital system design, August 2006
4. Benini L, De Micheli G (2002) Networks on chips: a new SoC paradigm. IEEE Comput 35(1):70–78
5. Bertozzi D et al (2005) NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. IEEE Trans Parallel Distrib Syst 16(2):113–129
6. Bertsekas D, Gallager R (1992) Data Networks. Prentice Hall, Upper Saddle River
7. Chatterjee S, Kishinevsky M, Ogras UY (June 2010) Quick formal modeling of communication fabrics to enable verification. In Proceedings IEEE international high-level design validation and test workshop, June 2010, pp 42–49
8. Dally WJ, Towles B (2001) Route packets, not wires: on-chip interconnection networks. In: Proceedings of design automation conference, June 2001

9. Duato J, Yalamanchili S, Ni L (2002) Interconnection networks: an engineering approach. Morgan Kaufmann, San Mateo
10. Goossens K et al (2005) A design flow for application-specific networks-on-chip with guaranteed performance to accelerate SoC design and verification. In: Proceedings of design, automation and test in Europe conference, March 2005
11. Grecu C, Ivanov A, Pande P, Jantsch A, Salminen E, Ogras UY, Marculescu R (2007) An initiative towards open network-on-chip benchmarks. NoC benchmarking white paper [Online]. http://www.ocpip.org/uploads/documents/NoC-Benchmarks-WhitePaper-15.pdf
12. Hemani A, Jantsch A, Kumar S, Postula A, Oberg J, Millberg M, Lindvist D (2000) Network on a chip: an architecture for billion transistor era. In: Proceedings of the IEEE NorChip conference, November 2000
13. Jantsch A, Tenhunen H (eds) (2003) Networks-on-chip. Kluwer, Norwell
14. Keutzer K, Malik S, Newton AR, Rabaey J, Sangiovanni-Vincentelli A (2000) System-level design: orthogonalization of concerns and platform-based design. IEEE Trans Comput-Aided Design Integr Circ Syst 19(12):1523–1543
15. Lee K et al (2004) A 51mW 1.6GHz on-chip network for low-power heterogeneous SoC platform. In: International solid-state circuits conference, February 2004
16. Marculescu R, Ogras UY, Peh L, Jerger NE, Hoskote Y (2009) Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. IEEE Trans Computer-Aided Design Integr Circ Syst 28(1):3–21
17. Ryoo S et al (2007) Automatic discovery of coarse-grained parallelism in media applications. Trans High-Performance Embed Archit Compilers 1(1):187–206
18. Semiconductor Association (2006) In: The international technology roadmap for semiconductors (ITRS)
19. Srinivasan K, Chatha KS, Konjevod G (2006) Linear programming based techniques for synthesis of network-on-chip architectures. IEEE Trans Very Large Scale Integr Syst 14(4):407–420
20. Wolkotte PT, Smit GJM, Kavaldjiev N, Becker JE, Becker J (2005) Energy model of networks-on-chip and bus. In: Proceedings of the international symposium on system-on-chip, November 2005

Chapter 2
Literature Survey

One of the most important reasons for using NoC architectures is their promise for scalability [15, 41, 60, 64, 148]. Several books provide an introduction to the NoC concept and discuss various research issues [50, 76, 87, 107, 125], while an exhaustive list of references can be found in some NoC bibliographies available on-line [121, 127]. Likewise, a comprehensive introduction to NoCs and existing design practices is presented in [21]. In what follows, we provide a systematic literature review which is structured along the lines discussed in [105].

2.1 Application Modeling and Optimization for NoC Communication

2.1.1 Traffic Models

Traffic models refer to the mathematical characterization of workloads generated by various classes of applications. With network performance being highly dependent on the actual traffic, it is obvious that accurate traffic models are needed for a thorough understanding of the huge design space of network topologies, protocols, and implementations. Since implementing real applications is time-consuming and lacks flexibility, such analytical models can be used instead to evaluate the network performance early in the design process.

Traffic characteristics have been long recognized as playing a major part in multicore systems design. For instance, in [171] the authors introduce an analytical traffic model based on identifying self-similar effects in multimedia traffic. These effects have important consequences for the design of on-chip multimedia systems since self-similar processes have properties which are completely different from traditional short-range dependent or Markovian processes that have been traditionally used in system-level analyses. Later, the authors in [160] derive a comprehensive traffic model for NoCs which exposes both spatial and temporal dimensions of traffic via three statistical parameters: hop count, burstiness, and

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1_2,
© Springer Science+Business Media New York 2013

packet injection distribution. Interestingly enough, it has been subsequently reported that even the traffic generated by programmable cores consists of multiple program phases which exhibit the same type of self-similar behavior [145].

It should be noted, however, that the research in this area is still behind due to the lack of a widely accepted set of NoC benchmarks. This situation has two primary reasons. First, the applications suitable for NoC platforms are typically very complex. For instance, it is common for applications to be partitioned among tens of processes (or more) in order to allow for evaluations of scheduling, mapping, etc. For general purpose chip multiprocessors (CMPs), benchmarks such as SPLASH, originally designed for shared-memory multiprocessors, may be used but it is, however, unclear how effectively such benchmarks can actually stress the NoCs. Second, compared to traditional research areas like physical design where the design constraints are static (e.g., the aspect ratio of the blocks, number of wires between different blocks, etc.), the NoC research requires detailed information about the dynamic behavior of the system; this is hard to obtain even using detailed simulation or prototyping. As a result, most researchers and designers still rely on synthetic traffic patterns such as uniform random, bit-permutation traffic, to stress-test a network design [40, 90]. A first step towards a unified approach for embedded platforms has been made recently via the OCP-IP benchmarking initiative [58]. Similarly, there have been initial steps towards releasing parallel benchmarks targeting the future CMPs [19]. Such initiatives can certainly boost the research progress in this area. However, there needs to be more research aimed at developing accurate traffic models, as well as in-depth studies that project the NoC traffic for emerging workloads.

2.1.2 Application Mapping

Applications are typically described as a set of concurrent tasks that have been already assigned and scheduled onto a set of selected IP cores. The mapping problem for NoCs is to decide how to topologically place the selected set of cores onto the PEs of the network, such that some metrics of interest are optimized. We note that PE simply means a placeholder connected to one of the network routers. In other words, “mapping” here means determining which IP core connects to which router in the network; this, obviously, greatly impacts both performance and energy consumption of the NoC.

The mapping problem for NoCs has been first addressed by the authors of [68], where a branch and bound algorithm is proposed to map a given set of IP cores onto a regular NoC architecture such that the total communication energy is minimized. At the same time, the performance of the resulting communication system is guaranteed to satisfy the specified design constraints through bandwidth reservation. Follow-up work considered the mapping problem with increased path diversity [113] as well as additional latency constraints [162]. Likewise, a multi-objective mapping algorithm that finds the Pareto mappings with optimum performance and power consumption is presented in [7]. Improving upon these studies, the authors in [62] propose a more general, unified approach for application mapping and routing path selection which considers both best effort and guaranteed service traffic.

One key component needed to solve the application mapping problem is the analytical model used for solution evaluation. For instance, if the goal is communication energy minimization, an accurate energy model is crucial. We note that many mapping algorithms use (directly or indirectly) the average packet hop count as a cost function, by relating the average number of packet hops to the communication energy consumption [71] or communication cost [162]. Along the same lines, effective performance models (such as those discussed in Chap. 5) are needed. When PEs have different sizes, the communication latency and power consumption per unit of data exchanged between any two neighboring routers may differ significantly. Therefore, embedding floorplanning information within the mapping loop becomes necessary to get more accurate energy/latency estimates [112].

With increasing level of programmability, MPSoCs are used under multiple use case scenarios. Hence, it becomes necessary to allocate the NoC resources based on different communication requirements (i.e., bandwidth and latency) and traffic patterns that characterize various use cases [116]. By the same token, with ever increasing power density and cooling costs, it is important to reduce or eliminate the potential hotspots and have a thermally-balanced design [73]. Finally, efficient techniques for *run-time* mapping and management of applications are needed. Towards this end, software development and code placement for embedded multiprocessors are discussed in [54]. Similarly, for applications launched dynamically, run-time mechanisms for mapping [3, 37] and/or migrating [17] are needed. Since execution time and arrival order of applications are not known a priori, finding optimal solutions is difficult and remains a big challenge.

2.1.3 Application Scheduling

Another important problem in NoC design is communication and task scheduling. Although scheduling is a traditional topic in computer science, most previous work focuses on maximizing performance [139, 176]. More recently, energy-aware scheduling techniques for hard real-time [59, 146, 156] and distributed [99, 109] systems have also been introduced, but they address only the bus-based or P2P communication. Without taking into consideration the network congestion which may change dynamically during tasks execution, such techniques cannot be directly applied to NoC scheduling. We note that mapping and scheduling problems can be considered jointly. However, finding the optimum solution remains an open problem due to its complexity.

Communication and task scheduling for NoCs is addressed in [69] where the authors present a scheduling algorithm which minimizes the overall energy consumption of the system while guaranteeing the real-time deadlines imposed on tasks. Likewise, scheduling and arbitration policy for NoCs that use code division multiple access protocol is presented in [81].

Capturing the dynamic system behavior, i.e., the change system behavior due to the incoming and completed applications, is also important. To this end, the work in [140] models the applications as a set of independent jobs and presents exact timing models that capture both computation and communication of a job. Similarly, the work in [165] extends the NoC scheduling to consider multiple use case scenarios, hence communication patterns. By allowing the bandwidth to be shared among multiple communication scenarios, a better resource utilization of the NoC is obtained.

It should be also noted that dynamic voltage frequency scaling (DVFS) can be used in conjunction with scheduling in order to minimize the overall energy consumption. Such techniques have been proposed in the past for both bus- [59, 146] and NoC-based communication [155]. In these approaches, voltage scaling is applied to tasks and/or communication to minimize the power consumption, while accounting for the DVS overhead and satisfying the application deadlines.

We note that although we discuss here the mapping and scheduling problems separately, they can also represent a joint optimization problem. However, since they are both very hard problems to solve, such an integrated approach remains an open problem. This is particularly challenging for NoCs since communication delay is difficult to estimate and so deriving accurate models that can be used to guarantee hard deadlines is a huge problem by itself.

2.2 Communication Paradigm

2.2.1 Packet Routing

Given an underlying topology, the routing protocol determines the actual route taken by a message. The routing protocol is important as it impacts all network metrics, namely, latency (as the hop count is directly affected by the actual route), throughput (as congestion depends on the ability of the routing protocol to load balance), power dissipation (as each hop incurs a router energy overhead), QoS (as routing can be used to channel different message flows along distinct paths to avoid interference) and finally reliability (as the routing protocol needs to choose routes that avoid faults).

Routing has been extensively studied in classical interconnection networks, many of which have been leveraged in on-chip networks. One example is the dimension-ordered routing which routes packets in one dimension, then moves on to the next dimension, until the final destination is reached. While such a technique is very popular due to its simplicity, adaptive routing techniques (e.g., turn model routing, planar adaptive routing [40, 47]) can provide better throughput and fault tolerance by allowing alternative paths depending on the network congestion and run-time faults. Oblivious routing algorithms which generate routes without any knowledge of traffic have also been extensively studied in the context of classical interconnection networks [147] and can be relevant to on-chip networks due to their low overhead [169].

New routing protocols have also been investigated specifically for NoCs. For instance, *deflective routing* in [119] routes packets to one of the free output channels belonging to a minimal path; if this is not possible, then packets are misrouted. Techniques have been also proposed to dynamically switch between deterministic and adaptive routing to exploit the trade-off between them [70].

Application-specific customization of routing protocols [71, 113] and techniques to provide low overhead routing algorithms with high path diversity [1, 114] have been explored. With NoCs being increasingly concerned with power, thermal and reliability issues, there exists recent work proposing thermal- [151], and reliability-aware [103] routing algorithms.

While some existing research into routing algorithms for off-chip interconnection networks can be leveraged for NoCs, the significantly different constraints of on-chip implementations lead to new challenges. First, the ultra-low latencies [56] and very high frequencies [170] of some NoCs make it difficult to incorporate sophisticated routing algorithms such as adaptive routing. The tight power constraints and reliability issues also lead to challenges in power-aware and fault-tolerant routing. With static topology irregularity, it is difficult to find minimal routes that can avoid deadlock and livelock situations (e.g., [30]). This is a research direction that needs more attention in the future. Relying on dimension-ordered routing as the escape routing function in irregular topologies becomes difficult and implementations typically require tables that incur delay, area, and power overheads [26]. To date, the vast majority of NoC routing solutions have focused on unicast traffic, that is, sending from one PE to another. Support for on-chip multicast [49] needs to be considered too, with emphasis on lightweight solutions such that the tight on-chip constraints can be met.

2.2.2 Switching Techniques

Switching, also called flow control,[1] governs the way in which messages are forwarded through the network. Typically, the messages are broken down into *flow control units* (*flits*) which represent the smallest unit of flow control. The switching algorithm then determines *if* and *when* flits should be buffered, forwarded, or simply dropped [40, 47]. As a result, the switching algorithm has the most direct impact on router microarchitecture and pipeline.

[1] The two terms "switching" and "flow control" have been used interchangeably in leading NoC texts [40, 47, 107].

Among the commonly used switching techniques in interconnection networks, wormhole switching seems the most promising for NoCs due to the limited availability of buffering resources and tight latency requirements. Virtual channels [42] are widely used in off-chip interconnection networks and are naturally adopted for NoC design to improve network bandwidth and tackle deadlock. However, as the design requirements change dramatically, the underlying substrate presents new opportunities for designing flow control algorithms.

Early work on NoC flow control aggressively drives down the router delay to a single cycle, through static compiler scheduling of network switching operations [168], dedicated look-ahead signals for setting up the switch ahead of time [56], by speculatively allocating resources to move the latency associated with resource allocation and multiplexing off the critical path at low traffic loads [111, 135], or through advanced reservation of resources [134]. While most studies focus on packet switching, several papers investigate the potential of circuit switching and time division multiplexing, to reduce the arbitration and buffering overheads of packet-switched routers [48, 62, 174].

Several techniques tackle NoC throughput such as dynamically varying the number of virtual channels (VCs) assigned to each port, to better adapt to the traffic load [117]. Express virtual channels aggressively drive down the router latency to just link latency, while extending throughput by having VCs that are statically defined to cross multiple hops [88]. Tackling latency and throughput simultaneously, layered switching in [98] hybridizes wormhole and cut-through switching by allocating groups of data words which are larger than flits but smaller than packets.

There is still a significant latency/throughput gap between the state-of-the-art NoCs and the ideal interconnect fabric of dedicated wires [88]. This disparity largely lies in the complex routers necessary at each hop for delivering ultra-low-latency and/or high bandwidth. In order for NoCs to be an efficient and effective replacement of dedicated wires as the primary communication fabric, there is a need for new switching techniques that can obviate this router overhead and truly deliver the energy-delay-throughput of dedicated wires.

2.2.3 QoS and Congestion Control

Conventional packet-switched NoCs multiplex message flows on links and share resources among these flows. While this results in high throughput, it also leads to unpredictable delays per individual message flows. For many applications with real-time deadlines, this non-determinism can substantially degrade the overall application performance. Thus, there is a need for research into NoCs that can provide deterministic bounds for communication delay and throughput.

QoS in NoCs is typically handled through three types of approaches. First, resources such as VCs can be pre-reserved with a fair mechanism for allocating resources between different traffic flows [20, 55, 94, 95, 108]. Second, multiple priority levels can be supported within the network such that the urgent traffic can have a higher priority over the regular traffic [13, 25, 63, 104]. Techniques to ensure global fairness to network hot-spots have been proposed in [93]. Finally, QoS-aware congestion control algorithms have been proposed to avoid the spikes in delay when the traffic load approaches saturation by having congestion control at the network interface regulate traffic and ensure fairness [27, 46, 119, 124].

Future CMPs and MPSoCs impose increasingly more QoS demands on NoCs; yet, support for QoS has to be extremely light weight. Cache-coherent CMPs would benefit from NoC being able to preserve the ordering semantics of a bus, thereby easing consistency support. Being able to provide guarantees on packet deliveries such as snoop responses will also ease protocol design and lower the protocol overhead. Both SoCs and CMPs will benefit from NoCs that can support dynamically defined QoS levels and needs, as well as dynamically defined partitions of the NoC that support different QoS.

Further, it will greatly strengthen the fundamental basis of NoCs to have research into analytical models that can estimate network latency and/or throughput for arbitrary traffic patterns, as these can be used to feed into QoS engines with low implementation overhead.

2.2.4 Power and Thermal Management

Due to concerns on battery lifetime, cooling and thermal budgets, power issues are at the forefront of NoC design. Seminal work on router power modeling [133] along with position papers that highlight the importance of NoC power consumption [15, 41] motivated research into low-power NoCs. There has been research into run-time NoC power management using dynamic voltage scaling on links [150], as well as shutting links down based on their actual utilization [80, 161]. Globally Asynchronous Locally Synchronous (GALS) approaches to dynamic voltage and frequency scaling further leverage the existing boundaries between various clocking domains [14, 126].

Besides average power, peak power control mechanisms for NoCs have also been explored due to their impact on thermal hotspots [18]. Power and thermal management are also very tightly related to reliability. An approach for joint power and reliability management is presented in [159], while error coding schemes for improved reliability and power consumption are presented in [16, 175].

Thermal dissipation is another metric of interest and mechanisms have been investigated to control peak power with respect to its impact on thermal dissipation [73, 151]. To this end, thermally-aware task scheduling for MPSoCs is presented in [39] and thermal optimization of 3D implementations via task scheduling and voltage scaling has been studied in [166].

With NoCs facing highly-constrained power envelopes, run-time power management techniques are needed to reduce peak power consumption so as to avoid thermal emergencies. Challenges remain in dynamically estimating the thermal hotspots as well as dynamic power profile in the presence of high workload variations. Holistic approaches covering hardware (for estimation and low level control), firmware (for implementing system level power manager) and operating system (for application characterization) are needed to tackle this problem.

Another wide open area for research is related to *distributed* control strategies for power and thermal management in NoCs. Techniques like [126, 158] are based on a centralized power manager but perhaps relying only on localized information may have its own advantages for NoCs; whether or not this is indeed the case remains to be investigated. It is important to note, again, that the accuracy of the energy models is crucial for these optimization techniques. Ideally, such models should target both dynamic and static power dissipation. While there exist preliminary efforts in this direction (e.g., [106] where adaptive body biasing is used to minimize static energy consumption), more work is needed to achieve practical solutions.

2.2.5 Reliability and Fault Tolerance

As CMOS technology approaches the nanoscale domain, there is an increasing need for studying how NoC architectures can tolerate faults and underlying process variations. For instance, shrinking transistor sizes, smaller interconnect features and higher operating frequencies of current CMOS circuits lead to higher soft-error rates and an increasing number of timing violations [154]. Moreover, the combination of smaller devices and voltage scaling in future technologies will likely result in increased susceptibility to transient faults. Therefore, in order to reduce the cost of design and verification, the future SoC architectures need to rely on fault-tolerant approaches.

Fault-tolerant multi-chip interconnection networks have been investigated, mostly in the areas of fault-tolerant routing or microarchitecture [47]. For NoCs, one of the earliest fault-tolerant communication approaches is the stochastic communication described in [23]. This approach is based on probabilistic broadcast where packets are forwarded randomly to the neighboring nodes. A theoretical model explaining the stochastic communication and relating the node coverage to the underlying properties of a grid topology was also proposed. Similarly, the studies in [138, 141] explore how NoC routing algorithms can route around faults and sustain network functionality in the presence of faults.

Researchers have also modeled the interaction between various NoC metrics, like delay, throughput, power and reliability [52]. Several studies look specifically into router design and ways to improve the NoC reliability through microarchitectural innovations that go beyond the expensive alternative of having redundant hardware [6]. The fault tolerance overhead of various flow control techniques is analyzed in [142]. Similarly, power consumption of link level and end-to-end data protection in NoCs is analyzed in [75], while energy efficiency of error correction at the receiver end is studied in [16, 115].

With devices moving into deep submicron technologies, reliability becomes a very important issue. However, research into NoC reliability is still in its infancy and thus realistic fault models that are a good representation of physical realities for NoCs are needed. Research exploring the trends in soft error rates for combinational and sequential logic (e.g., [110, 157]) can potentially be relevant to router microarchitectures as well. Future work that will critically impact the NoC power-performance-reliability trade-off includes the cost effectiveness of providing fault-tolerance, while maintaining suitable levels of fault isolation and containment.

2.3 Communication Infrastructure

2.3.1 Topology Design

The ability of the network to efficiently disseminate information depends largely on the underlying topology. Indeed, besides having a paramount effect on the network bandwidth, latency, throughput, overall area, fault-tolerance and power consumption, topology plays an important role in designing the routing strategy and mapping the IP cores to the network.

The simplicity and regularity of mesh structures makes design approaches based on such a modular topologies very attractive. More precisely, regularity improves timing closure, reduces dependence on interconnect scalability, and enables the use of high performance circuits. Typically, one-dimensional topologies (e.g., ring [136]) and two-dimensional topologies (e.g., mesh and torus [56, 167]) are the default choices for NoC designers. Node clustering to obtain topologies like the concentrated mesh [9] and hierarchical star [92] is a viable alternative to amortize the router overhead and reduce latency. Higher-radix networks like the flattened butterfly [85] reduce power and latency by reducing the number of intermediate routers and the wiring complexity over conventional butterfly but they increase the number of long wires.

Despite the benefits of regular network topologies, customization is also desirable for several reasons. First, when the size or shape of the cores varies widely, using regular topologies may waste area. Moreover, for real applications, the communication requirements of the components can vary widely. Designing the network to meet the requirements of highly communicating cores results in under utilization of other components, while designing it for the average case results in performance bottlenecks. Finally, for application-specific NoCs, a detailed a priori understanding of the communication workload can be exploited to fully customize the network topology [65, 122]. For instance, the approach proposed in [137] enables the automatic design of the communication architecture of a complex system using a library of pre-defined IP components. Similarly, the work in [164] presents a mixed integer linear programming-based technique for

NoC topology synthesis with the objective of minimizing the power consumption subject to performance constraints.

Interestingly enough, the two extreme points in the design space (i.e., completely regular and fully customized topologies) are not the only possible solutions for on-chip communication. Indeed, by inducing small world effects, the performance of regular topologies can be significantly improved with minimal impact on area and energy consumption [123]; this idea will be discussed in detail later in Chap. 6. However, for now it suffices to say that inducing small world effects via long-range links has a wide applicability as it has been demonstrated by its extension to on-chip radio-frequency links [33], express virtual channels [88] and wireless links [132].

Generally speaking, the problem of optimal topology synthesis for a given application does not have a known theoretical solution. Although the synthesis of customized architectures is desirable, distorting the regular grid structure leads to various implementation issues such as complex floorplanning, uneven wire lengths, etc. Although we have discussed only planar substrates, 3D die stacking provides the opportunity for higher radix topologies through the use of inter-die connections [84, 166, 177] and warrants further exploration.

2.3.2 Router Design

The design of a router involves determining the flow control techniques, number of virtual channels, buffer organization, switch design, pipelining strategy while adhering to target clock frequency and power budgets. All these issues require careful design since they have significant impact in terms of performance, power consumption and area.

The main focus in designing a router is to minimize the latency through it, while meeting bandwidth requirements. Reservation [134] and speculation [111, 135] can be used to hide the routing and arbitration latencies and achieve a single-stage router design. Decoupled parallel arbiters and smaller crossbars for row and column connections can reduce contention probability and reduce latency [83]. Moreover, techniques such as segmented crossbars, cut-through crossbars and write-through buffers can be used to design low power routers [172].

The impact of the number of VCs on performance varies with the network load. A lightly loaded network does not need many VCs, whereas a heavily loaded network does. A virtual channel regulator which dynamically allocates VCs and buffers according to traffic conditions, thereby reducing total buffering requirements and saving area and power, is presented in [117]. Arbitration during VC allocation is another area of potential optimization. Free virtual channel queues at each output port can effectively remove the need for VC arbitration by predetermining the order of grants [111].

An efficient algorithm for the buffer size allocation problem is proposed in [72]. The authors derive the blocking rate of each individual channel and then add more buffering resources only to the highly utilized channels. Similarly, the properties of on-chip buffers and gate-level area estimates are studied in [143]. Finally, advanced circuit-level techniques have been employed to achieve high-speed and low power operation. For instance, the router presented in [170] employs a double pumped pipeline stage to interleave alternate data bits using dual edge-triggered flip-flops; this optimization reduces the crossbar area by 50 %. Similarly, serial on-chip links, partial crossbar activation, and low energy transmission coding techniques are used in the router design presented in [92].

Tools that enable micro-architecture exploration to trade off latency and bandwidth of the router against power consumption can help NoC designers make the right design decisions for particular application requirements. Accurate performance analysis of on-chip routers under arbitrary input traffic and methodologies for choosing the correct design parameters such as optimal channel width, buffer depth, pipeline depth, number of VCs for high performance and low power remain open problems. Finally, energy-efficient routers that can interface a variety of IP cores designed for legacy communication protocols with minimal performance overhead is an important challenge.

2.3.3 Network Channel Design

The links interconnecting the network routers also need to be designed efficiently in order to consume low power and area. The ideal interconnect should be such that its performance and cost come close to that of just the network channels (or links), with the performance delivered and power consumed by the network channel largely determined by the signaling techniques.

Non-uniform channel capacity allocation is presented in [61] where the traffic is assumed to be heterogeneous and critical delay requirements vary significantly. In addition to the effects mentioned above, the choice of W has implications on the wire sizing and spacing, which affect the channel operating frequency. The bandwidth of a network channel is given by $BW = f_{ch} \times W$. Hence, bandwidth cannot be optimized by simply considering f_{ch} and W separately. Pileggi et al. [96] discuss maximizing channel throughput by controlling the size and spacing of wires, as well as their number. In [78], the authors discuss a framework for equalized interconnect design for on-chip networks. The proposed approach finds the best link design for target throughput, power and area constraints, and enables architectural optimization for energy-efficiency.

It is also of interest to explore different implementation styles for network links. For instance, delay-insensitive current mode signaling [118] as well as low-swing, differential signaling techniques [77] can be used to improve performance and reduce power.

Alternatives to wire channels present interesting opportunities for future research. For instance, analog-digital hybrid routing approaches [102], optical links [120, 149], RF interconnects [33] and wireless links [179] have been considered as alternatives to traditional on-chip repeated interconnects, but more work is needed to make such approaches applicable to real designs.

2.3.4 Floorplanning and Layout Design

Standard tile sizes help controlling the link lengths and ensuring that link delays do not limit the operating frequency. However, if the size of the network tiles varies significantly, or irregular topologies are used, the floorplanning step becomes mandatory. In this case, emphasis needs to be put on the shape and placement of tiles so as to control the link lengths. Reducing the total interconnect length is also important for reducing the power dissipation across the links. Another problem is the placement of special tiles like those connected to peripheral devices (e.g., memory controllers, I/Os) so as to minimize the average latency to these devices. In addition to link length, the goal here is to minimize link area by routing links over logic or caches as much as possible.

Layout-aware area, power and performance analysis of mesh-based NoCs is discussed in [130]. Likewise, the authors in [5] present a comparison in terms of performance, area and power scalability between crossbar designs within a pre-existing communication fabric and the NoC approach at layout level. Considering physical layout is also important while solving mapping and topology synthesis problems. To this end, floorplan aware solutions to these problems are presented in [112, 163]. We note that the size of the IP cores are assumed to be fixed in the SoC context. On the other hand, this does not necessarily hold for the CMP context where one can make trade-off between cache size and interconnect area [89]. For example, larger private caches can filter traffic and reduce the requirement on the network, while performance drawback of smaller caches can be mitigated by higher performance NoCs.

2.3.5 Clocking and Power Distribution

The traditional approach of designing fully synchronous chips with a single global clock is not attractive anymore due to smaller process geometries, larger wire delays, higher levels of integration of multiple cores on large dies. The large effort required for skew control and the significant power consumption of the global clock call for alternative clocking strategies [12]. Indeed, in addition to multiple frequencies, different cores can have their own optimal supply voltage to allow for fine-grain power/performance management.

Strategies such as asynchronous or mesochronous clocking [8, 22, 152] are alternatives that hold the promise of simplifying timing closure and global clock distribution. For instance, an approach to minimize the strict skew requirements without going fully asynchronous using all-in-phase clocking is presented in [153].

In [22], the authors present a mesochronous clocking strategy that avoids timing related errors while maintaining globally synchronous system perspective. The 80-tile teraflop NoC presented in [170] employs phase tolerant mesochronous interfaces between the routers with FIFO-based synchronization. Similarly, latency insensitive or synchronous elastic systems are developed to exploit the inherent advantages of synchronous design while decoupling the functionality from the channel delays [29, 38].

The GALS approach has been used with several tile-based multiprocessor implementations [178]. However, there are extra costs in terms of synchronization latency and power that need to be considered. Chelcea et al. [35] discuss interfacing different clock domains which is essential for implementing globally asynchronous systems. A systematic comparison between asynchronous and GALS implementations of an NoC is presented in [152]. The authors conclude that while the two approaches result in similar silicon area, power consumption and bandwidth, the asynchronous implementation has a clear advantage in terms of average packet latency. In [28], the authors propose asynchronous delay insensitive links to support the GALS NoC paradigm. This approach removes the constraints on wire propagation delays and enables designing links of any width with low wire and logic overhead.

Open problems in this area include robust design of clock crossing synchronizers with minimal latency penalty and low power consumption, since locally generated clocks for GALS SoCs are prone to syncronization failures due to clock delays [44]. Recent research in resonant clocking shows promise for reducing power and delivering high performance [32]; however, the use of resonant clocking with NoCs has yet to be investigated. Also, while controlling NoCs with multiple voltage-frequency islands has been discussed in [126], techniques that consider other control objectives such as chip temperature, power consumption, and nonlinear effects are needed.

2.4 NoC Evaluation and Validation

2.4.1 Analysis and Simulation

Fast and accurate approaches for analyzing critical metrics such as performance, power consumption or system fault-tolerance are important to guide the design process. However, in order to be used within an optimization loop or make early design choices, the analysis techniques need to be tractable and provide meaningful feedback to designers. At later design phases, one can obtain more accurate estimates through simulation.

Communication latency and network bandwidth are common performance metrics of interest. While it is relatively easier to find the communication latency for guaranteed service traffic [43, 108], analyzing the average latency for best effort traffic is a challenging task. Therefore, the average hop count or free packet delay are commonly used to approximate the average packet latency [113, 164]. Techniques for analyzing the average communication latency in networks are proposed in [45, 67]. While not directly applicable to NoC performance analysis, these approaches can be used as a starting point and then account for NoC-specific constraints, such as application specific traffic and on-chip router parameters.

Analytical power models for early-stage power estimation in NoCs have also been investigated, starting with [133] which models the power consumed of multichip interconnection networks; this generated follow-up work that specifically targets on-chip network power dissipation [10, 31, 36, 51, 82, 151].

Performance analysis largely depends on various simplifying assumptions on the network or traffic characteristics (e.g., uniform traffic vs. bursty traffic) and typically assumes deterministic routing due to the difficulty in handling the more general problem. Approaches that relax the Markovian assumption and analytical power consumption models that accurately account for the application and architecture characteristics are highly needed [24].

Simulation-based approaches are still popular for architectural exploration of on-chip networks due to their accuracy, flexibility and ability to run real workloads [86, 87, 100, 129, 131, 173]

The major issue with simulation-based approaches is the trade-off between the level of implementation detail and simulation time [74]. Detailed models can deliver very accurate results, but the simulation time can be prohibitive. Realistic synthetic trace simulation [101, 171] or hardware acceleration [53] can be used for improving the simulation speed therefore these are wide open directions for research.

2.5 Prototyping, Testing and Verification

While simulation offers flexibility for power-performance evaluations under various network parameters, it still relies on many approximations that may affect the accuracy of the results. Prototyping can be further used to improve the evaluation accuracy by bringing the design closer to reality, at the expense of increased implementation effort and reduced flexibility. Finally, it is also important that testing and verification must be considered to ensure correctness.

Several concrete NoC architectures have been presented in the literature. In [2], the authors present the SPIN interconnect architecture and implement a 32-port network architecture using a 0.13um process. This architecture uses credit-based flow control to provide QoS.

A flexible FPGA-based NoC design that consists of processors and reconfigurable components is presented in [11]. The FPGA prototype presented in [123] illustrates the impact of application-specific long-range links on the performance and energy consumption of 2D mesh networks. The aSoC architecture presented in [95] supports compile-time scheduling for on-chip communication and provides software-based dynamic routing. The RAW chip [167], attacks the wire-delay problem by proposing a direct software interface to the physical resources. The static network used in the RAW chip also enables new application domains.

The 80-core teraflops chip recently introduced by Intel [170] is a good example of a major NoC prototyping effort. The chip uses a 2D mesh with mesochronous clocking for a high bandwidth, scalable design. The authors in [92] present a highly optimized NoC implementation using hierarchical star topology. Finally, the work presented in [79] addresses both architectural aspects and circuit level techniques for practical NoC implementation.

We note though that most studies dealing with concrete NoC implementations lack performance evaluation under *real* driver applications. This is an important issue that needs to be addressed in order to bring the NoC prototypes closer to real applications. Towards this end, the authors in [91] compare and contrast the NoC, bus- and P2P-based implementations of an MPEG-2 encoder using an FPGA-based prototype. The advantages of the NoC approach are illustrated in terms of scalability, throughput, energy consumption and area, both analytically and using direct measurements on the prototype.

In NoCs, the routers and links have been utilized to test the PEs and the network itself based on built-in self-test mechanisms [4, 57, 66, 97]. In [4], the authors propose a scalable test strategy for the routers in an NoC, based on partial scan and an IEEE 1500-compliant test wrapper. Similarly, the strategy proposed in [66] exploits the regularity of the switches, and broadcasts the test vectors through the minimum spanning tree to test the switches concurrently. The authors in [57] propose testing the routing logic and FIFO buffers recursively by utilizing the NoC component that already passed the test. The work presented in [97] also considers the power consumption required for testing purposes.

NoC verification has received less attention compared to other design aspects or even testing. The NoC verification approach in [55] relies on monitoring the network traffic and checking special events such as connection opened/closed, data received by a connection, etc. Likewise, the MAIA framework aims at automated generation and verification of NoC architectures [128]. More recently, formal verification of asynchronous NoC architectures is considered in [144], while a framework for quick formal modeling and verification of communication fabrics is presented in [34].

References

1. Abad P, Puente V, Gregorio JA, Prieto P (2007) Rotary router: An efficient architecture for CMP interconnection networks. In: Proceedings of the international symposium on computer architecture, June 2007
2. Adriahantenaina A, Greiner A (2003) Micro-network for SoC: implementation of a 32-Port SPIN network. In: Proceedings of design, automation and test in Europe conference, March 2003

3. Al Faruque MA, Ebi T, Henkel J (2007) Run-time adaptive on-chip communication scheme. In: Proceedings of IEEE/ACM international conference on computer-aided design (ICCAD'07), San Jose, California, USA, 26–31, 2007
4. Amory AM, Briao E, Cota E, Lubaszewski M, Moraes FG (2005) A scalable test strategy for network-on-chip routers. In: Proceedings of IEEE international test conference, Nov 2005
5. Angiolini F, Meloni P, Carta S, Benini L, Raffo L (2006) Contrasting a NoC and a traditional interconnect fabric with layout awareness. In: Proceedings of design, automation and test in Europe conference, March 2006
6. Angiolini F, Atienza D, Murali S, Benini L, De Micheli G (2006) Reliability support for on-chip memories using networks-on-chip. In: Proceedings of the international conference on computer design, Oct 2006
7. Ascia G, Catania V, Palesi M (2004) Multi-objective mapping for mesh-based NoC architectures. In: Proceedings of international conference on hardware-software codesign and system synthesis, Sept 2004
8. Bainbridge W, Furber S (2001) Delay insensitive system-on-chip interconnect using 1-of-4 data encoding. In: Proceedings of international symposium on asynchronous circuits and systems, March 2001
9. Balfour J, Dally WJ (2006) Design tradeoffs for tiled CMP on-chip networks. In: Proceedings of the international conference on supercomputing, June 2006
10. Banerjee N, Vellank P, Chatha KS (2004) A power and performance model for network-on-chip architectures. In: Proceedings of design, automation and test in Europe conference, Feb 2004
11. Bartic TA et al (2003) Highly scalable network on chip for reconfigurable systems. In: Proceedings of international symposium system-on-chip, Nov 2003
12. Beerel P, Roncken ME (Dec. 2007) Low power and energy efficient asynchronous design. J Low Power Electron 3(3):234–253
13. Beigne E, Clermidy F, Vivet P, Clouard A, Renaudin M (2005) An asynchronous NOC architecture providing low latency service and its multi-level design framework. In: Proceedings of international symposium on asynchronous circuits and systems, May 2005
14. Beigne E, Clermidy F, Miermont S, Vivet P (2008) Dynamic voltage and frequency scaling architecture for units integration with a GALS NoC. In: Proceedings of IEEE international symposium on network on chip, 2008
15. Benini L, De Micheli G (Jan. 2002) Networks on chips: a new SoC paradigm. IEEE Comput 35(1):70–78
16. Bertozzi D, Benini L, De Micheli G (2005) Error control schemes for on-chip communication links: the energy-reliability tradeoff. IEEE Trans Comput Aided Des Integr Circuits Syst 24(6):818–831
17. Bertozzi S, Acquaviva A, Bertozzi D, Poggiali A (2006) Supporting task migration in multi-processor systems-on-chip: a feasibility study. In: Proceedings of design, automation and test in Europe conference March 2006
18. Bhojwani P, Lee JD, Mahapatra R (2007) SAPP: scalable and adaptable peak power management in NoCs. In: Proceedings of international symposium on low power electronic devices, Aug 2007
19. Bienia C, Kumar S, Singh JP, Li K (2008) The PARSEC benchmark suite: characterization and architectural implications. Princeton University Technical Report TR-811-08, Jan 2008
20. Bjerregaard T, Sparso J (2005) A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In: Proceedings of design, automation and test in Europe conference, March 2005
21. Bjerregaard T, Mahadevan S (2006) A survey of research and practices of Network-on-chip. ACM Comput Surv 38(1):1–51
22. Bjerregaard T, Stensgaard MB, Sparso J (2007) A scalable, timing-safe, network-on-chip architecture with an integrated clock distribution method. In: Proceedings of design, automation and test in Europe conference, April 2007

23. Bogdan P, Dumitras T, Marculescu R (2007) Stochastic communication: a new paradigm for fault-tolerant networks-on-chip. Hindawi VLSI design, special issue on networks-on-chip, vol 2007, Hindawi Publishing Corporation
24. Bogdan P, Marculescu R (2010) Workload characterization and its impact on multicore platform design.In: Proceedings of 8th IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis (CODES/ISSS), 2010
25. Bolotin E, Cidon I, Ginosar R, Kolodny A (Feb. 2004) QNoC: QoS architecture and design process for network on chip. J Syst Architecture (EUROMICRO J) 50(2–3):105–128
26. Bolotin E, Cidon I, Ginosar R, Kolodny A (2007) Routing table minimization for irregular mesh NoCs. In: Proceedings of design, automation and test in Europe conference, April 2007
27. van den Brand JW, Ciordas C, Goossens K, Basten T (2007) Congestion-controlled best-effort communication for networks-on-chip. In: Proceedings of design, automation and test in Europe conference, April 2007
28. Campobello G, Castano M, Ciofi C, Mangano D (2006) GALS networks on chip: a new solution for asynchronous delay-insensitive links. In: Proceedings of design, automation and test in Europe conference, March 2006
29. Carloni LP, McMillan KL, Sangiovanni-Vincentelli AL (Sep. 2001) Theory of latency-insensitive design. IEEE Trans Comput Aided Des Integr Circuits Syst 20(9):1059–1076
30. Catania V, Holsmark R, Kumar S, Palesi M (2006) A methodology for design of application specific deadlock-free routing algorithms for NoC systems. In: Proceedings of CODES-ISSS, Oct 2006
31. Chan J, Parameswaran S (2005) NoCEE: energy macro-model extraction methodology for network on chip routers. In: Proceedings the of international conference on computer aided design, Nov 2005
32. Chan SC, Shepard KL, Restle PJ (2003) Design of resonant global clock distributions. In: Proceedings of the international conference on computer design, Oct 2003
33. Chang MF et al (2008) CMP network-on-chip overlaid with multi-band RF-interconnect. In: Proceedings of the international symposium on high-performance computer architecture, Feb 2008
34. Chatterjee S, Kishinevsky M, Ogras UY (2010) Quick formal modeling of communication fabrics to enable verification. In: Proceedings of IEEE international high level design validation and test workshop, 42–49 June 2010
35. Chelcea T, Nowick SM (2000) A low latency fifo for mixed-clock systems. In: Proceedings of IEEE computer society workshop on VLSI, April 2000
36. Chen X, Peh L (2003) Leakage power modeling and optimization in interconnection networks. In: Proceedings of the international symposium on low power electronics and design, Aug 2003
37. Chou C-L, Ogras UY, Marculescu R (2008) Energy- and performance-aware incremental mapping for networks-on-chip with multiple voltage levels. IEEE Trans Comput Aided Des Integr Circuits Syst (TCAD) 27(10):1866–1879
38. Cortadella J, Kishinevsky M, Grundmann B (2006) Synthesis of synchronous elastic architectures. In: Proceedings of design, automation conference, July 2006
39. Coskun AK, Rosing TS, Whisnant K (2007) Temperature aware task scheduling in MPSoCs. In: Proceedings of design, automation and test in Europe conference, April 2007
40. Dally WJ, Towles B (2004) Principles and practices of interconnection networks. Morgan Kaufmann Press, San Francisco
41. Dally WJ, Towles B (2001) Route packets, not wires: on-chip interconnection networks. In: Proceedings of design automation conference, June 2001
42. Dally WJ (1992) Virtual-channel flow control. IEEE Trans Parallel Distrib Syst 3(2):194–205
43. Dielissen J, Radulescu A, Goossens K, Rijpkema E (2003) Concepts and implementation of the Philips network-on-chip. In: Proceedings of IP-based SoC design, 2003

44. Dobkin R, Ginosar R, Sotiriou C (2004) Data synchronization issues in GALS SoCs. In: Proceedings of international symposium on asynchronous circuits and systems, April 2004
45. Draper J, Ghosh J (1994) A comprehensive analytical model for wormhole routing in multicomputer systems. J Parallel Distrib Comput 23(2):202–214
46. Duato J et al (2005) A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. In: Proceedings of the international symposium on high-performance computer architecture, Feb 2005
47. Duato J, Yalamanchili S, Ni L (2002) Interconnection networks: an engineering approach. Morgan Kaufmann, San Mateo, CA
48. Enright-Jerger N, Peh L, Lipasti M (2008) Circuit-switched coherence. In: Proceedings of the international symposium networks-on-chips, May 2008
49. Enright-Jerger N, Peh L-S, Lipasti M (2008) Virtual circuit tree multicasting: a case of on-chip hardware multicast support. In: Proceedings of ISCA, June 2008
50. Enright-Jerger N, Peh L (2009) On-chip networks. Synthesis lecture. Morgan-Claypool Publishers
51. Eisley N, Peh L (2004) High-level power analysis for on-chip networks. International conference on compilers, architectures and synthesis for embedded systems, Sep 2004
52. Ejlali A, Al-Hashimi BM, Rosinger P, Miremadi SG (2007) Joint consideration of fault-tolerance, energy-efficiency and performance in on-chip networks. In: Proceedings of design, automation and test in Europe conference, April 2007
53. Genko N, De Micheli G, Atienza D, Mendias J, Hermida R, Catthoor F (2005) A complete network-on-chip emulation framework. In: Proceedings of design, automation and test in Europe conference, March 2005
54. Goldfeder CM (2005) Frequency-based code placement for embedded multiprocessors. In: Proceedings of design automation conference, July 2005
55. Goossens K et al (2005) A design flow for application-specific networks-on-chip with guaranteed performance to accelerate SoC design and verification. In: Proceedings of design, automation and test in Europe conference, March 2005
56. Gratz P, Kim C, McDonald R, Keckler SW, Burger DC (2006) Implementation and evaluation of on-chip network architectures. In: Proceedings of international conference on computer design, Oct 2006
57. Grecu C, Pande PP, Wang B, Ivanov A, Saleh R (2005) Methodologies and algorithms for testing switch-based NoC interconnects. In: Proceedings of international symposium on defect and fault tolerance in VLSI systems, Oct 2005
58. Grecu C, Ivanov A, Pande P, Jantsch A, Salminen E, Ogras UY, Marculescu R (2007) An initiative towards open network-on-chip benchmarks. NoC benchmarking white paper, 2007. http://www.ocpip.org/uploads/documents/NoC-Benchmarks-WhitePaper-15.pdf
59. Gruian F (2001) Hard real-time scheduling for low energy using stochastic data and DVS processors. In: Proceedings of international symposium on low-power electronics and design, Aug 2001
60. Guerrier P, Greiner A (2000) A generic architecture for on-chip packet switched interconnections. In: Proceedings of design, automation and test in Europe conference, March 2000
61. Guz Z, Walter I, Bolotin E, Cidon I, Ginosar R, Kolodny A (2006) Efficient link capacity and QoS design for wormhole network-on-chip. In: Proceedings of design, automation and test in Europe conference, March 2006
62. Hansson A, Goossens K, Radulescu A (2007) A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic. Hindawi VLSI Design, Hindawi Publishing Corporation
63. Harmanci M, Escudero N, Leblebici Y, Ienne P (2005) Quantitative modeling and comparison of communication schemes to guarantee quality-of-service in networks-on-chip. In: Proceedings of the international symposium on circuits and systems, May 2005

64. Hemani A, Jantsch A, Kumar S, Postula A, Oberg J, Millberg M, Lindvist D (2000) Network on a chip: an architecture for billion transistor era. In: Proceedings of the IEEE NorChip conference, Nov 2000
65. Ho WH, Pinkston TM (2003) A methodology for designing efficient on-chip interconnects on well-behaved communication patterns. In: Proceedings of the international symposium on high-performance computer, architecture, Feb 2003
66. Hosseinabady M, Dalirsani A, Navabi Z (2007) Using the inter- and intra-switch regularity in NoC switch testing. In: Proceedings of design, automation and test in Europe conference, April 2007
67. Hu P, Kleinrock L (1997) An analytical model for wormhole routing with finite size input buffers. 15th International teletraffic congress, June 1997
68. Hu J, Marculescu R (2003) Energy-aware mapping for tile-based NoC architectures under performance constraints. In: Proceedings of ASP-DAC, Jan 2003
69. Hu J, Marculescu R (2005) Communication and task scheduling of application-specific networks-on-chip. IEE Proc comput Digital Tech 152(5):643–651
70. Hu J, Marculescu R (2004) DyAD—Smart routing for networks-on-chip. In: Proceedings of design automation conference, June 2004
71. Hu J, Marculescu R (2005) Energy- and performance-aware mapping for regular NoC architectures. IEEE Trans Comput Aided Des Integr Circuits Syst 24(4):551–562
72. Hu J, Ogras UY, Marculescu R (2006) System-level buffer allocation for application-specific networks-on-chip router design. IEEE Trans Comput Aided Des Integr Circuits Syst 25(12):2919–2933
73. Hung W et al (2004) Thermal-aware IP virtualization and placement for Networks-on-Chip architecture. In: Proceedings of ICCD, 2004
74. Ibrahim KZ (2005) Correlation between detailed and simplified simulations in studying multiprocessor architecture. In: Proceedings of international conference on computer design, Oct 2005
75. Jantsch A, Lauter R, Vitkowski A (2005) Power analysis of link level and end-to-end data protection in networks on chip. In: Proceedings of the international symposium on circuits and systems, May 2005
76. Jantsch A, Tenhunen H (eds) (2003) Networks-on-Chip. Norwell, MA, Kluwer
77. Jose AP, Patounakis G, Shepard KL (2005) Near speed-of-light on-chip interconnects using pulsed current-mode signaling. In: Proceedings of symposium on VLSI Circuits, June 2005
78. Kim B, Stojanovic V (2007) Equalized interconnects for on-chip networks: modeling and optmization framework. International conference on computer-aided design, Nov 2007
79. Kim D, Kim K, Kim J, Lee S, Yoo H (2007) Solutions for real chip implementation issues of NoC and their application to memory-centric NoC. In: Proceedings of international symposium on networks-on-chips, May 2007
80. Kim EJ et al (2003) Energy optimization techniques in cluster interconnects. In: Proceedings of the international symposium on low power electronics and design, Aug 2003
81. Kim M, Kim D, Sobelman GE (2005) Adaptive scheduling for CDMA-based networks-on-chip. In: Proceedings of the IEEE northeast workshop on circuits and systems, May 2005
82. Kim JS, Taylor MB, Miller J, Wentzlaff D (2003) Energy characterization of a tiled architecture processor with on-chip networks. In: Proceedings of the international symposium on low power electronics and design, Aug 2003
83. Kim J, Nicopoulos CA, Park D, Vijaykrishnan N, Yousif MS, Das CR (2006) A gracefully degrading and energy-efficient modular router. In: Proceeings of the international symposium on computer architecture, June 2006
84. Kim J et al (2007) A novel dimensionally-decomposed router for on-chip communication in 3D architectures. In: Proceedings of the international symposium on computer architecture, June 2007
85. Kim J, Dally WJ, Abts D (2007) Flattened butterfly: a cost-efficient topology for high-radix networks. In: Proceedings of ISCA, June 2007

86. Kogel T et al (2003) A modular simulation framework for architectural exploration of on-chip interconnection networks. In: Proceedings of international conference on hardware-software codesign and system, synthesis, Oct 2003
87. Kogel T, Leupers R, Meyr H (2006) Integrated system-level modeling of network-on-chip enabled multi-processor platforms. Springer, New York
88. Kumar A, Peh L, Kundu P, Jha NK (2007) Express virtual channels: Towards the ideal interconnection fabric. In: Proceedings of the international symposium on computer architecture, June 2007
89. Kumar R, Zyuban V, Tullsen DM (2005) Interconnections in multi-core architectures: understanding mechanisms, overheads and scaling. In: Proceedings of the international symposium on computer architecture, June 2005
90. Lahiri K et al (2000) Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. In: Proceedings of the international conference on VLSI design, Oct 2000
91. Lee HG, Chang N, Ogras UY, Marculescu R (2007) On-chip communication architecture exploration: a quantitative evaluation of point-to-point, bus and network-on-chip approaches. ACM Trans Des Autom Electron Syst 12(3):1–20
92. Lee K et al (2004) A 51mW 1.6GHz on-chip network for low-power heterogeneous SoC platform. International Solid-State Circuits Conference, Feb 2004
93. Lee JW, Ng A, Asanovic K (2008) Globally-synchronized frames for guaranteed quality of service in on-chip networks. In: International symposium on computer architecture, 2008
94. Leung LF, Tsui CY (2006) Optimal link scheduling on improving best-effort and guaranteed services performance in network-on-chip system. In: Proceedings of design automation conference, July 2006
95. Liang J, Laffely A, Srinivasan S, Tessier R (2004) An architecture and compiler for scalable on-chip communication. IEEE Trans Very Large Scale Integr Syst 12(7):711–726
96. Lin T, Pileggi LT (2002) Throughput-driven IC communication fabric synthesis. In: Proceedings of the international conference on computer aided design, 2002
97. Liu C, Shi J, Cota E, Iyengar V (2005) Power-aware test scheduling in network-on-chip using variable-rate on-chip clocking. In: Proceedings of VLSI test symposium, May 2005
98. Lu Z, Liu M, Jantsch A (2007) Layered switching for networks on chip. In: Proceedings of design automation conference, June 2007
99. Luo J, Jha NK (2000) Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In: Proceedings of international conference on computer-aided design, Nov 2000
100. Madsen J, Mahadevan S, Virk K, Gonzales M (2003) Network-on-chip modeling for system-level multiprocessor simulation. In: Proceedings of the IEEE international real-time systems symposium, 82–92, Dec 2003
101. Mahadevan S et al (2005) A network traffic generator model for fast network-on-chip simulation. In Proceedings of design, automation and test in Europe conference, March 2005
102. Mak TS, Sedcole P, Cheung PY, Luk W, Lam KP (2007) A hybrid analog-digital routing network for NoC dynamic routing. In: Proceedings of the international symposium on networks-on-chip, May 2007
103. Manolache S, Eles P, Peng Z (2005) Fault and energy-aware communication mapping with guaranteed latency for applications implemented on NoC. In: Proceedings design automation conference, July 2005
104. Marescaux T, Corporaal H (2007) Introducing the superGT network-on-chip. In: Proceedings of design automation conference, June 2007
105. Marculescu R, Ogras UY, Peh L, Jerger NE, Hoskote Y (2009) Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. IEEE Trans Comput Aided Des Integr Circuits Syst 28(1):3–21

106. Martin S, Flautner K, Mudge T, Blaauw D (2002) Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In: Proceedings of international conference on computer aided design, Nov 2002
107. De Micheli G, Benini L (eds) (2006) Networks on chips: technology and tools (systems on silicon). Morgan Kaufmann, San Francisco
108. Millberg M, Nilsson E, Thid R, Jantsch A (2004) Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In: Proceedings of design, automation and test in Europe conference, Feb 2004
109. Mishra R et al (2003) Energy aware scheduling for distributed real-time systems. International parallel and distributed processing symposium, April 2003
110. Miskov-Zivanov N, Marculescu D (2010) Multiple transient faults in combinational and sequential circuits: a systematic approach. IEEE Trans CAD Integr Cir Syst 29(10):1614–1627
111. Mullins R, West A, Moore S (2004) Low-latency virtual-channel routers for on-chip networks. In: Proceedings of international symposium on computer architecture, June 2004
112. Murali S et al (2006) Designing application-specific networks on chips with floorplan information. In: Proceedings of ICCAD, Nov 2006
113. Murali S, De Micheli G (2004) Bandwidth-constrained mapping of cores onto NoC architectures. In: Proceedings of design, automation and test in Europe conference, Feb 2004
114. Murali S, Atienza D, Benini L, De Micheli G (2007) A method for routing packets across multiple paths in NoCs with in-order delivery and fault-tolerance guarantees. Hindawi VLSI Des 2007:11
115. Murali S et al (2005) Analysis of error recovery schemes for networks on chip. IEEE design and test of computers, 2005
116. Murali S, Coenen M, Radulescu A, Goossens K, De Micheli G (2006) A methodology for mapping multiple use-cases onto networks on chips. In: Proceedings of design automation and test in Europe conference, March 2006
117. Nicopoulos CA et al (2006) ViChaR: a dynamic virtual channel regulator for network-on-chip routers. In: Proceedings of the international symposium on microarchitecture, Dec 2006
118. Nigussie E, Lehtonen T, Tuuna S, Plosila J, Isoaho J (2007) High-performance long NoC link using delay-insensitive current-mode signaling. Hindawi VLSI Des (special issue on networks-on-chip) 2007:1–13
119. Nilsson E, Millberg M, Oberg J, Jantsch A (2003) Load distribution with the proximity congestion awareness in a network on chip. In: Proceedings of design, automation and test in Europe conference, March 2003
120. Connor IO, Gaffiot F (2004) Advanced research in on-chip optical interconnects. In: Piguet C (ed) Lower Power electronics and design, CRC Press
121. OCP International Partnership, http://www.ocpip.org/university_research_bibliography.php
122. Ogras UY, Marculescu R (2005) Energy- and performance-driven NoC communication architecture synthesis using a decomposition approach. In: Proceedings of design, automation and test in Europe conference, March 2005
123. Ogras UY, Marculescu R (2006) It's a small world after all": NoC performance optimization via long-range link insertion. IEEE Trans Very Large Scale Integr Syst Spec Sect Hardw Softw Codesign Syst Synth 14(7):693–706
124. Ogras UY, Marculescu R (2006) Prediction-based flow control for network-on-chip traffic. In: Proceedings of design automation conference, July 2006
125. Ogras UY, Marculescu R (2006) Communication-based design for nanoscale SoCs. In: Chen W-K (ed) VLSI handbook, 2nd edn. CRC Book Press
126. Ogras UY, Marculescu R, Marculescu D, Jung EG (2009) Design and management of voltage-frequency island partitioned networks-on-chip. IEEE Trans Very Large Scale Integr Syst 17(3):330–341

127. On-Chip Networks Bibliography, http://www.cl.cam.ac.uk/∼rdm34/onChipNetBib/browser.htm
128. Ost L, Mello A, Palma J, Moraes F, Calazans N (2005) MAIA: a framework for networks on chip generation and verification. In: Proceedings of Asia South Pacific design automation conference, Jan 2005
129. Palermo G, Silvano C (2004) PIRATE: a framework for power/performance exploration of network-on-chip architectures. In: Proceedings of international workshop on power and timing modeling, optimization and simulation, Sept 2004
130. Pamunuwa D, Öberg J, Zheng LR, Millberg M, Jantsch A, Tenhunen H (2003) Layout, performance and power trade-offs in mesh-based network-on-chip architectures. In: IFIP international conference on very large scale integration, Dec 2003
131. Pande PP, Grecu C, Jones M, Ivanov A, Saleh R (Aug. 2005) Performance evaluation and design trade-offs for network-on-chip interconnect architectures. IEEE Trans Comput 54(8):1025–1040
132. Ganguly A et al (2010) Scalable hybrid wireless network-on-chip architectures for multicore systems. IEEE Trans Comput 60(10):1485–1502
133. Patel CS, Chai SM, Yalamanchili S, Schimmel DE (1997) Power constrained design of multiprocessor interconnection networks. In: Proceedings of the international conference on computer design, Oct 1997
134. Peh L, Dally WJ (2000) Flit-reservation flow control. In: Proceedings of the international symposium on high-performance computer architecture, Jan 2000
135. Peh L, Dally WJ (2001) A delay model for router micro-architectures. IEEE Micro
136. Pham D et al (2005) The design and implementation of a first-generation CELL processor. In: Proceedings of the solid-state circuits conference, Feb 2005
137. Pinto A, Carloni LP, Sangiovanni-Vincentelli AL (2003) Efficient synthesis of networks on chip. In: Proceedings of international conference on computer design , Oct 2003
138. Pirretti M, Link GM, Brooks RR, Vijaykrishnan N, Kandemir M, Irwin MJ, (2004) Fault tolerant algorithms for network-on-chip interconnect. In: Proceedings of IEEE symposium on VLSI, Feb 2004
139. Pop P et al (2001) An approach to incremental design of distributed embedded systems. In: Proceedings of design automation conference, June 2001
140. Poplavko P, Basten T, Bekooij M, van Meerbergen J, Mesman B (2003) Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In: Proceedings of the international conference on compilers, architecture and synthesis for embedded systems, 2003
141. Puente V, Gregorio JA, Vallejo F, Beivide R (2004) Immunet: a cheap and robust fault-tolerant packet routing mechanism. In: Proceedings of the international symposium on computer, architecture, June 2004
142. Pullini A, Angiolini F, Bertozzi D, Benini L (2005) Fault tolerance overhead in network-on-chip flow control schemes. In: Proceedings of symposium on integrated circuits and system design, Sep 2005
143. Saastamoinen I, Alho M, Nurmi J (2003) Buffer implementation for proteo network-on-chip. In: Proceedings of international symposium on circuits and systems, May 2003
144. Salaun G, Serwe W, Thonnart Y, Vivet P (2007) Formal verification of CHP specifications with CADP illustration on an asynchronous network-on-chip. In: Proceedings of the IEEE international symposium on asynchronous circuits and systems, 2007
145. Scherrer A, Fraboulet A, Risset T (2006) Automatic phase detection for stochastic on-chip traffic generation. In: Proceedings International Conference on Hardware-Software Codesign, Oct 2006, pp 88–93
146. Schmitz MT, Al-Hashimi BM, Eles P (2004) Iterative schedule optimization for voltage scalable distributed embedded systems. ACM Trans Embedded Comput Syst 3(1):182–217. doi:10.1145/972627.972636

147. Seo D, Ali A, Lim W, Rafique N, Thottethodi M (2005) Near-optimal worst-case throughput routing for two-dimensional mesh networks. In: Proceedings of the international symposium on computer, architecture, June 2005
148. Sgroi M et al (2001) Addressing the system-on-a-chip interconnect woes through communication-based design. In: Proceedings of design automation conference, June 2001
149. Shacham A, Bergman K, Carloni LP (2007) The case for low-power photonic networks-on-chip. In: Proceedings of design automation conference, June 2007
150. Shang L, Peh L, Jha NK (2003) Dynamic voltage scaling with links for power optimization of interconnection networks. In: Proceedings of the international symposium on high-performance computer, architecture, Jan 2003
151. Shang L, Peh L, Kumar A, Jha N K (2004) Thermal modeling, characterization and management of on-chip networks. In: Proceedings of international symposium on microarchitecture, Dec 2004
152. Sheibanyrad A, Panades IM, Greiner A (2007) Systematic comparison between the asynchronous and the multi-synchronous implementations of a network-on-chip architecture. In: Proceedings of design, automation and test in Europe conference, April 2007
153. Shibayama A, Nose K, Torii S, Mizuno M, Edahiro M (2007) Skew-tolerant global synchronization based on periodically all-in-phase clocking for multi-core soc platforms. In: Proceedings of symposium on VLSI circuits, June 2007
154. Shim B, Shanbhag NR (2006) Energy-efficient soft-error tolerant digital signal processing. IEEE Trans VLSI 14(4):336–348
155. Shin D, Kim J (2004) Power-aware communication optimization for networks-on-chips with voltage scalable links. In: Proceedings of international conference on hardware/software codesign and system synthesis, Sept 2004
156. Shin D, Kim J, Lee S (2001) Intra-task voltage scheduling for low-energy, hard real-time applications. IEEE Des Test 18(2):20–30
157. Shivakumar P, Kistler M, Keckler S, Burger D, Alvisi L (2002) Modeling the effect of technology trends on soft error rate of combinational logic. In: Proceedings of the international conference on dependable systems and networks, June 2002
158. Simunic T, Boyd S (2002) Managing power consumption in networks on chip. In: Proceedings of design, automation and test in Europe conference, March 2002
159. Simunic Rosing T, Mihic K, De Micheli G (2007) Power and reliability management of SOCs. IEEE Trans on VLSI 15:391–403
160. Soteriou V, Wang H-S, Peh L (2006) A statistical traffic model for on-chip interconnection networks. In: Proceedings of the international symposium on modeling, analysis and simulation of computer and telecommunication systems, Sept 2006
161. Soteriou V, Peh L (2004) Design space exploration of power-aware on/off interconnection networks. In: Proceedings of the ICCD, Oct 2004
162. Srinivasan K, Chatha KS (2005) A technique for low energy mapping and routing in network-on-chip architectures. In: Proceedings of the international symposium on low power electronics and design, Aug 2005
163. Srinivasan K, Chatha KS (2006) A low complexity heuristic for design of custom network-on-chip architectures. In: Proceedings of design, automation and test in Europe conference, March 2006
164. Srinivasan K, Chatha KS, Konjevod G (2006) Linear programming based techniques for synthesis of network-on-chip architectures. IEEE Trans on Very Large Scale Integr Syst 14(4):407–420
165. Stuijk S, Basten T, Geilen M, Ghamarian AH, Theelen B (2008) Resource-efficient routing and scheduling of time-constrained streaming communication on networks-on-chip. J Syst Architect (the EUROMICRO Journal) 54(3–4):411–426
166. Sun C, Shang L, Dick RP (2007) Three-dimensional multi-processor system-on-chip thermal optimization. In: Proceedings of international conference on hardware/software codesign and system synthesis, Oct 2007

167. Taylor MB et al (2002) The Raw microprocessor: A computational fabric for software circuits and general purpose programs. IEEE Micro
168. Taylor MB, Lee W, Amarasinghe S, Agarwal A (2005) Scalar operand networks. IEEE Trans Parallel Distrib Syst (special issue on on-chip networks) 16(2):145–162
169. Towles B, Dally WJ (2002) Worst-case traffic for oblivious routing functions. ACM Symp Parallel Algori Architect
170. Vangal S et al (2007) An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In: Proceedings of solid-state circuits conference, Feb 2007
171. Varatkar G, Marculescu R (2004) On-chip traffic modeling and synthesis for MPEG-2 video applications. IEEE Trans VLSI 12(1):108–119
172. Wang H, Peh L, Malik S (2003) Power-driven design of router microarchitectures in on-chip networks. In: Proceedings of the international symposium on microarchitecture, Nov 2003
173. Wang H, Zhu X, Peh L, Malik S (2002) Orion: a power-performance simulator for interconnection networks. In: Proceedings of annual international symposium on microarchitecture, Nov 2002
174. Wolkotte PT, Smit GJM, Kavaldjiev N, Becker JE, Becker J (2005) Energy model of networks-on-chip and bus. In: Proceedings of the international symposium on system-on-chip, Nov 2005
175. F. Worm, P. Ienne, P. Thiran, G. D. Micheli, “A robust selfcalibrating transmission scheme for on-chip networks. IEEE Trans on Very Large Scale Integr Syst 12(12):1360–1373
176. Xie Y, Wolf W (2001) Allocation and scheduling of conditional task graph in hardware/software co-synthesis. In: Proceedings of design, automation and test in Europe conference, March 2001
177. Yan S, Lin B (2008) Design of application-specific 3D networks-on-chip architectures. In: Proceedings of ICCD, 2008
178. Yu Z, Baas B (2006) Implementing tile-based chip multiprocessors with GALS clocking styles. In: Proceedings of the international conference on computer design, Oct 2006
179. Zhao D, Wang Y (2008) SD-MAC: design and synthesis of a hardware-efficient collision-free QoS-aware MAC protocol for wireless Network-on-Chip. IEEE Trans Comput (TC) 8:1046–1057

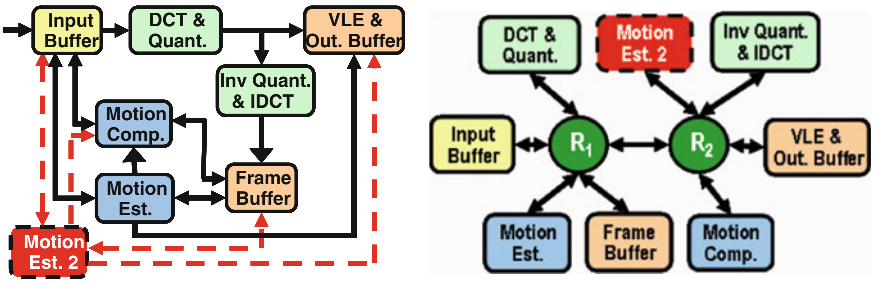
Chapter 3
Motivational Example: MPEG-2 Encoder Design

While NoCs gained recently a significant momentum, there are few NoC implementations of *real applications* reported to date [1, 5]. In this chapter, we present an MPEG-2 encoder using the NoC approach and compare it against the P2P and non-segmented bus-based designs running the same application. The MPEG-2 encoder has been selected as driver application since it covers a rich class of multimedia applications where similar considerations apply from an implementation standpoint. For instance, the basic JPEG, Motion-JPEG and MPEG-1 encoders, can all be implemented using a similar architecture and set of IP cores. We also note that, due to the small number of point-to-point connections in its architecture, the MPEG-2 encoder lends itself to a P2P implementation, as shown in Fig. 3.1. It should be noted that for many other applications where a subset of cores communicate with all the remaining nodes, the overhead incurred by the dedicated channels of the P2P architecture is significant. Similarly, the performance of bus architectures drops quickly as the number of communicating cores in the design increases. As a result, the conclusions derived herein with respect to the benefits of the NoC architecture compared to the P2P and bus architecture are rather conservative, so they shed light on a wide range of practical scenarios.

3.1 Overall Approach

After the computational resources that are needed for the implementation of the MPEG-2 encoder (e.g. discrete cosine transformation, motion estimation and variable length encoding modules) are implemented using Verilog HDL, they are connected to each other using P2P links, a bus, and an NoC architecture, as shown in Figs. 3.1 and 3.2. For the bus and NoC implementations, we also design a bus control unit (BCU) and an on-chip router, respectively. Once the designs are complete, they are evaluated in terms of area, performance, energy and power consumption. In addition to these standard metrics, we are also interested in analyzing the *scalability* of these communication architectures. For this reason, we increase the parallelism of the encoder by duplicating the motion estimation (*ME*)

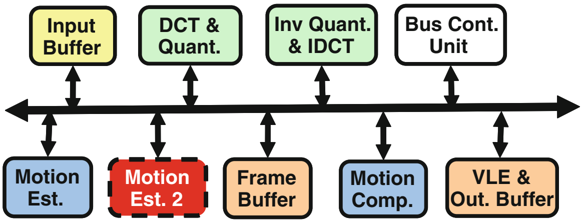
U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1_3,
© Springer Science+Business Media New York 2013



(a) (b)

Fig. 3.1 MPEG-2 encoder implementation using **a** point-to-point and **b** network-on-chip communication architectures

Fig. 3.2 MPEG-2 encoder implementation using bus communication architecture



module, which is the true performance bottleneck for this multimedia application. This way, we increase the number of cores in the design and perform evaluations with 1, 2, 4 and 8 motion estimation modules[1]; we refer to the design with one ME as the *baseline implementation*. Designs with 1 and 2 ME modules are implemented and evaluated using an FPGA prototype based on the Xilinx XC2V3000 platform [2]. However, the designs with 4 and 8 ME modules do not fit to this FPGA due to their large size. Therefore, the area, performance and energy consumption of these designs are found analytically by using the area, performance and energy consumption of individual components. Implementation details can be found in [2].

3.2 Evaluation of the NoC Architecture

3.2.1 Area Evaluation

The area of the P2P, bus and NoC-based implementations with 1, 2, 4, 8 ME modules is shown in Fig. 3.3a. As we can see, the P2P implementation is

[1] This corresponds to having MPEG-2 encoder implementations with a total of 7, 8, 10, and 14 cores, respectively.

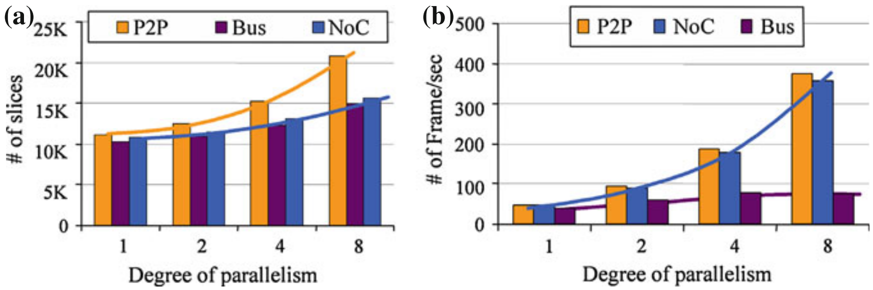


Fig. 3.3 **a** Area and **b** throughput comparisons of the MPEG-2 encoder implemented using P2P, bus and NoC architectures for increasing level of parallelism

consistently larger than the bus and NoC implementations. More importantly, the area of the P2P implementation scales considerably worse. For example, the P2P version is more than 24.7 % larger than the NoC version for the implementation with 8 ME modules, while the difference in performance is only about 4.4 %. This shows that P2P architectures scale poorly in terms of area, and they are not suitable for designs involving a large number of cores. On the other hand, the NoC-based implementations scale as well as the bus-based implementation with the increasing number of cores. For instance, the NoC version has about 4.6 % overhead compared to the bus version even for the 8 ME implementation.

3.2.2 *Performance Evaluation*

The throughput of the P2P, bus and NoC-based implementations as a function of number of ME modules in the design is plotted in Fig. 3.3b. The throughput of the P2P implementation is 47.0 Frames/sec for a CIF frame of size 352 × 288. The throughput of the corresponding bus and NoC implementations is 38.9 Frame/sec and 46.4 Frames/sec, respectively. We note that, the NoC implementation achieves a throughput very close to the P2P version which provides the utmost communication performance. The throughput of all implementations as a function of increasing number of ME modules is shown in Fig. 3.3b. We observe that the NoC implementation scales as well as the P2P implementation in terms of throughput, while the bus-based implementation scales poorly. However, one should note that beyond a certain degree of parallelism, the communication itself becomes the bottleneck and therefore the NoC performance saturates. It is possible to eventually stretch the performance beyond this point by customizing the network topology [3, 4].

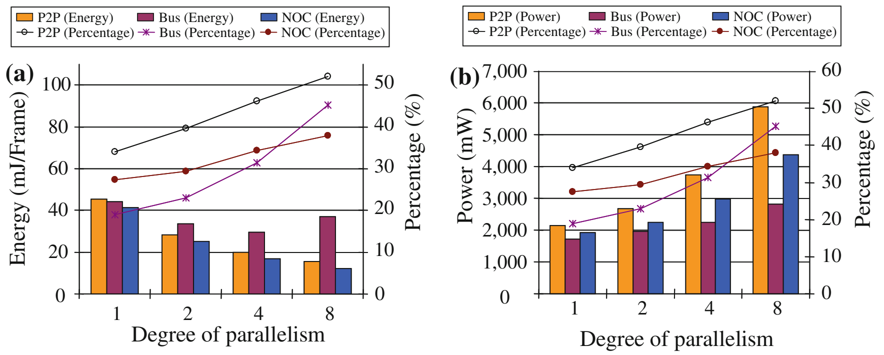


Fig. 3.4 **a** Energy and **b** power consumption (mW@ 100 MHz) as a function of the degree of parallelism for P2P, bus and NoC implementations

3.2.3 Energy Consumption Evaluation

Figure 3.4a shows that the energy consumption for the NoC implementation is consistently smaller than the P2P and bus counterparts for different levels of parallelism. The P2P implementation has larger energy consumption, since it has more interfaces and links than the NoC counterpart. On the other hand, the energy consumption of the bus-based implementation is large due to the longer time needed to encode real data (i.e. smaller throughput compared to the P2P and NoC-based implementations). The longer encoding time of the bus-based implementation also results in a smaller power consumption, as shown in Fig. 3.4b.

On the other hand, the power consumption of the P2P implementation is larger than the power consumed by the NoC even for the baseline implementation. Furthermore, we observe that the NoC design scales better in terms of power consumption compared to the P2P implementation, as shown in Fig. 3.4b. The power consumption of the P2P architecture scales poorly as the degree of parallelism increases, since the P2P implementation requires a significantly larger number of additional links and network interfaces to accommodate the addition of extra *ME* modules. More specifically, the NoC-based implementation consumes up to 42 % less power compared to the P2P implementation for an implementation involving 8 *ME* modules. This corresponds to about 17 % of the total power consumption which is quite important when optimizing portable systems.

3.3 Overall Comparison

Even for the baseline implementation, the NoC architecture performs as well as the P2P architecture, while having a smaller area overhead. At the same time, the real benefits of using the NoC approach are observed when we analyze the scalability of these designs as a function of the number of cores. More specifically, when the motion estimation module is replicated, the area occupied by the P2P implementation grows abruptly. The bus architecture, on the other hand, scales well in terms of area, but it suffers from a energy/performance standpoint. Unlike both of these approaches, the NoC implementation incurs only a modest area overhead, while keeping up with the performance increase achieved by the P2P implementation. Finally, the NoC-based implementation has also better scalability in terms of energy consumption compared to the P2P and bus-based implementations.

Based on analytical estimations and direct measurements on the FPGA prototype, we observe the following:

- The performance of the NoC-based implementation is very close to that of the P2P for the same application. Moreover, the scalability analysis based on duplicating the bottleneck module in the MPEG-2 design shows that the performance of the NoC design scales as well as the P2P, while the bus-based implementation scales much more poorly.
- In terms of area, the NoC scales as well as the bus-based implementation. However, the P2P implementation does not scale well due to the overhead involved in redesigning the interfaces. Moreover, the design effort for adding new cores to an existing design is much smaller for the NoC case as compared to P2P.
- Finally, the energy consumption of the NoC-based implementation is smaller than both P2P and bus-based implementations and it scales much better with the number of extra ME modules added to the base design.

In summary, the NoC design scales very well in terms of area, performance, power/energy consumption and overall design effort, while the P2P architecture scales poorly on all accounts except performance. By contrast, the bus-based architecture scales poorly in terms of performance and energy consumption.

References

1. Howard J et al (2010) A 48-Core IA-32 message-passing processor with DVFS in 45 nm CMOS. In: Proceedings of the IEEE international solid-state circuits conference, February 2010
2. Lee HG, Chang N, Ogras UY, Marculescu R (2007) On-chip communication architecture exploration: a quantitative evaluation of point-to-point, bus and network-on-chip approaches. ACM Trans Des Autom Electron Syst 12(3)
3. Ogras UY, Marculescu R (2005) Energy- and performance-driven NoC communication architecture synthesis using a decomposition approach. In: Proceedings of design automation and test in Europe conference, March 2005
4. Srinivasan K, Chatha KS, Konjevod G (2006) Linear programming based techniques for synthesis of network-on-chip architectures. IEEE Trans Very Large Scale Integr Syst 14(4):407–420
5. Vangal S et al (2007) An 80-Tile 1.28TFLOPS network-on-chip in 65 nm CMOS. In: Proceedings of solid-state circuits conference, February 2007

Chapter 4
Target NoC Platform

This chapter provides an overview of the network-on-chip architecture and application models utilized in this book. We first describe the target NoC platform and list our basic assumptions. Then, we present the NoC architecture and application models employed throughout the book. Finally, we conclude this chapter by discussing the technology implications on networks-on-chip design.

4.1 Basic Assumptions

The nodes in the network are composed of a processing core and a router, as shown in Fig. 4.1. The routers are connected to the neighboring routers and to the local processing/storage element (PE hereafter) via bi-directional point-to-point links. The on-chip routers are likely to be used with many existing IP cores which have been designed with certain protocols in mind (e.g., CoreConnect, AMBA). In order to proliferate the utilization of NoC architectures, wrappers which can efficiently interface the existing bus-based IPs and the NoC communication infrastructure are required. Therefore, communication interfaces (or wrappers) that interface the router with the processing cores and perform packetization/depacketization operations are utilized whenever necessary.

The routers are composed of input/output buffers, switching fabric and control logic, as depicted in Fig. 4.1. The buffers can be implemented using either registers or SRAMs for energy efficiency. The buffers typically hold a fraction of a packet. In a P-port router $P \times P$ crossbar switch serves as the switching fabric in the router. The control logic, which consists of routing engine, virtual channel allocation module and arbiter determines which output link the packet should be delivered to based on a routing algorithm. Different network topologies such as mesh, torus or customized can be realized by interconnecting the routers according to a given rule.

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1_4,
© Springer Science+Business Media New York 2013

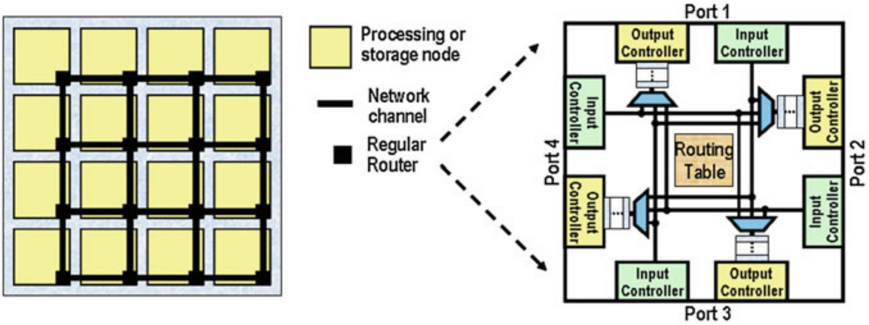


Fig. 4.1 Regular tile-based networks-on-chip architecture and a simple on-chip router with four ports

4.1.1 Routing Algorithm

The routing algorithm determines how the packets are routed in the network between the source-destination pairs. The objective of routing algorithms can be selecting the minimal routing paths, avoiding congestion (while producing minimal paths or dropping minimal path requirement), avoiding deadlock, maintaining uniform power consumption across routers, improving fault-tolerance, etc. In general, routing algorithms are classified as adaptive and oblivious algorithms [6, 10]. Adaptive routing algorithms use the state of the network such as congestion in the routers while making the routing decision. On the other hand, the routing decision is independent of the network state in oblivious routing. For this type of routing, the routing path can be completely determined by the source and destination addresses (deterministic routing [10, 19]) or it can be determined by probabilistic rule (probabilistic routing [3]) to improve fault-tolerance.

Implementation complexity and performance requirements are two major concerns in selecting the routing strategy. While adaptive routing provides better throughput and lower latency by allowing alternate paths based on the network congestion [13], oblivious routing requires less resources, which is critical for NoCs. However, for application-specific NoCs, the routing algorithm can be selected to match the application traffic pattern [14]. Finally, for ordered packet arrival, freedom from deadlock and livelock can be easily guaranteed by deterministic algorithms. Therefore, unless otherwise specified, we assume deterministic, minimal and deadlock-free routing throughout the book.

4.1.2 Switching Technique

The switching technique determines when the routing decisions are made, how the switches inside the routers are set/reset, and how the packets are transferred along the switches. Switching techniques are usually categorized as packet switching (store-and-forward, virtual cut-through and wormhole switching) and circuit switching [10].

In store-and-forward switching, which is the dominant choice in internet, the entire packet is stored at the intermediate nodes. The packet is forwarded to one of the neighboring nodes, when the output channel is available and the chosen neighboring node has enough empty buffering space available to hold the entire packet. Storing the entire packet requires large buffering space and causes large packet latency. Virtual cut-through switching solves the latency problem in store-and-forward switching by forwarding the packet as soon as the desired output channel becomes available. Hence, a packet is stored at an intermediate node only if the outgoing channel is busy. However, virtual cut-through switching still requires large buffering space at each node, since the entire packet needs to be stored in case of congestion. For wormhole switching, a packet is divided into flow control digits called flits. The first flit (called the header flit) contains all the routing information and leads the packet through the network, while the remaining flits follow the header flit in a pipelined fashion. When the header flit is blocked due to congestion in the network, all of the following flits wait at their current locations. Therefore, the need for large buffers at each router is eliminated. At the same time, wormhole routing achieves small routing latency similar to virtual cut-through routing. The major drawback of wormhole routing is the chained blocking which occurs as a result of the multiple occupied channels due to the blocked header flits. However, this problem can be alleviated using virtual channels [8].

In circuit switching, a physical circuit is set-up between the source and destination nodes during the circuit establishment phase unlike the packet switching techniques. Then, all the packets that belong to that stream are transmitted along this circuit without any arbitration overhead and with minimal buffering.

Due to the limited buffering resources available and the stringent latency requirements for typical NoC applications, we assume wormhole-based routing which is the commonly preferred technique in NoCs [2, 4, 7, 14, 18]. Wormhole routing is usually preferred to circuit switching in data networks due to the poor performance of the latter under dynamic traffic. However, for application-specific NoCs, this does *not* represent a major handicap. Moreover, guaranteed service operation, as required by some applications, is relatively easier to satisfy by using circuit switching [9] as opposed to wormhole routing [2, 4]. Therefore, circuit switching is a promising alternative, despite its implementation complexity and static nature. It remains to be seen whether or not a particular switching technique, or a hybrid combination [16], is more advantageous.

4.2 NoC Architecture Modeling

In this section, we formally define a 3D design space that involves issues related to communication infrastructure synthesis, communication paradigm selection and application mapping optimization. Having such a unifying formalism available can not only catalyze the research towards improving the already existing solutions, but also inspire new solutions to many outstanding research problems in NoC design.

Definition An NoC architecture can be uniquely described by the triple $Arch(T(R, Ch),\ P_R, \Omega(C))$, where:

- The labeled graph $T(R, Ch)$ represents the network *topology*. The routers and channels in the network are given by the sets R and Ch, respectively, as follows:
- $\forall(ch) \in Ch,\ w(Ch)$ gives the bandwidth of channel ch;
- $\forall r \in R,\ l(d, r)$ gives the buffer size (depth) of channel d, located at router r;
- $\forall r \in R,\ Pos(r)$ gives the xy coordinates of router r in the chip floorplan;
- $\{P_R(r, i, j) | i, j, r \in R\}$ defines the routing policy P_R at router r, for any source router i, destination router j, while considering a particular switching technique;
- $\Omega : C \to R$ is a function that maps each vertex $c_i \in C$ in the APCG to a router in R. For *direct* topologies, Ω is a bijective function, i.e., every router is connected to a core, while in *indirect* topologies a router may be connected only to other routers.

We can conceive the choices of designing NoCs as representing a 3D design space, where each component of the triple $Arch(T(R, Ch),\ P_R,\ \Omega(C))$ defines a separate dimension of the design space.[1] Using this type of concise representation has the advantage of keeping the discussion simple and precise.

Finally, besides the obvious functional constraints like correctness, freedom from deadlock, etc., there are a number of *performance* and *cost* metrics that the design techniques developed for NoCs should comply with. We list below a non-exhaustive list of such metrics:

$$\begin{aligned} Performance\ Metrics = \{&average/maximum\ packet\ latency,\ bisection \\ &bandwidth,\ network\ throughput,\ QoS\} \end{aligned} \tag{4.1}$$

$$\begin{aligned} Cost\ Metrics = \{&average/peak\ energy/power\ consumption,\ network \\ &area\ overhead,\ total\ area,\ average/peak\ temperature\} \end{aligned} \tag{4.2}$$

This formalism helps to identify and formulate key research problems in NoC design in a standard way, as illustrated in the subsequent sections. For instance, topology synthesis, buffer size allocation, channel width sizing, and floorplanning problems need to be solved within the first dimension, i.e. communication infrastructure design. A more detailed description of the key research problems along each of these dimensions can be found in [17].

[1] We note that similar definitions have been used by the EDA research community [11, 14, 18, 22].

4.3 Application Modeling

In this section, we present a unified application model utilized throughout the book. At different stages in the design process (e.g. whether or not the application has been mapped, scheduled, etc.), the target application can be specified at different levels of granularity. In this book, we describe the target application either by a *communication task graph* (CTG) [12] or *application characterization graph* (APCG) [14]. The nodes of a CTG are the tasks that define a particular application, while its edges specify the control and data dependencies. A CTG is used to model an application before the tasks are bound to a particular IP core that will run the task. When the tasks are already scheduled to run on certain IP cores, then we utilize APCG to model the target application. Consequently, an CTG models the target application at a finer level of granularity. Formal definitions of CTG and APCG are given below.

Definition A *communication task graph* (CTG) $G' = G'(S_T, C)$ is a directed acyclic graph, where each vertex represents a computational module of the application referred to as a task $t_i \in S_T$. Each t_i is annotated with relevant information such as the execution time on each type of PE, energy consumption (e_j^i) when executed on the jth PE, task deadlines $(d(t_i))$, etc. Each directed arc $c_{i,j} \in C$ characterizes the communication (or control) dependency between tasks t_i and t_j. Each $C_{i,j}$ has associated $v(C_{i,j})$, which stands for the communication volume (*bits*) exchanged between cores c_i and c_j.

Definition An application characterization graph (APCG) $G = G(C, A)$ is a *directed* graph, where each vertex $c_i \in C$ represents an IP core, and each directed arc $a_{i,j} \in A$ characterizes the communication from vertex c_i to vertex c_j. Each $a_{i,j}$ can be tagged with application-specific information (e.g., communication volume $v(a_{i,j})$ between vertices c_i and c_j), and specific design constraints (e.g., communication bandwidth $b(a_{i,j})$ and latency requirements of the application, etc.).

4.4 Technology Implications on Networks-on-Chip Platforms

Technology has an important impact on area, power consumption and performance of the NoCs. For instance, in [21], the authors investigate the impact of the technology on the choice of different design parameters such the network topology, number of virtual channels and buffer sizes. For this reason, this section addresses the implications of the particular technology used to implement on the NoC design. To this end, we will use the 80-core teraflops (tera floating point

Table 4.1 Technological predictions for high performance Microprocessor (MPU) and ASIC product generations by ITRS report 2005 edition [20]

Technology node	90 nm (2005)	65 nm (2007)	50 nm (2009)	36 nm (2012)	25 nm (2015)
MPU chip size (mm^2)	310	310	310	310	310
ASIC chip size (mm^2)	858	858	858	858	858
MPU / ASIC Mtransistor/cm^2	225	357	566	1133	2265
Max. on-chip frequency (MHz)	5,204	9,285	12,369	20,065	33,403
Power supply voltage V_{dd}(V)	1.1	1.1	1.0	0.9	0.8
Allowable max. power[a] (W)	167	189	198	198	198
Max. power for hand-held systems (W)	2.8	3	3	3	3
Number of metal layers-minimum[b]	11	11	12	12	13
Number of metal layers-maximum	15	15	16	16	17
Metal 1 wiring pitch (nm)	180	136	104	72	50
Metal 1 RC delay (ps) (for 1 mm minimum pitch Cu wire)	440	767	1388	2857	5951
Minimum global wire pitch (nm)	300	210	156	108	75
Global wire RC delay (ps) (for 1 mm minimum pitch Cu wire)	111	209	410	787	896

[a] According to ITRS, power will be limited by system level cooling and test constraints rather than packaging

[b] The minimum number of wiring levels represents the interconnect metal levels, and the maximum number of interconnect wiring levels includes the minimum number of wiring levels plus additional optional levels required for power, ground, signal conditioning, and integrated passives (i.e., capacitors)

operations per second) research chip introduced by Intel as a basis to predict how NoCs will look like in the future technology nodes [23]. There are two reasons for this choice. First, this chip is one of the largest NoC prototypes to date. Second, its specifications which are publicly available can be used together with the ITRS roadmap [20] to study technological implications on NoC research.

Intel's 80-core chip is implemented using 65 nm technology, and it is arranged as a 10 × 8 2D mesh network consisting of 3 mm^2 tiles, as shown in Fig. 4.1. The chip has a total die area of 275 mm^2 which is close to the maximum chip size (310 mm^2) predicted by ITRS through 2015, as shown in the second row of Table 4.1. The chip has 100M transistors; this is smaller than the current state-of-the art processors (e.g. Intel Core 2Duo processor contains 291M transistors [15]) and ITRS projections, which are shown in the fourth row of Table 4.1. When the chip is powered at 1.2 V supply voltage, the maximum frequency is 4 GHz. This results in 256 GB/s bisection bandwidth[2], 1.28 TFLOPS peak performance and 181 W power consumption. As it can be seen from the ITRS predictions (see Table 4.1, row 6), this power consumption is almost the maximum allowed value.

[2] Bisection bandwidth is defined as the total bandwidth across smallest cut that divides network into two equal halves [6].

When the supply voltage is decreased to 1.0 V, the power consumption reduces to 98 W. At the same time, the maximum clock frequency and total performance drop to 3.13 GHz and 1.0 TFLOPS, respectively. Finally, a 0.6 V supply voltage results in 11 W power consumption with a total performance of 310 GFLOPS.

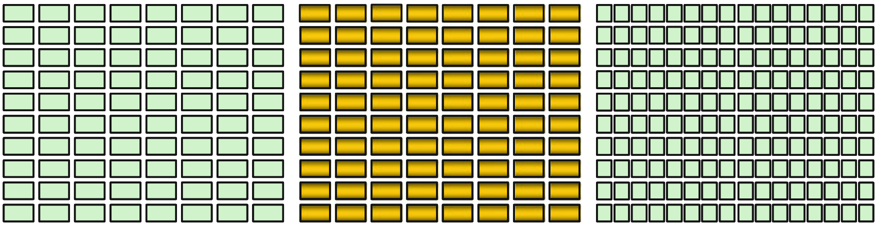
Having reviewed the specifications of the 80-core intel chip, we will project this design into future technology nodes using the ITRS roadmap predictions summarized in Table 4.1. According to the ITRS roadmap the chip size remains constant, while the feature sizes shrink. So, we will assume that 275 mm^2 chip size does not change. Based on this assumption, we consider the following two scenarios:

Scenario 1: First, we assume that the tile sizes remain the same over different technology nodes, i.e. the tiles in the network remain 3 mm^2. This choice still implies 80-core networks, but with more complex tiles. More specifically, at 50 nm technology node, the tiles could have 158M transistors (as opposed to 100M transistors). Under this scenario, the tiles become more complex as we move to new technology nodes, while the network size and the lengths of the network links remain the same. Combined with the scaling in the width and pitch of the global wires (see last two rows of Table 4.1), this will result in larger network bandwidth, hence performance, for each new technology. At the same time, due to the increase in the tile complexity and increased operating frequency, the power consumption will go up. According to ITRS roadmap, it will not be sufficient to reduce the supply voltage to meet the power constraints as the technology changes. On the contrary, system level approaches that are coupled with the low level techniques (similar to our *voltage-frequency island synthesis* approach presented in Chap. 8) are vital for the successful realization of future NoCs (Fig. 4.2).

Scenario 2: In the second scenario, we assume that the tile complexity in terms of number of transistors remains the same. This will be the case when the IP cores in the tiles are reused in the new design. Under this scenario, the number of tiles in the network will grow, since the total area remains the same and each tile can fit into a smaller area. More specifically, there would be 126 tiles at 50 nm, 254 nodes at 36 nm, and so on[3], as illustrated in Fig. 4.1. Under this scenario, the lengths of the network links scale down as the tile dimensions. This has two consequences with opposite effects on network performance. On one hand, single hop communication between neighboring nodes will be faster; on the other hand average hop distance in the network will increase. As a result, localized communication becomes faster, while communication across different parts of the network slows down. At this point, we note that *long-range links* that can connect remotely situated nodes (see Chap. 6) becomes a powerful tool to improve network performance.[4] We also note that the overhead of the network remains about the same, since the reduction in the router area is compensated by the increased

[3] We consider two corner cases in this section. In reality, we expect that both average tile complexity and network size will increase.

[4] We discuss the use and implementation of long-range links in Chap. 6



The original 10×8 network. Each tile is 3 mm^2 and the total area is 275 mm^2.

First scenario: The design is mapped to a smaller technology node. Individual tiles and the network have the same area, but each tile is more complex.

Second scenario: Individual tiles have the same complexity as the original design. Due to shrinking device sizes, the resulting network has more tiles.

Fig. 4.2 Illustration of the mapping the 80-core design to smaller technology nodes

number of routers. Finally, power consumption will go up as the technology scales down, similar to the previous scenario.

To sum up, the NoC design becomes more challenging with the introduction of each new technology node. Besides the complexity of dealing with a large scale network, the interplay between system level decisions such as topology selection and physical implementation parameters makes NoC design a challenging task. Indeed, the *Design Chapter* of the ITRS roadmap lists “design methodology” as one of the key technology challenges [20]. According to this report, each technology generation requires designers to consider more issues; hence, new analysis methods and tools must be developed to evaluate new phenomena and aid the designer in making critical design decisions. Our *NoC performance analysis* technique presented in Chap. 5 addresses exactly this need. Through real design case studies and comparison with cycle-accurate simulations, we demonstrate that this technique can indeed provide guidance in the huge design space. ITRS report further states that it becomes increasingly difficult to determine the most effective sequence in which design decisions made such that the iterations are minimized. To this end, we believe that the design methodologies and NoC analysis tools we present in the subsequent chapters will foster the design and implementation of large-scale NoCs. Finally, the advent of FPGAs with as many transistors as state-of-the-art processors and multi-FPGA development platforms, such as the BEE2 platform [5], will play an important role in multi processor research [1]. Therefore, the FPGA prototypes presented in this book establish an important step towards systematic evaluation of NoCs through prototyping.

References

1. Arvind AK et al (2005) RAMP: Research accelerator for multiple processors—A community vision for a shared experimental parallel HW/SW platform, RAMP Technical report. http://ramp.eecs.berkeley.edu/Publications/ramp-nsf2005.pdf

2. Bjerregaard T, Sparso J (2005) A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In: Proceedings of design, automation and test in Europe conference, March 2005
3. Bogdan P, Dumitras T, Marculescu R (2007) Stochastic communication: A new paradigm for fault-tolerant networks-on-chip. Hindawi VLSI design, special issue on networks-on-chip, vol 2007, Hindawi Publishing Corporation
4. Bogdan P, Marculescu R (2010) Workload characterization and its impact on multicore platform design. In: Proceedings of the 8th IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis (CODES/ISSS), 2010
5. Chang C, Wawrzynek J, Brodersen RW (2005) BEE2: A high-end reconfigurable computing system. IEEE Des Test Comput 22(2):114–125
6. Dally WJ, Towles B (2004) Principles and practices of interconnection networks. Morgan Kaufmann Press, San Francisco
7. Dally WJ, Towles B (2001) Route packets, not wires: On-chip interconnection networks. In: Proceedings of design automation conference, June 2001
8. Dally WJ (1992) Virtual-channel flow control. IEEE Trans Parallel Distrib Syst 3(2):194–205
9. Dielissen J, Radulescu A, Goossens K, Rijpkema E (2003) Concepts and implementation of the Philips network-on-chip. IP-based SoC Design, 2003
10. Duato J, Yalamanchili S, Ni L (2002) Interconnection networks: an engineering approach. Morgan Kaufmann, San Mateo, CA
11. Hansson A, Goossens K, Radulescu A (2007) A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic. Hindawi VLSI Design, Hindawi Publishing Corporation
12. Hu J, Marculescu R (2005) Communication and task scheduling of application-specific networks-on-chip. IEE Proc Comput Digital Tech 152(5):643–651
13. Hu J, Marculescu R (2004) DyAD—Smart routing for networks-on-chip. In: Proceedings of design automation conference, June 2004
14. Hu J, Marculescu R (April 2005) Energy- and performance-aware mapping for regular NoC architectures. IEEE Trans Comput Aided Des Integr Circuits Syst 24(4):551–562
15. Intel architecture, http://www.intel.com/pressroom/kits/core2duo/pdf/microprocessor_timeline.pdf
16. Lee K et al (2004) A 51mW 1.6GHz on-chip network for low-power heterogeneous SoC platform. International solid-state circuits conference, Feb 2004
17. Marculescu R, Ogras UY, Peh L, Jerger NE, Hoskote Y (Jan. 2009) Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. IEEE Trans Comput Aided Des Integr Circuits Syst 28(1):3–21
18. Murali S, De Micheli G (2004) Bandwidth-constrained mapping of cores onto NoC architectures. In: Proceedings of design, automation and test in Europe conference, Feb 2004
19. Ni LM, McKinley PK (Feb. 1993) A survey of wormhole routing techniques in direct networks. IEEE Comput 26(2):62–76
20. Semiconductor Association (2006) The international technology roadmap for semiconductors (ITRS)
21. Soteriou V, Eisley N, Wang H, Li B, Peh L (2006) Polaris: a system-level roadmap for on-chip interconnection networks. In: Proceedings of international conference on computer design, Oct 2006
22. Srinivasan K, Chatha KS, Konjevod G (April 2006) Linear programming based techniques for synthesis of network-on-chip architectures. IEEE Trans Very Large Scale Integr VLSI Syst 14(4):407–420
23. Vangal S et al (2007) An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In: Proceedings of solid-state circuits conference, Feb 2007

Chapter 5
NoC Performance Analysis

Traditionally, performance evaluation of networks-on-chip (NoC) is largely based on simulation which, besides being extremely slow, provides little insight on how different design parameters can affect the actual network performance. Therefore, it is practically impossible to use simulation for optimization purposes. This chapter presents a mathematical model for on-chip routers and utilize this new model for NoC performance analysis. The model presented in this chapter can be used not only to obtain fast and accurate performance estimates, but also to guide the NoC design process within an optimization loop. The accuracy of our approach and its practical use is illustrated through extensive simulation results.

5.1 Introduction

NoC communication architectures offer a scalable and modular solution for implementing complex systems which consist of a large number of heterogeneous components. The complexity of such systems, as well as the tight requirements in terms of power, performance, cost and time-to-market, place a tremendous pressure on the design team. To cope with this situation, application and platform models are usually developed separately [10]. After that, the application is mapped to the target platform and the resulting system is evaluated to ensure its compliance with the design specifications, as depicted in Fig. 5.1.

The application model comes with some workload characterization (usually given in probabilistic terms), while the platform itself may come with some low-level information from the designers, depending on the targeted level of accuracy for this type of evaluation. These models are used during the mapping step when the target application is mapped onto the target architecture. Next, a performance analysis step is needed to determine whether or not the chosen application-architecture combination satisfies the imposed design constraints. Finally, the information provided during the performance analysis step is used to refine the communication architecture and decide the communication topology (e.g., bus, point-to-point (P2P)), buffer space, etc.

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1_5,
© Springer Science+Business Media New York 2013

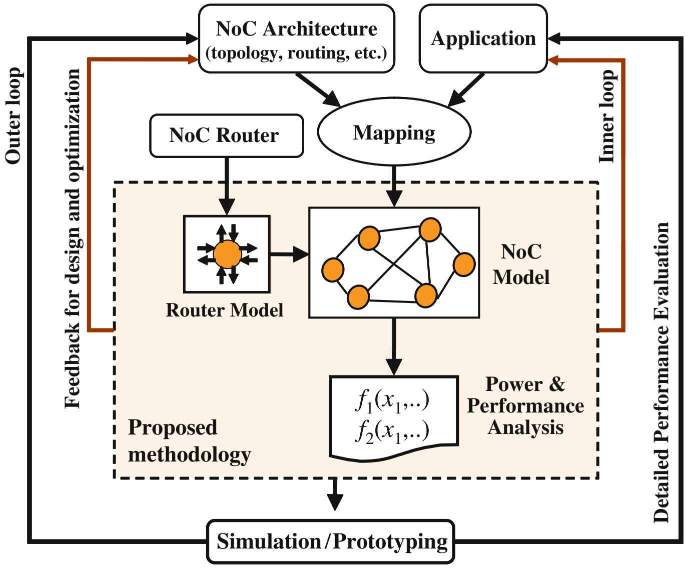


Fig. 5.1 The use of proposed performance analysis approach is illustrated with the Y-chart scheme [1]

The success of this methodology depends critically on the availability of adequate power and/or performance analysis tools that can guide the overall design process. In order to be used in an optimization loop, such as the one shown in Fig. 5.1, the analysis needs to be tractable, while still providing meaningful feedback to the designer. Time consuming simulations can only come into the picture at later stages, typically *after* the design space is already reduced to fewer practical choices (the outer loop in Fig. 5.1).

We note that for traffic with guaranteed service [3, 12], accurate performance figures can be easily derived. However, performance analysis for best effort traffic relies largely on simulation or performance metrics derived under idealized conditions. For example, the average hop count is commonly used to approximate the average packet latency [13]. While this metric ignores the queuing delays and network contention, approaches that do consider queuing delays often make other idealistic assumptions such as exponential service times, infinite buffers, etc. [4, 5, 9]. Likewise, analysis of power consumption of the NoC architectures depends largely on simulation [17, 21].

Starting from these overarching considerations, this chapter presents a formal approach for NoC average-case performance analysis. At the very heart of the proposed methodology lies a new router model which is based on a set of FIFO buffers interconnected by a switch as shown in Fig. 5.2. The newly proposed router model enables us to derive a closed-form expression for *the average number of packets* at each input buffer in the router under Poisson traffic arrival assumption

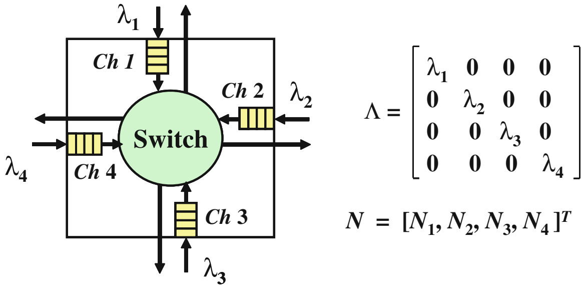


Fig. 5.2 The router model as a collection of FIFO buffers is illustrated for a router with four channels. The arrival rates to the router are described by the diagonal matrix Λ, and the average number of packets at each input channel is described by N

for the *header flits*.[1] The proposed network performance approach provides three performance metrics:

- Average buffer utilization of each buffer,
- Average packet latency per flow,
- Maximum network throughput.

Consequently, the proposed analytical framework can be easily used for design and optimization purposes, as well as obtaining quick performance estimates. At the same time, using the newly proposed performance model, we are able to formally prove that dedicated P2P links result in better performance than router-based communication, while router-based communication results in better performance figures than shared buses under identical input traffic and service time distributions. Finally, we utilize the proposed model to analyze the performance of a basic NoC, given its topology, routing algorithm, as well as the driver application and its mapping to the network nodes.

The remaining of this chapter is organized as follows. Section 5.2 reviews related work and highlights novelty of the presented approach. Section 5.3 presents the router model used for performance analysis. Sections 5.4 and 5.5 present techniques for router and network performance analysis, respectively. Experimental results are provided in Sect. 5.6. Finally, Sect. 5.7 concludes the chapter.

5.2 Related Work

The design of application-specific NoCs is commonly formulated as a constrained optimization problem [8, 13, 14]. Therefore, performance analysis techniques that can be used in optimization loops are extremely important. The authors in [9] consider the buffer sizing problem and present a performance model based on queuing theory. However, the approach is applicable to only packet switched networks. The work in [6] presents a wormhole delay model applicable to routers with single flit buffers and assume that packet size dominates the overall latency.

[1] Our assumptions and router model are detailed in Sect. 5.5.

Related work about analysis techniques for wormhole routing comes mainly from parallel computing and classical networks research communities. Many studies target specific network topologies such as *k*-ary *n*-cubes [2, 4] and hypercubes [16]. The study presented in [5] is not restricted to a particular topology, but it assumes an exponential message length distribution and it has a very high complexity for high dimensional networks. A more general analytical model for wormhole routing is presented in [7]. The model provides average packet latency estimates using a sophisticated analysis.

This chapter presents a novel performance model for on-chip routers which generalizes the traditional delay models for single queues and consequently captures the classical results as a special case. This new model is used to develop a thorough performance analysis for wormhole routing with *arbitrary size messages* and *finite buffers* under application-specific traffic patterns. This model has several unique features. More precisely, (i) it directly targets NoCs under application-specific traffic patterns, (ii) it supports arbitrary network topologies and deterministic routing and, finally, (iii) it provides not only aggregate performance metrics, such as average latency and throughput, but also feedback about the network behavior at finer granularity (e.g., utilization of all buffers, the average latency experienced at each router, average latency per flow). Given the generality of this newly proposed model, it can be invoked in any loop for NoC optimization (e.g., application mapping [8, 13], network architecture synthesis [14], buffer space allocation [9]) for fast and accurate performance estimations.

5.3 Router Modeling for Performance Analysis

5.3.1 Basic Assumptions and Notations

We consider input buffered routers with P channels and target wormhole flow control under deterministic routing algorithms. The *size* of the packets (*bits*) is denoted by the random variable $\boldsymbol{S}$, as listed in Table 5.1 along with other parameters. The probability distribution of $\boldsymbol{S}$ is given by the driver application.

The network channel *bandwidth* is denoted by W (*bits/sec*). The router *service time* for the *header flit* is given by H_s. We note that H_s is a function of the router design and includes the time to traverse the router (t_R) and the link (t_L). Since the remaining flits follow the header flit in a pipelined fashion, the service time of a packet, excluding the queuing delay (this will be accounted for in Sect. 5.5), is given by:

$$\boldsymbol{T} = H_s + \left\lceil \frac{\boldsymbol{S}}{W} \right\rceil \tag{5.1}$$

We denote by x_{sd} (*packets/sec*) the rate of the traffic transmitted from the source node at router s to the destination node at router d. Likewise, the traffic arrival rate of the header flits to input channel j of router i is given by λ_{ij} (*packets/sec*). We

Table 5.1 List of input parameters used in this chapter

Input	Explanation	Depends on
$\boldsymbol{S}$	Random variable (rv) denoting the packet size	Application
x_{sd}	Packet transmission rate from node s to node d	
R	Residual packet waiting time	
H_s	Service time for the header flit	Router
W	Network channel bandwidth	
B_{ij}	Size of the input buffer at router i, channel j	
$\boldsymbol{T}$, T, T^2	rv $\boldsymbol{T}$ denotes the packet service time. T and T^2 are its 1st and 2nd order moments	Application and Router
c_{ij}, C	Contention probability between channels i and j	
λ_j	Mean arrival rate at input buffer of channel j	Topology, routing, application
λ_{ij}	Traffic arrival rate at router i, channel j	

Bold symbols (e.g., $\boldsymbol{S}$ and $\boldsymbol{T}$) denote random variables

assume that the arrival process of the *header flits* to the router inputs (λ_{ij}) follows a Poisson process. Note that under this model, the arrival process for the body flits is *not* assumed to be Poisson; the Poisson assumption refers only to the header flit distribution. This assumption, which is actually quite common in network performance analysis [4, 6, 7], enables us to derive closed loop solutions and show that our model generalizes the classical results for single queue systems.

We note that our evaluations on the actual distribution of the header interarrival times based on the Pearson's chi-square test [19] show that the Poisson assumption holds pretty well at low and medium traffic rates, and it deviates from the Poisson assumption at high traffic loads. Hence, the accuracy of the average packet latency predicted by the proposed analytical model is expected to degrade only when the network gets closer to its saturation point. However, we note that the degradation in the accuracy of average packet latency is acceptable as long as the saturation point can be predicted accurately. So while, in general, the real arrival process may exhibit a more deterministic or long-range dependent behavior [20], our model provides valuable insights into the router design process and reasonably accurate results for pruning the design space at early design stages (see Sect. 5.6). Relaxing the Poisson assumption remains an open problem [11, 15].

5.3.2 *Analytical Model of the Router*

This section focuses on modeling a single router as a set of first-come first-serve (FCFS) buffers connected by a crossbar switch, as shown in Fig. 5.2. The parameter of interest is the *average number of packets at the input buffers*, at *each* *input channel* 1,...,P, i.e., $N = [N_1, N_2, \ldots N_P]^T$. Since Poisson arrivals see time averages (PASTA),[2] the following equilibrium equation is valid for the input buffer at any channel j:

$$\lambda_j = \frac{N_j}{\tau_j} \tag{5.2}$$

where τ_j denotes the *average time* an incoming packet spends in queue j. τ_j is composed of the following components:

I Service time of the packets already waiting in the same buffer;

II The packets waiting in the other buffers of the same router and served before the incoming packet;

III The residual service time seen by an incoming packet (R).

Therefore, τ_j can be written as:

$$\tau_j = \overset{I}{TN_j} + T \overset{II}{\sum_{k=1,\,k\neq j}^{P}} c_{jk}N_k + \overset{III}{R} \tag{5.3}$$

where the coefficients c_{jk} denote the contention probabilities, i.e., the probability that channels j and k compete for the same output. The second component of the average waiting time (i.e. term *II* in Eq. 5.3) applies only to those packets that will be served *before* the incoming packet. The second component of the average waiting time (i.e., term *II* in Eq. 5.3) applies only to those packets that will be served before the incoming packet. More precisely, the fraction of packets that will be served before the incoming packets is captured by the coefficients c_{ij}'s which represent the contention probabilities among the packets arriving at ports i and j. This makes our approach an average-case rather than a worst-case type of analysis. The computation of these contention probabilities is illustrated in Sect. 5.3.3.

Depending on the output channel requested by the incoming packet and the router scheduling policy (e.g. priority, round robin, etc.), an incoming packet can be served earlier than a packet that is already waiting in one of the other buffers. In the following, we assume the Round Robin policy, but the results can be extended for other scheduling disciplines. Round Robin arbitration ensures that the router bandwidth is utilized equally by all the input ports, as detailed in Sect. 5.3.3 when we illustrate the computation of the contention probabilities. For example, if a priority-based arbitration is used, then the average packet latency needs to be different for each input port. More precisely, the port with the highest priority would have the smallest latency, basically the router service time plus the residual

[2] The PASTA definition implies that the distribution of packets in the system that is seen by a new (arriving) packet is the same as the long run (time asymptotic) or steady state of the packet distribution. This allows us to relate the mean waiting time a packet in the buffer with the mean number of packets in all the buffers of the router.

time. Similarly, the latency of the packets arriving at all other ports would need to only consider the ports with higher priorities.

Let C_j be the row vector $C_j = [c_{j1}, c_{j2}, \ldots, c_{jP}]$ of the contention probabilities, where $c_{jj} = 1$. Then, Eq. 5.2 can be written using τ_j from Eq. 5.3 as:

$$\lambda_j = \frac{N_j}{TC_jN + R}, \text{ since } C_jN = N_j + \sum_{k=1,\, k \neq j}^{P} c_{jk}N_k$$

so re-arranging the terms yields:

$$\lambda_j TC_jN + \lambda_j R = N_j \tag{5.4}$$

Equation 5.4 describes the *equilibrium condition* of the buffer at the input channel j only. For the entire router, we denote the arrival rates (Λ), the contention matrix (C) and the residual time $(\overline{R})$ by:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \lambda_P \end{bmatrix}_{P\times P}, \quad C = \begin{bmatrix} C_1 \\ C_2 \\ \cdots \\ C_P \end{bmatrix}_{P\times P}, \quad \overline{R} = R \begin{bmatrix} 1 \\ 1 \\ \cdots \\ 1 \end{bmatrix}_{P\times 1}$$

Then, the equilibrium condition for the router can be written as:

$$T\Lambda CN + \Lambda\overline{R} = N$$
$$(I - T\Lambda C)N = \Lambda\overline{R}$$

Finally,

$$N = (I - T\Lambda C)^{-1}\Lambda\overline{R} \tag{5.5}$$

The router model described by Eq. 5.5 provides a closed form expression for the average number of packets at *each* buffer of the router, given the traffic arrival rates (Λ), the packet contention probabilities (C), router design specifications (H_s, W) and packet size distribution (S). Equation 5.5 generalizes the single queue model; this is one of the major contributions of this work.

We note that when $det(I - T\Lambda C) = 0$, the packet population in the router grows to infinity. This corresponds to the case when the utilization is 1 for a system with a single queue. The following example gives more intuition for Eq. 5.5.

Example 1 Consider the case $P = 1$ (i.e., single queue system) and infinite buffers. In this case, Eq. 5.5 simply becomes:

$$N = \frac{\lambda R}{1 - T\lambda}$$

Furthermore, the residual waiting time $R = 1/2\lambda\overline{T^2}$ where $\overline{T^2}$ is the second moment of the service time [20]. As a result:

$$N = \frac{\lambda^2\overline{T^2}}{2(1 - T\lambda)}, \tag{5.6}$$

which is precisely the average number of packets in an *M/G/*1 system. Hence, the commonly studied distributions *M/G/*1, *M/M/*1 and *M/D/*1 become *special cases* of our newly proposed model.

5.3.3 Computation of the Contention Matrix

Let f_{ij} be the probability that a packet arrives at channel i and leaves the router through channel j. The *forwarding probability matrix* is:

$$F = \begin{bmatrix} 0 & f_{12} & f_{13} & \cdots & f_{1P} \\ f_{21} & 0 & f_{23} & \cdots & f_{2P} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ f_{P1} & f_{P2} & f_{P3} & \cdots & 0 \end{bmatrix}, \quad \text{where} \quad f_{ij} = \frac{\lambda_{ij}}{\sum\limits_{k=1}^{P} \lambda_{ik}}, \quad 0 \le i, j \le P \tag{5.7}$$

where λ_{ij} is the traffic arrival rate at input channel i which is routed towards the output channel j. Assuming that the forwarding probabilities are independent for a deterministic routing algorithm, the contention probabilities can be written in terms of the forwarding probabilities as:

$$0 \le i, j \le P, i \ne j, \quad c_{ij} = \sum_{k=1}^{P} f_{ik} f_{jk}; i = j, \quad c_{ii} = 1 \tag{5.8}$$

Example 2 Consider a router with three ports. The packets arriving at port 1 are directed either to port 2 or port 3 with probability 0.5, i.e.,

$$f_{12} = 0.5, f_{13} = 0.5$$

Similarly, suppose that the packets arriving at port 2 are forwarded with probability 0.4 to port 1 and with probability 0.6 to port 3. Finally, assume that packets arriving at port 3 are always routed to port 1. Hence, the forwarding matrix for this simple scenario becomes:

$$F = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.4 & 0 & 0.6 \\ 1 & 0 & 0 \end{bmatrix}$$

The diagonal entries c_{11}, c_{22}, c_{33} are 1 by definition. Other entries are found using Eq. 5.8, as follows:

$$c_{12} = \sum_{k=1}^{3} f_{1k} f_{2k} = 0.3, \quad c_{13} = \sum_{k=1}^{3} f_{1k} f_{3k} = 0, \quad \text{and} \quad c_{23} = \sum_{k=1}^{3} f_{2k} f_{3k} = 0.4$$

In particular, we note that $c_{13} = 0$, i.e., there is no contention, since all packets arriving at port 3 are routed to port 1, while no packet arriving at input port 1 is directed back to output port 1.

Example 3 Suppose that for a particular router with three ports, the packet arrival rates are given as follows:

$\lambda_{12} = 0.01$ *packet/cycle* (i.e. the rate of the traffic that enter the router at channel and leaves at channel 2 is 0.01 *packet/cycle*),

$\lambda_{13} = 0.01$ *packet/cycle*,

$\lambda_{21} = 0.02$ *packet/cycle*, $\lambda_{23} = 0.01$ *packet/cycle*,

$\lambda_{31} = 0.01$ *packet/cycle*, $\lambda_{32} = 0.0$ *packet/cycle*.

Then, the probability that a packet arrives at channel 1 and leaves the router through channel 2 is found using Eq. 5.7 as:

$$f_{12} = \frac{\lambda_{12}}{\lambda_{12} + \lambda_{13}} = 0.5$$

That is, given that a packet arrives at channel 1, it will leave the router from channel 2 with 0.5 probability. Similarly, other probabilities are computed as follows:

$$f_{13} = \frac{\lambda_{13}}{\lambda_{12} + \lambda_{13}} = 0.5, \quad f_{21} = \frac{\lambda_{21}}{\lambda_{21} + \lambda_{23}} = 0.67, \quad f_{23} = \frac{\lambda_{23}}{\lambda_{21} + \lambda_{23}} = 0.33,$$
$$f_{31} = \frac{\lambda_{31}}{\lambda_{31} + \lambda_{32}} = 1.0$$

After that, the contention probabilities are found by plugging these probabilities into Eq. 5.8.

5.4 Performance Analysis of Router, Shared Bus and Point-to-Point Configurations

In this section, we first show how we model the performance of routers with multiple virtual channels (VCs) using the proposed approach. Then, we show that the on-chip router model given captures the shared bus architecture with Round Robin arbitration policy as a special case. Using this model, we formally prove that the on-chip router has always a better performance compared to a shared bus architecture. Finally, we compare the performance of on-chip routers with varying

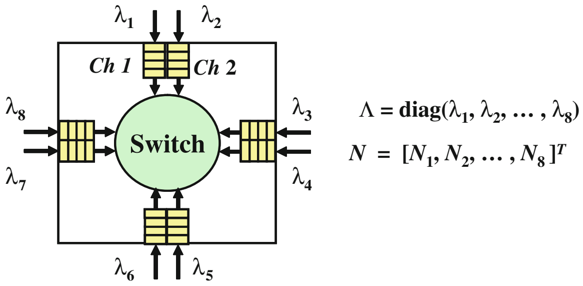


Fig. 5.3 The router model with multiple virtual channels. The outgoing links are not shown for simplicity. The structure of λ, and N remains the same as for the no virtual channel case

number of virtual channels against the performance of a shared bus using a multimedia system as a driver application.

5.4.1 Router with Multiple Virtual Channels

When there are multiple virtual channels connected to each physical port, we can view the router as shown in Fig. 5.3. Essentially, the input received from the physical link is statically demultiplexed to the available virtual channels. The on-chip router model developed in Sect. 5.3 is valid when there are multiple virtual channels. In this case, the state of the router is given by the average number of packets at each virtual channel; hence the dimension of the router model becomes the total number of input channels available, i.e., number of physical channels (P) times the number of virtual channels per physical channel (V).

The second major change is related to the contention matrix. The contention matrix is computed as shown in Sect. 5.3.3 and takes into account the forwarding probabilities between each pair of virtual channels.

5.4.2 Performance Models for Shared Bus and Point-to-Point Architectures

Figure 5.4 shows a set of processing elements connected through a shared bus. The PEs write the messages to dedicated buffers which correspond to the input channels of the router shown in Fig. 5.2. When more than one buffer has data ready to be sent, an arbitration phase takes place to determine which flow can use the shared bus. As opposed to the router where simultaneous connections between different input channels are possible, the shared bus can be used only by a single flow at any given time. This means that the contention probabilities between each pair of buffers (i.e. c_{ij}'s in Eq. 5.3) are equal to one; that is,

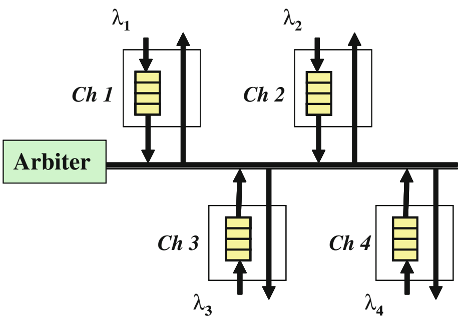


Fig. 5.4 Interconnection of the input channels shown in Fig. 5.2 using a share bus. While the router architecture shown in Figs. 5.2 and 5.3 allow multiple simultaneous connections, the shared bus allows only one connection at any given time

$$C_{Bus} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \tag{5.9}$$

so, we can use Eq. 5.5 to compute the average number of packets in the buffers connected by a shared bus as follows:

$$N_{Bus} = (I - T_{Bus}\Lambda C_{Bus})^{-1}\Lambda\overline{R} \tag{5.10}$$

where T_{Bus} is the service time of the Bus architecture and C_{Bus} is the contention matrix given in Eq. 5.9.

Finally, we note that when each buffer shown in Fig. 5.4 is connected to all the remaining buffers via P2P links, the contention probabilities will be zero (i.e., $c_{ij} = 0$ when $i \neq j$). Consequently, the average number of packets in the buffers connected by dedicated P2P links is found as follows:

$$N_{P2P} = (I - T_{P2P}\Lambda I)^{-1}\Lambda\overline{R} = (I - T_{P2P}\Lambda)^{-1}\Lambda\overline{R} \tag{5.11}$$

where T_{P2P} is the service time of the P2P architecture and the contention matrix becomes the identity matrix. In fact, it is easy to see that Eq. 5.11 merely models the system as a combination of independent M/G/1 buffers, which is expected.

5.4.3 Analytical Performance Comparisons

Since we obtained analytical expressions of average number of packets for NoC, shared bus, and P2P architectures, we can now make a qualitative comparison among them. In general, in order to prove that the NoC architecture performs better than the shared bus architecture, we need to prove that $N_{NoC} \leq N_{Bus}$, i.e.,

$$\begin{gathered} N_{NoC} \leq N_{Bus} \\ (I - T_{NoC}\Lambda C_{NoC})^{-1}\Lambda\overline{R} \leq (I - T_{Bus}\Lambda C_{Bus})^{-1}\Lambda\overline{R} \end{gathered} \tag{5.12}$$

Similarly, in order to prove that the P2P architecture performs better than the NoC architecture, we need to prove that $N_{P2P} \leq N_{NoC}$. In what follows, we provide a proof based on the properties of the contention matrix C, which makes the major difference between these three architectures.

In general, the service times T_{P2P}, T_{NoC}, and T_{Bus}, depend on the micro-architecture (e.g., how many clock cycles the arbitration, buffer read/write, routing take) and layout design (e.g., how fast each operation is performed or what is the clock frequency). While the shared bus and P2P architectures enjoy the simplicity of the control logic, the clock frequency may be lower due to large capacitive load of the long interconnects. On the other hand, it usually takes more clock cycles to pass through the router, but the router may operate faster due to the structured wiring. Here, we consider the case when the overall service times are equal, i.e., $T_{P2P} = T_{NoC} = T_{Bus}$, and leave the general solution for future work.

Lemma 1 *Let $C_2 = C_1 + \delta$, where C_1, C_2 and δ are $P \times P$ square matrices and, $\delta_{i,j} \geq 0$ for $\forall 0 \leq i, j \leq P$, i.e., all elements of δ are non-negative. Suppose that T is a positive constant and Λ is a $P \times P$ diagonal matrix with non-negative components, such that*[3]

$$\lim_{k\to\infty}(T\Lambda C_1)^k = 0, \quad \lim_{k\to\infty}(T\Lambda C_2)^k = 0 \tag{5.13}$$

Then,

$$(I - T\Lambda C_2)^{-1} \geq (I - T\Lambda C_1)^{-1} \tag{5.14}$$

where the inequality applies to each element of the matrices.[4] *That is, each component of $(I - T\Lambda C_2)^{-1}$ is greater than or equal to each component of $(I - T\Lambda C_1)^{-1}$.*

Proof When the conditions given in Eq. 5.13 hold, we can use the Neumann series expansion for matrix inversion, i.e.

$$(I - T\Lambda C_i)^{-1} = \sum_{k=0}^{\infty}(T\Lambda C_i)^k \text{ for } i = 1, 2 \tag{5.15}$$

So, we need to prove

$$\sum_{k=0}^{\infty}(T\Lambda C_2)^k \geq \sum_{k=0}^{\infty}(T\Lambda C_1)^k \tag{5.16}$$

To prove the inequality given in Eq. 5.16 holds, we show that

[3] Equivalently, the eigenvalues of matrices $T\Lambda C_1$ and $T\Lambda C_1$ are less than one.

[4] Note that, in this chapter, we use A $\geq$ B to denote the element-wise comparison for two matrices of same dimensions.

$$(T\Lambda C_2)^k \geq (T\Lambda C_1)^k \text{ for } \forall k \geq 0 \tag{5.17}$$

This inequality is trivially satisfied for $k = 0$. We employ mathematical induction to show that the equality also holds for $k > 0$.

For $\boldsymbol{k = 1}$: $T\Lambda C_2 - T\Lambda C_1 = T\Lambda(C_2 - C_1) = T\Lambda\delta \geq 0$, since all elements of Λ and δ are non-negative.

For $\boldsymbol{k = n}$: Suppose $(T\Lambda C_2)^n \geq (T\Lambda C_1)^n$ and let $(T\Lambda C_2)^n - (T\Lambda C_1)^n = \Delta_n \geq 0$, i.e., all components of Δ_n are non-negative.

For $\boldsymbol{k = n + 1}$: Let

$$(T\Lambda C_2)^{n+1} - (T\Lambda C_1)^{n+1} = \Delta_{n+1} \tag{5.18}$$

$$\begin{aligned}
\Delta_{n+1} &= (T\Lambda C_2)(T\Lambda C_2)^n - (T\Lambda C_1)(T\Lambda C_1)^n \\
&= T\Lambda\{C_2(T\Lambda C_2)^n - C_1(T\Lambda C_1)^n\} \\
&= T\Lambda\{(C_1 + \delta)(T\Lambda C_2)^n - C_1(T\Lambda C_1)^n\} \\
&= T\Lambda\{(C_1\{(T\Lambda C_2)^n - (T\Lambda C_1)^n\} + \delta(T\Lambda C_2)^n\} \\
&= T\Lambda\{C_1 \mathit{\Delta}_n + \delta(T\Lambda C_2)^n\} \\
&\geq 0
\end{aligned} \tag{5.19}$$

In line (1) of Eq. 5.19, we simply rewrite Eq. 5.18 by factoring out $T\Lambda C_2$ and $T\Lambda C_1$, respectively. Then, we further factor $T\Lambda$ out and substitute C_2 with $C_1 + \delta$, in lines (2) and (3). Next, we observe that $(T\Lambda C_2)^n - (T\Lambda C_1)^n = \Delta_n$ in line (4). Finally, we note that all the matrices in line (5) are non-negative. So,

$$(T\Lambda C_2)^k \geq (T\Lambda C_1)^k \, for \, \forall k \geq 0$$

Hence,

$$\sum_{k=0}^{\infty}(T\Lambda C_2)^k \geq \sum_{k=0}^{\infty}(T\Lambda C_1)^k$$

since all the terms in the left hand side are greater than or equal to the corresponding terms in the right hand side.

□

Example 4 Consider the case $P = 1$. The lemma reduces to the following statement:

For $a \geq 0, c_1 \geq 0, c_2 \geq 0, ac_1 < 1, ac_2 < 1,$

$$c_2 \geq c_1 \Rightarrow \frac{1}{1 - ac_2} \geq \frac{1}{1 - ac_1}$$

Theorem 1 *Consider P buffers connected by a router* (*as in* Fig. 5.2), *by a shared bus* (*as in* Fig. 5.4), *and via point-to-point links. Also*, *assume that*:

(i) *The arrival rates to the buffers, $\lambda_1, \lambda_2, \ldots, \lambda_{\mathrm{P}}$ are all identical and follow the assumptions stated in* Sect. 5.3.1;
(ii) *The service times are all equal, i.e. $T_{P2P} = T_{NoC} = T_{Bus} = T$,*
Then, we have that: $N_{P2P} \leq N_{NoC} \leq N_{Bus}$.

Proof Let

$$C = C_{Bus} = C_{NoC} + \delta_{NoC} = C_{P2P} + \delta_{P2P} + \delta_{NoC} \tag{5.20}$$

where δ_{P2P} and δ_{NoC} are matrices with non-zero entries. We can write these relations, since C_{P2P} is the $P \times P$ identity matrix, C_{Bus} consists of all ones, and $0 \leq c_{i,j} \leq 1$ for all entries of C_{NoC}.

The average number of packets in the buffers for the shared bus case is given by Eq. 5.5 as follows:

$$N = (I - T\Lambda C)^{-1}\Lambda\overline{R}$$

We are interested in the operating domain where the inverse of $T\Lambda C$ exists, i.e., N is finite.[5] When $\lim\limits_{k\to\infty}(T\Lambda C)^k$ exists (or equivalently the eigenvalues of the matrix $(T\Lambda C)$ are less than one [16]), Eq. 5.15 holds. Consequently, using the result of Lemma 1, we have:

$$(I - T\Lambda C_{P2P})^{-1} \leq (I - T\Lambda C_{NoC})^{-1} \leq (I - T\Lambda C_{Bus})^{-1}$$

where the equality holds for each component of the $P \times P$ matrices. Finally, since all elements of $\Lambda\overline{R} \geq 0$, using Eq. 5.5 we obtain:

$$\begin{array}{ccccc} (I - T\Lambda C_{P2P})^{-1}\Lambda\overline{R} & \leq & (I - T\Lambda C_{NoC})^{-1}\Lambda\overline{R} & \leq & (I - T\Lambda C_{Bus})^{-1}\Lambda\overline{R} \\ N_{P2P} & \leq & N_{NoC} & \leq & N_{Bus} \end{array} \tag{5.21}$$

□

Equation 5.21 gives the relation between the average number of packets in queues connected by P2P links, router and shared bus under identical service times and arrival rates. The relation for arbitrary arrival rates and service times can be analyzed using Eqs. 5.5, 5.10 and 5.11, respectively.

5.4.4 Using Equation 5.5 for Router Design

The router model described by Eq. 5.5 provides an analytical approach to analyze the effect of various router parameters on network performance. Consider the multimedia system design in [8] where the packets in the network carry data as

[5] When $P = 1$, this condition reduces to $\lambda T < 1$ known as the stability condition for a single buffer.

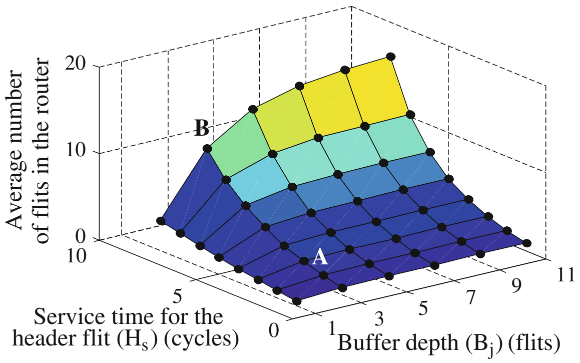


Fig. 5.5 The average number of flits in the router (i.e., the sum of the flits in all five buffers) is shown as a function of the buffer size and service time of the router

8 × 8 fixel blocks. Each pixel value is represented by 16 bits, so $S = 1024$ bits. We assume that the channel bandwidth is given by $W = 256 \times f_{ch}$, where f_{ch} is the clock frequency of the router.

Two major concerns in router design are the number of pipeline stages, i.e., the number of cycles it takes to route the header flit (H_s), and the size of the input buffers (B). To analyze the impact of these parameters on router utilization, we first map the system to a 4 × 4 mesh network running under XY routing, and determine arrival rates (Λ) and the contention matrix C for the bottleneck router. Then, we use Eq. 5.5 to analyze the impact of H_s and B on buffer utilization.

Figure 5.5 shows the average number of flits in the router (at all buffers) as a function of H_s and B_j. For a given buffer size, the average number of flits in the router increases with increasing service time, as expected. This increase is more severe for larger buffers, since more flits are stored in the buffer before being blocked. Likewise, for a given service time, the router utilization saturates, as the buffer size increases. The saturation occurs earlier for lower service times, as depicted in Fig. 5.5. For example, when $H_s = 2$, increasing the buffer size beyond $B_j = 2$ for $1 \leq j \leq 5$ does not increase the buffer utilization (see point "A" in Fig. 5.5), since the router is very fast. On the other hand, for larger service times (e.g., $H_s = 8$, point "B" in Fig. 5.5), the saturation point moves further away, i.e., more flits wait in the buffer before being served.

This case study illustrates the possible use of the proposed router model as a powerful tool for router design space exploration. Indeed, this model can be used by designers to evaluate possible trade-offs offered by different design choices (e.g., buffer size, channel width) that are nowadays determined mostly in an ad-hoc manner.

5.5 Network Performance Analysis

The router model presented in Sect. 5.3 enables the calculation of the average utilization of the input buffers given the traffic input to the router. In this section, we discuss how this model can be actually utilized to analyze the performance of the *entire* network. More specifically, we compute the average buffer utilization in the router, average packet latency and the maximum network throughput using the proposed model.

The processing elements connected to the routers, put the new packets first to an egress queue. This queue is connected to one of the input ports of the router. Therefore, packets experience some latency before being injected to the network; M/G/1/*m* queuing is used to model this queuing delays at the sources. In order to account for these effects, we compute the traffic arrival rates (λ_{ij}) at channel i being routed through channel j for all routers as follows:

$$\lambda_{ij} = \sum_{\forall s} \sum_{\forall d} x_{sd} \Re(s, d, i, j) \tag{5.22}$$

where $\Re$ is the *routing function* such that $\Re(s, d, i, j) = 1$ if the packet sent from the source PE s to the destination PE d is routed through the input channel i and routed via output channel j of a router, and $\Re(s, d, i, j) = 0$ otherwise.

We note that the routers are typically interconnected in a network. Hence, the service time for the header flits may increase due to chained blocking at the downstream routers. In general, the blocking probabilities, hence the expected waiting time of the header flit due to blocking, can be computed using an iterative process similar to [6] or through computation ordering [7].

In order to compute the delay experienced at the intermediate routers, in our experimental work, we first apply Eq. 5.22 by going through all the flows x_{sd} and, following the routing algorithm, traverse the network from source s to destination d to find the packet arrival rates at each input port of each router. Knowing the traffic arrival rates λ_{ij}, we compute the forwarding probabilities, i.e., matrix F using Eq. 5.7 and then we compute the contention matrix C using Eq. 5.8; this captures the congestion effects at various points in the network. Finally, we use Eq. 5.5 to find the utilization of each input port.

5.5.1 Average Buffer Utilization and Packet Latency

Given the arrival rates, λ_{ij}, at each input channel in the network, the contention matrix for each router, and $\boldsymbol{T}$, we use Eq. 5.5 to find the average number of packets in the input buffer at each router. This information can be used for optimization purposes (e.g. to determine the buffer sizing), since buffer utilization provides information about the distribution of the traffic load across the network. The average buffer utilization can also be used to compute the average waiting time in buffers. By Little's theorem:

$$W_{ij} = N_{ij}/\lambda_{ij} \tag{5.23}$$

where W_{ij} is the average waiting time in the channel j buffer at router i. Since we already know the packet service time, W_{ij} enables us to compute the average packet latency at each router.

The delay experienced at each router is a performance metric with very fine granularity. Indeed, it can be used to compute the average latency for each traffic source/destination pair separately, as well as the average packet latency in the network. When a packet is sent from the source node s to the destination node d, it traverses a set of routers and the corresponding input buffers denoted by $\prod_{sd}$. The average latency for any packet from node s to node d (denoted by L_{sd}) is given by:

$$L_{sd} = W_s + \sum_{(i,j)\in\prod_{sd}} (W_{ij} + T)$$

where W_s is the queueing delay at the source, W_{ij} is the queuing delay at channel j of router i, and T is the average service time. W_s is computed using the $M/G/1/m$ model, since the buffers in the PEs are also finite. Then, the *overall average packet latency* in the network is found as:

$$L = \frac{1}{\sum_{\forall s,d} x_{sd}} \sum_{\forall s,d} x_{sd} \times L_{sd} \tag{5.24}$$

This relation provides fast and accurate estimates of L for a variety of traffic patterns and application mappings, as in Sect. 5.6. It can be applied to a wide range of optimization problems, since average packet latency is a common performance metric.

5.5.2 Network Throughput

The network throughput is defined as the rate at which the packets are delivered to the destination nodes. At low traffic loads, the packet delivery rate is equal to the packet injection rate. However, as the traffic load increases, the throughput starts saturating. In order to investigate the impact of increasing packet injection rates on the average number of packets at the router input buffers and implicitly on the network throughput, we multiply the source to destination traffic generation rates by a positive scaling parameter (i.e., $\alpha > 1$ but still close to 1). Scaling the traffic arrival rates to the router inputs, $\alpha\lambda_j$, in Eq. 5.5 allows us to write the following relation between the throughput N and scaling parameter α:

$$N(\alpha) = (I - \alpha T \Lambda C)^{-1} \alpha \Lambda R \tag{5.25}$$

where $N(\alpha)$ is the average number of packets in the router as a function of α. When the utilization of the input buffers approaches unity, the router will be always busy so its throughput will saturate. The approximate value of α that will saturate a given router can be found by solving the following equation:

$$\sum_{j=1}^{P} N_j(\alpha) = 1 \tag{5.26}$$

We solve Eqs. 5.25 and 5.26 to find the minimum value of α over all the routers, i.e. α_{min}. Then, the traffic generation rates at which the application throughput saturates is found as $\alpha_{min} x_{sd}$. Finally, the *saturation throughput* of the network is found as:

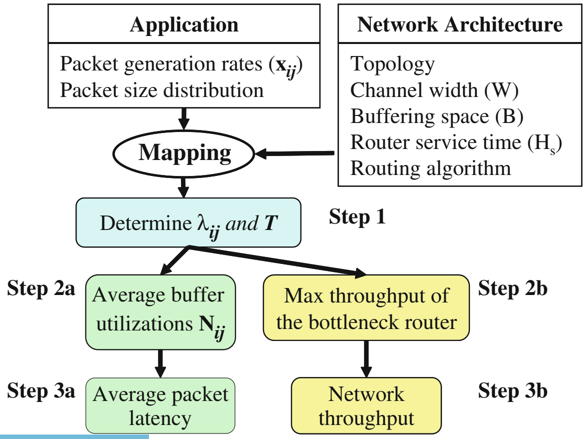
$$\Gamma = \alpha_{min} \sum_{\forall s,d} x_{sd} \tag{5.27}$$

In summary, the basic idea of our approach is to identify the bottleneck router, which happens to be the router with the highest amount of traffic through it. The critical load of this router defines the critical load of the overall network, since the congestion propagates quickly across the entire network.

5.5.3 *Overview of the Performance Analysis Methodology*

The proposed analysis technique is summarized in Fig. 5.6. First, the traffic input rates to the routers and packet service time (including the waiting time due to blocking) are computed using Eqs. 5.1 and 5.22. In order to find the average packet latency, we follow the path on the left in Fig. 5.6. The average utilization of the input buffers in the routers are found using Eq. 5.5 (Step 2a). Next, the average packet latency in the network is found using Eq. 5.24 (Step 3a). Finally, to find the

Fig. 5.6 Overview of the proposed performance analysis approach



saturation throughput, we identify the bottleneck router and use Eq. 5.27 (Steps 2b and 3b in Fig. 5.6).

5.6 Experimental Results

This section provides a detailed study on the accuracy and run-time of the proposed approach. The analytical results obtained using the proposed method are compared against those obtained with a cycle-accurate (flit-level) NoC simulator [22]. Both the simulator and the analytical model are implemented in C++ and tested on a Pentium 4 computer with 512M memory running Linux OS.

Throughout the experiments, we assume that each input and output buffer slot can hold a 64-bit flit. In the absence of contention, the router service time is 4 cycles. We also assume that the link transmission takes 1 cycle. Simulations run for 5×10^4 cycles with an initial warm-up period of 2000 cycles. Also, simulations of a particular configuration are repeated 100 times with different seeds in order to collect reliable averages.

5.6.1 *Average Packet Latency*

We first consider the multimedia application in [8] which is manually mapped to a 4×4 2D mesh network with input and output buffer sizes of 5×64 and 1×64 bits, respectively. We compare the average packet latency obtained using the proposed approach against the values obtained by simulation. The average packet latency as a function of the packet injection rate is shown in Fig. 5.7.

We observe that the latency values estimated by the proposed approach follow the simulation results closely. More precisely, for packet injection rates below 0.2 *pkt/cycle*, the relative error between the analytical and simulation results is within 5 %. After that, the latency values start increasing abruptly, since at this critical

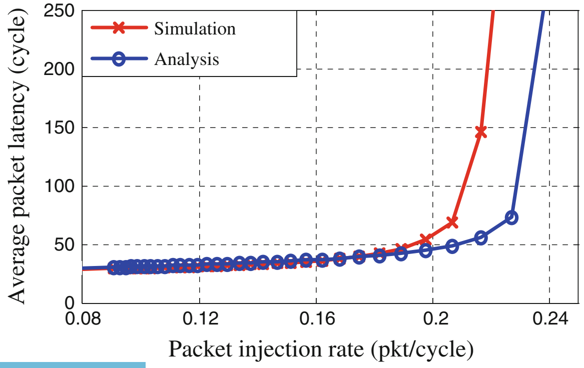


Fig. 5.7 Average packet latency values found with the proposed approach and by simulation are shown

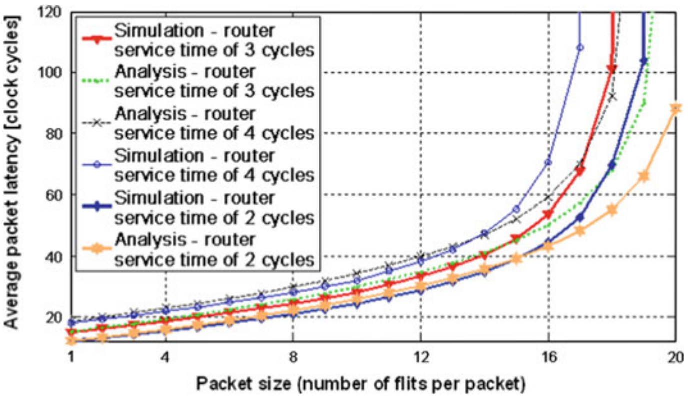


Fig. 5.8 Average packet latency found analytically and by simulation as a function of packet sizes (i.e., number of flits per packet) and three router service times

traffic load the network enters the congestion region. Our approach is also capable of estimating this critical value, as we demonstrate in Sect. 5.6.3.

We also investigate the impact of packet sizes (i.e., the number of flits per packet) and router service time (i.e., 2, 3 and 4 clock cycles) on the average packet latency for a multimedia application mapped on a 4 × 4 2D mesh NoC with input and output buffer sizes of 5 × 64 and 5 × 64 bits, respectively, and a packet injection rate of 0.16 packets/cycle. The channels are 64-bit wide and link traversal latency is 1 clock cycle. The simulation length was 2×10^6 clock cycles.

As shown in Fig. 5.8, the average packet latency values are close for both analysis and simulation for all three router service times and for packet sizes up to 16 flits per packet. For instance, the average packet latency for a router service time of 2 clock cycles and packet sizes of 16 flits is 44.5 cycles via our proposed approach and 43.2 cycles via simulation. Similarly, for router service time of 3 clock cycles and packet sizes of 15 flits, the average packet latency is 45.2 cycles via our approach and 45.7 via simulation. The differences that emerge for packet sizes larger than 16 flits per packet can be attributed to the congestion effects.

Next, we assess the accuracy of our approach for different application mappings. We performed experiments for 1000 random mappings. For each mapping, the average packet latency is computed using the proposed approach and by simulation, at 0.16 *pkt/cycle* injection rate, which is a possible operating point (see Fig. 5.7). We repeat each simulation 50 times with different seeds; the results are averaged such that the measured latency is within one standard deviation of the actual value with 95 % confidence. More formally, let $L_S(i)$ be the average packet latency for mapping i obtained by simulation and $L_A(i)$ be the corresponding latency obtained using Eq. 5.24. The relative error between the analytical and simulation results, for 1000 different mappings, is:

$$Err = \frac{1}{1000}\sum_{i=1}^{1000}\frac{|L_S(i) - L_A(i)|}{L_S(i)} \tag{5.28}$$

Using this definition, the relative error between the analytical and simulation results is about 9 %. This is actually a very good accuracy level, given that the relative error is very sensitive even to small differences in data values.

5.6.2 Case Study: Application Mapping

In general, the NoC design space is too big to explore by simulation. For instance, there are *n!* different ways to map a given application to a network with *n* nodes. Since the proposed performance model targets NoC design and optimization, we illustrate the effectiveness of our approach using application mapping, which is a common optimization problem for NoCs [8, 13]. More precisely, based on the average packet latency, we first rank order 1000 different mappings obtained in Sect. 5.6.1. It takes about 22 h to find the best mapping through simulation,[6] whereas our approach completes the analysis of all possible solutions in about 7 s, which is about *4 orders of magnitude faster!*

According to the simulation results, the best among all 1000 mappings is the mapping with ID 268 with an average latency of 35.5 cycles. According to the analysis we propose, the best mapping is the one with ID 732 which has average latency of 35.3 *cycles*. The latency for mapping ID 732 found by simulation is 36.2 *cycles*. As such, the analysis approach selects a mapping whose latency is within 2 % of the best one found by simulation. We also note that by using a zero-load model (i.e., ignoring the impact of communication) results in 33.54 % error in average packet latency estimation when compared to the simulation reference. Additionally, the analysis discovers the best mapping 4 orders of magnitude faster than the simulation approach and so much more mappings can be explored, within the same time budget, using the proposed analytical technique.

To evaluate the analysis approach from a different angle, assume now that the objective is to select the 10 best mappings for more detailed evaluations. Therefore, we denote the top 10 mappings obtained via simulation as being the golden set. Then, we find the top *k* mappings based on the analysis results, where $1 \leq k \leq 100$. When we pick strictly the top 10 mappings based on analysis, only 5 mappings selected by simulation are missed. However, the number of misses drops exponentially to zero as *k* increases. For instance, the top 20 mappings picked by our approach include 7 best mappings found by simulation, while top 46 mappings contain all 10 best mappings, as shown in Fig. 5.9. For completeness, we also

[6] Each mapping is simulated 100 times and the average latency over all runs is used for ranking to increase the confidence level of the results. Run-time comparisons against a single simulation run are presented in Sect. 5.6.5.

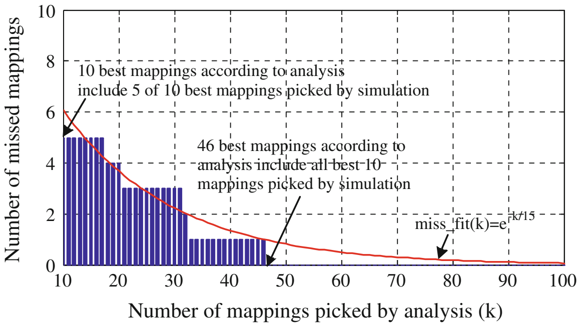


Fig. 5.9 The top mappings selected by simulation, but missed by analysis are shown

select the top 10 mappings according to zero-load model and do a pairwise comparison between these mappings in terms of average latency. Then, using simulation results, we check whether or not the conclusion drawn based on zero-load model (e.g., mapping configuration *i* is better than mapping *j*) is correct. We find that 69 % of the comparisons agree with the simulation results, while 31 % of the comparisons result in wrong decisions. Finally, we repeat the same experiment using the mappings obtained via the proposed approach. We observe that 87 % of the comparisons lead to the same conclusion with the simulation results. Hence, the proposed approach increases the comparison accuracy from 69 to 87 % which means about 26 % improvement.

To sum up, the proposed method can be used to prune the large design space in a very short time compared to simulation. Experiments performed on larger networks show several orders of magnitude achievable speed-up compared to a single simulation run. Considering that many simulations are needed to obtain high confidence intervals, the overall speed-up due to the analytical approach is significant. Moreover, the simulation run-time grows faster for heavier traffic, while the run-time of the analytical approach remains pretty much the same.

5.6.3 *Network Throughput*

Next, we compare the maximum network throughput obtained via simulation against the analysis results found using Eq. 5.27. In order to test the robustness of our approach to *non-uniform* traffic conditions, each node communicates only with the nodes that are located within a forwarding radius. Furthermore, if the distance between the source and destination nodes is given by *dist*(*s,d*), then the forwarding probability $p_f(s,d)$ is:

$$P_f(s,d) = \begin{cases} \sim 1/dist(s,d) & dist(s,d) \leq F_R \\ 0 & dist(s,d) > F_R \end{cases} \tag{5.29}$$

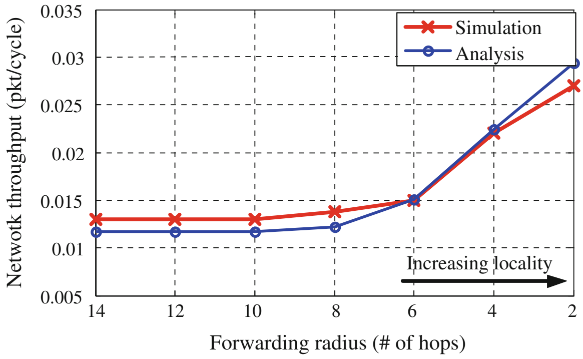
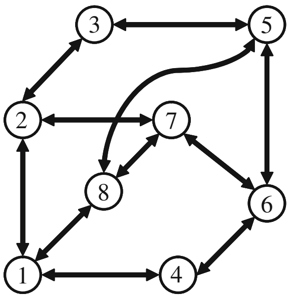


Fig. 5.10 Sustainable network throughput for 8 × 8 2D mesh network with local traffic described by Eq. 5.29

where F_R (number of hops) is the radius of the forwarding region. The maximum network throughput of a 8 × 8 mesh network, as a function of the traffic locality is given in Fig. 5.10. As expected, the network throughput increases with the level of the locality. Furthermore, our technique provides a close approximation to the simulation results over a wide range of characteristics in the traffic locality.

5.6.4 Application to Arbitrary Topologies

Since the proposed performance analysis is general, we apply it now on arbitrary topologies [14]. To this end, we analyze and simulate the simple network in Fig. 5.11a. Figure 5.11b describes the traffic pattern and the deadlock-free routing algorithm used in the network. The entries of routing matrix, RM$(i,j) 1 \le i,j \le 8$, show whether there is communication between nodes i and j, and the routing choice in case they communicate. For instance, in Fig. 5.11b, RM$(1,5) = –$ implies that node 1 does not send packets to node 5. On the other hand, RM$(1,6) = 4$ means that node 1 forwards the packets to node 4, when it needs to



(a)

RM =

Id	1	2	3	4	5	6	7	8
1	–	2	2	4	–	4	8	8
2	1	–	3	–	–	–	7	7
3	2	2	–	–	5	–	–	–
4	1	1	1	–	–	6	–	–
5	3	3	3	–	–	6	–	–
6	4	4	5	4	5	–	7	–
7	2	2	2	–	–	–	–	8
8	1	5	–	–	–	–	7	–

(b)

Fig. 5.11 **a** Arbitrary network topology used to test the proposed technique and **b** the Routing Matrix (RM) communicate with node 6. For the pairs that communicate the traffic rate is uniform. Finally, the traffic load between all pairs of communicating nodes is uniform and the packets consist of 15 flits.

The maximum throughput of this network is found as 0.2 *packets/cycle* using our technique. To evaluate the accuracy of this value, we also run 50 simulations with different random seeds and identify the maximum throughput as 0.18 *packets/cycle*. As such, the difference between simulation and analysis is about 11 %.

5.6.5 Complexity and Run-Time Analysis

Finally, we compare the run-time of the proposed analysis approach for latency computation (left branch in Fig. 5.6) with the run-time of the simulator. The methodology has three major steps:

- Computation of the input rates for each network channel (Eq. 5.22),
- Computation of the average buffer utilization and queuing delay at each input buffer (Eq. 5.5)
- Computation of the average packet latency (Eq. 5.24).

The computational complexity of solving steps 1 and 3 is proportional to the product between the number of routers (R), the number of ports per router (P), and the number of traffic flows in the network (T_f). On the other hand, the computational complexity of step 2 is proportional to $R \times P^3$. As a result the overall complexity is obtained as $\mathrm{O}(RPT_f) + \mathrm{O}(R \times P^3)$.

Both the simulator and the analytical model are implemented in C++ and tested on a Pentium 4 computer with 512M memory running Linux OS. Figure 5.12 shows the run-time values for the proposed analytical model and simulation on 2D mesh networks with sizes ranging from 4 × 4-to-12 × 12. We observe that the analysis is two order of magnitude faster than a single run of the simulation. It is important to note that the two orders of magnitude reduction we show in Fig. 5.12 is a lower bound, since it compares the analysis run-time against a single simulation run. In practice, simulations of a single configuration are repeated many

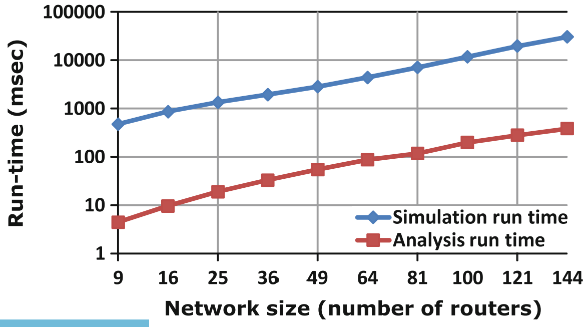


Fig. 5.12 The run times of the proposed analytical method and a single simulation run are shown for increasing network sizes. The analysis is about two orders of magnitude faster than a *single* simulation run, as the log scale y-axis shows

times to reduce the impact of randomness involved in traffic generations [18]. For instance, simulation results reported in this chapter are averaged over 100 simulations with different seeds. Consequently, the actual speed-up gained by using the proposed analytical technique as opposed to simulation is much higher as reported in Sect. 5.6.2. Finally, we also note that simulation run-time grows faster for heavier traffic, while the run-time of the analytical approach remains pretty much the same.

5.7 Summary

In this chapter, we presented a novel router model for NoC performance analysis. Our approach provides not only aggregate performance metrics such as average latency and throughput, but also feedback about the network characteristics (e.g., buffer utilization, average latency per router and per flow) at a fine-level of granularity. Furthermore, the presented approach makes the impact of different design parameters on the performance explicit so it provides invaluable insight into NoC design. As a result, the proposed approach can be used as a powerful design and optimization tool. Experimental results demonstrate the accuracy and efficiency of the analysis on real and synthetic benchmarks.

References

1. Bertsekas D, Gallager R (1992) Data networks. Prentice Hall, Upper Saddle River
2. Dally WJ (1990) Performance analysis of k-ary n-cube interconnection networks. IEEE Trans Comput 39(6):775–785
3. Dielissen J, Radulescu A, Goossens K, Rijpkema E (2003) Concepts and implementation of the Philips network-on-chip. IP-based SoC Design
4. Draper J, Ghosh J (1994) A comprehensive analytical model for wormhole routing in multicomputer systems. J Parallel Distrib Comput 23(2):202–214
5. Guan W, Tsai W, Blough D (1993) An analytical model for wormhole routing in multicomputer interconnection networks. In: Proceedings of international parallel processing symposium, April (1993)
6. Guz Z, Walter I, Bolotin E, Cidon I, Ginosar R, Kolodny A (2006) Efficient link capacity and QoS design for wormhole network-on-chip. In: Proceedings of design, automation and test in Europe conference, March (2006)
7. Hu P, Kleinrock L (1997) An analytical model for wormhole routing with finite size input buffers. In: 15th international teletraffic congress, June (1997)
8. Hu J, Marculescu R (2005) Energy- and performance-aware mapping for regular NoC architectures. IEEE Trans Comput Aided Des Integr Circuits Syst 24(4):551–562
9. Hu J, Ogras UY, Marculescu R (2006) System-level buffer allocation for application-specific networks-on-chip router design. IEEE Trans Comput Aided Des Integr Circuits Syst 25(12):2919–2933

10. Lieverse P, Van Der Wolf P, Vissers K, Deprettere E (2001) A methodology for architecture exploration of heterogeneous signal processing systems. J VLSI Signal Process Syst Signal Image Video Technol 29(3):197–206
11. Marculescu R, Ogras UY, Peh L, Jerger NE, Hoskote Y (2009) Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. IEEE Trans Comput Aided Des Integr Circuits Syst 28(1):3–21
12. Millberg M, Nilsson E, Thid R, Jantsch A (2004) Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In: Proceedings of design, automation and test in Europe conference, Feb (2004)
13. Murali S, De Micheli G (2004) Bandwidth-constrained mapping of cores onto NoC architectures. In: Proceedings of design, automation and test in Europe conference, Feb (2004)
14. Ogras UY, Marculescu R (2006) "It's a small world after all": NoC performance optimization via long-range link insertion. IEEE Trans Very Large Scale Integr Syst Special Sect Hardw/Softw Codesign Syst Synth 14(7):693–706
15. Ogras UY, Bogdan P, Marculescu R (2010) An analytical approach for network-on-chip performance analysis. In: IEEE transaction on computer-aided design of integrated circuits and systems (TCAD), vol 29, Issue 12, Dec (2010)
16. Ould-Khaoua M, Sarbazi-Azad H (2001) An analytical model of adaptive wormhole routing in hypercubes in the presence of hot spot traffic. IEEE Trans Parallel Distrib Syst 12(3):283–292
17. Pande PP, Grecu C, Jones M, Ivanov A, Saleh R (2005) Performance evaluation and design trade-offs for network-on-chip interconnect architectures. IEEE Trans Comput 54(8): 1025–1040
18. Ross S (2006) Simulation. Elsevier Academic Press, New York
19. Strang G (2009) Introduction to linear algebra, 4th edn. Wellesley-Cambridge Press, Welleseley
20. Varatkar G, Marculescu R (2004) On-chip traffic modeling and synthesis for MPEG-2 video applications. IEEE Trans VLSI 12(1):108–119
21. Wang H, Zhu X, Peh L, Malik S (2002) Orion: a power-performance simulator for interconnection networks. In: Proceedings of annual international symposium on microarchitecture, Nov (2002)
22. Worm_Sim: a cycle accurate simulator for Networks-on-Chip. http://www.ece.cmu.edu/∼sld/wiki/doku.php?id=shared:worm_sim

Chapter 6
Application-Specific NoC Architecture Customization Using Long-Range Links

Networks-on-chip (NoC) represent a promising solution to complex on-chip communication problems. The NoC communication architectures considered in the literature are based on either completely regular or fully customized topologies. This chapter presents a methodology to automatically synthesize an architecture which is neither regular, nor fully customized. Instead, the resulting communication architecture is a superposition of a standard mesh network and a few long-range links which induce small world effects. Indeed, the few application-specific longrange links we insert significantly increase the critical traffic workload at which the network transitions from a free to a congested state. This way, we can exploit the benefits offered by both complete regularity and partial topology customization.

6.1 Introduction

Regular NoC architectures based on grid-like topologies as in Fig. 6.1 provide structured global interconnects. This ensures well-controlled electrical parameters, and reduced power consumption on the global wires. However, such architectures may suffer from long packet latencies due to the lack of fast paths between remotely situated nodes. Indeed, having to traverse many hops between any two remotely communicating nodes increases the message blocking probability. This makes the message latencies unpredictable and guaranteed service operation hard to achieve. Moreover, since most of the real-life applications have widely varying communication requirements, such general purpose platforms may become less attractive for application-specific designs that need to guarantee a certain level of performance.

On the other hand, fully customized topologies [19, 23, 25] improve the overall system performance at the expense of altering the regularity of the grid structure.

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1_6,

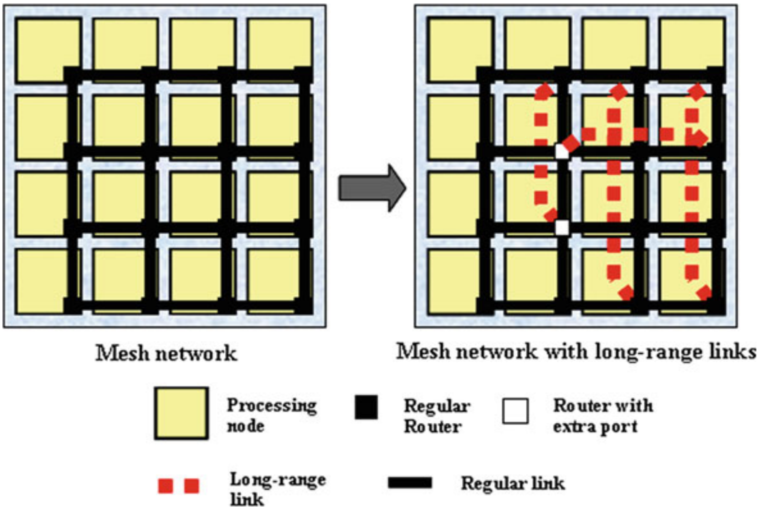


Fig. 6.1 Adding long-range links to a 4 × 4 standard mesh network

This results in global wires with widely varying lengths, performance and power consumption. Consequently, better *logical* connectivity comes at the expense of a penalty in the structured wiring. Hence, usual problems like cross-talk, timing closure, wire routing, etc. may undermine the advantages expected from customization. Besides these issues, the customized topologies require specific routing algorithms, which can be difficult to implement.

Fortunately, these two extreme points in the design space (i.e. purely regular or completely customized topologies), are *not* the only possible solutions for NoC architectures. In fact, many technological, biological, and social networks are neither completely regular, nor completely irregular [12, 27, 28]. One can view these networks as a superposition of clustered nodes with many short-range links and a few long-range links that produce shortcuts among different regions of the network. The existence of short paths between such remotely situated nodes lies at the heart of the *small-world* phenomenon, popularly known as *six degrees of separation* [16, 27]. A useful feature of these small-world networks (e.g. WWW, electrical power grid, collaboration networks) is the logarithmic relation between the mean internode distance and network size.

Starting from this idea, this chapter explores the potential of using standard mesh in conjunction with a few additional long-range links, to improve the performance of NoCs. Inserting a few long-range links to the basic regular architecture (as illustrated in Fig. 6.1) clearly reduces the average distance between remotely situated nodes. Furthermore, the node/edge connectivity, hence the network reliability, is also improved. However, long-range link insertion has to be done judiciously as it has a more pronounced, yet barely studied, impact on the *dynamic properties* of the network characterized by traffic congestion. At low traffic loads, the average packet latency exhibits a weak dependence on the traffic

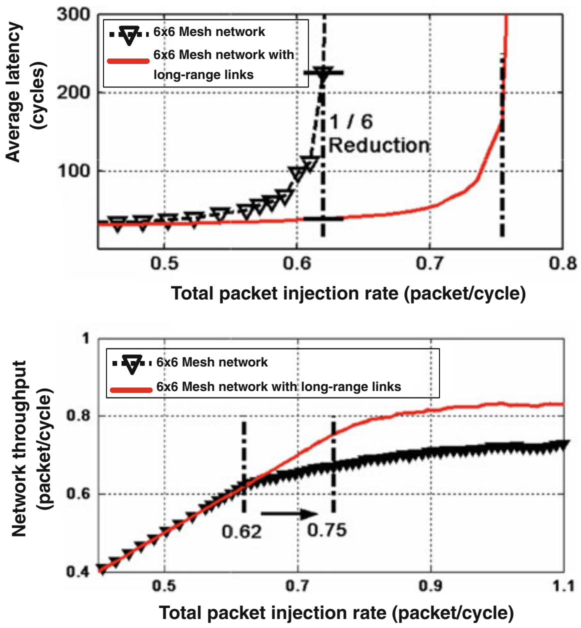


Fig. 6.2 Shift in the phase transition region due to the insertion of long-range links to a 6 × 6 mesh network

injection rate. However, when the traffic injection rate exceeds a critical value, the packet delivery latency rises abruptly and the network throughput starts collapsing (Fig. 6.2). The state before the congestion (that is, the area at the left hand side of the critical value) represents the *free state*, while the state beyond the critical value is the *congested state*. Finally, the transition from a free to the congested state is known as *phase transition* region.

The emergence of congestion in mesh networks can be significantly delayed by introducing a few additional long-range links (see Fig. 6.2) [9]. It is important to note that, due to the abrupt rise of the latency values beyond criticality, even a small right shift of the network critical value results in a huge reduction of the average packet latency. Similarly, the achievable network throughput grows significantly with the right shift of the critical traffic value. This phenomenon is at the very heart of the optimization technique presented in this chapter. Our main objective is to *optimize* the network performance (i.e. reduce the average packet latency and increase the network throughput) by maximizing the value of the critical traffic load through smart insertion of long-range links.

This chapter is organized as follows: In Sect. 6.2, we review related work, while in Sect. 6.3 propose an algorithm for long-range link insertion. The routing algorithm for the long-range links is given in Sect. 6.4. In Sect. 6.5, we discuss the practical considerations related to the implementation of long-range links. Section 6.6 discusses some energy-related issues, while Sect. 6.7 outlines possible applications of the proposed technique. The experimental results appear in Sect. 6.8. In Sect. 6.9, we summarize our contributions and indicate possible directions for future work.

6.2 Related Work

Insertion of express channels to k-ary n-cube networks in a systematic way is discussed in [4] where an interchange media is inserted periodically between the processing nodes; further, they are connected by express channels to reduce the network diameter and message latency. In [13], the authors propose express virtual channels to close the gap between the packet-switched network and the ideal interconnect. The proposed mechanism allows packets to bypass intermediate routers along pre-defined virtual express paths. This approach reduces the delay and energy consumption, while bringing the throughput close to that of a dedicated wire. This improvement is achieved without using extra links, unlike the use of express channels [4].

Previous work on links addition [9] investigate the effect of adding random links to 2D mesh (and torus) networks under the assumption of uniform traffic. The packets in the network consist of a single atomic entity containing the address information only. Moreover, due to the infinite buffer assumption, the authors of [9] do not deal with deadlock states explicitly.

In contrast to this prior work, we consider wormhole routing and routers with *bounded* input buffers. Most importantly, instead of uniform traffic, we assume *application-specific* traffic patterns and present an algorithm which inserts the long-range links by considering the traffic patterns characteristics. Due to the bounded input buffers, the additional long-range links may cause deadlock, so we also present a *deadlock-free* routing algorithm which exploits the long-range links in order to achieve the desired performance boost.

6.3 Long-Range Link Insertion Algorithm

6.3.1 System Model and Basic Assumptions

The system of interest consists of a set, T, of $m \times n$ tiles, interconnected by a 2D mesh network,[1] as shown in Fig. 6.1. The tiles of the network (referred to as *PE*s) are populated with processing and/or storage elements that communicate with each other via the network. We do not make any assumption about the distribution of the packet injection rates, but only consider the relative rate (or frequency) at which different PEs communicate with each other.

Due to limited on-chip buffer resources and low latency requirements, we assume wormhole switching. However, the results presented here are also applicable to packet- and virtual cut-through switching. The routing algorithm for the

[1] The proposed technique is applicable to topologies for which a distance definition (as in Eqs. 6.7 and 6.8) exists. The following discussion assumes a 2D mesh, since it has less links, hence, provides more flexibility compared to more densely connected networks (e.g. torus).

mesh network has to be minimal and deadlock-free, hence, *XY* routing is assumed for the mesh network. The deadlock-free property is desirable for NoCs since deadlock detection and recovery mechanisms are too expensive in terms of silicon resources and may lead to unpredictable delays.

To minimally distort the regularity of the original mesh, the number of long-range links that can be added to any router is limited to one. As such, we obtain significant performance gain with minimal modifications on the initial topology. The regular routers continue to use the default *XY* routing algorithm, while a new deadlock-free routing algorithm is proposed for the routers that have extra links.

6.3.2 Problem Formulation

The *communication volume* between the PE located at tile $i \in T$ and the PE located at tile $j \in T$ is denoted by V_{ij}. We compute the frequency of communication between the PEs i and j, f_{ij}, by normalizing the inter-tile communication volume:

$$\forall\, i,j,p,q \in T \quad f_{ij} = \frac{V_{ij}}{\sum\limits_{p} \sum\limits_{q \neq p} V_{pq}} \tag{6.1}$$

Inserting long-range links introduces an overhead due to the additional wires, extra ports in the routers and repeaters used in the implementation of long-range links. Hence, we need to have a precise measure of this overhead.

We measure the length of the long-range links, $s(l)$, in multiples of basic link units, which are identical to the regular links used in the mesh network, as shown in Fig. 6.9. This measure also reflects the repeater costs, since the number of repeaters required by a long-range link is given by $s(l) - 1$. For example, a resource constraint of S means that the total length of the long-range links inserted to the initial network consists of at most S units of standard links. Finally, the critical traffic load at which the network enters the congested phase is denoted as λ_c. Equipped with this notation, we can state now the application-specific long-range link insertion problem:

Given

- $f_{ij}\ \forall i,j \in T$
- Maximum number of links that can be added, S
- The default routing strategy for the mesh network, R

Determine

- The set of long-range links to be added on top of the mesh network, L_S
- A deadlock-free routing strategy which governs the use of the newly added long-range links,

such that

$$max(\lambda_c)\ subject\ to\quad \sum_{\ell \in Ls} s(\ell) < S \tag{6.2}$$

and *at most one* long-range link is added per router.

To give some intuition, the newly added long-range links are meant to *maximize* the critical traffic value, λ_c, subject to the total amount of available on-chip resources. Differently stated, inserting long-range links provides increased throughput and reduced latency compared to the original critical load, as shown in Fig. 6.2. We note, however, that the objective of inserting long-range links should be by no means limited to maximizing λ_c. On the contrary, other objective functions, such as increased fault-tolerance, guaranteed service, etc., can replace (or augment) the objective of maximizing λ_c.

6.3.3 Iterative Long-Range Link Insertion Algorithm

Figure 6.3 outlines our algorithm which inserts long-range links with the objective of maximizing the critical traffic load λ_c, subject to the available resources.

The algorithm starts with a standard mesh network and takes the communication frequencies (f_{ij} in Eq. 6.1) between the network tiles, the default routing algorithm (R) and the amount of resources allowed to use (S) as inputs. Then, the algorithm selects all possible pairs of tiles (i.e. $C(\|T\|, 2)$ pairs, where $\|T\|$ is the number of nodes in the network), and inserts long-range links between them. After inserting each long-range link, the resulting network is evaluated to find out the gain obtained over the previous configuration. Since we seek to maximize λ_c, the gain is measured as the increase in the critical traffic load, as detailed below in Sect. 6.3.4.

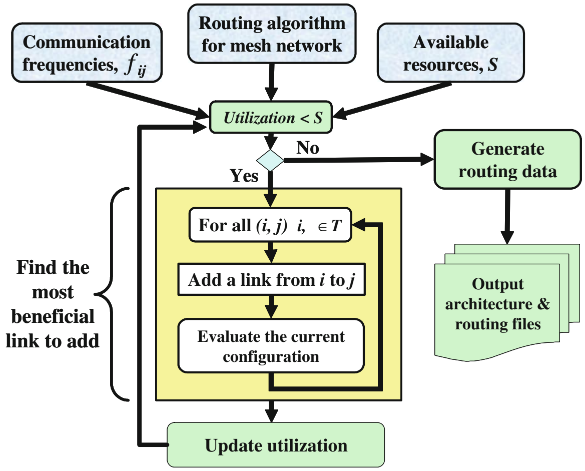

Communication frequencies, f_{ij}
Routing algorithm for mesh network
Available resources, S
Utilization < S
No
Yes
Generate routing data
Find the most beneficial link to add
For all (i, j) $i,$ $\in T$
Add a link from i to j
Evaluate the current configuration
Output architecture & routing files
Update utilization

Fig. 6.3 The flow of the long-range link insertion algorithm

After the most beneficial long-range link is found, the information about this link is stored and the amount of utilized resources updated. This procedure repeats until all available resources are used. Once this happens, the architecture file and routing data are generated for the new configuration.

6.3.4 Evaluation of the Critical Traffic Value

While the impact of the routing strategy, switching technique and network topology on the critical point have been studied through simulation [8], no work has been aimed at maximizing the critical traffic value subject to resource constraints. The major obstacle in optimizing the critical load comes from the difficulty in modeling the *variation* of the critical value, as a function of different design decisions.

Several theoreticians [9, 21] propose to estimate the criticality point of a network using mean field theory models. The key idea is to reduce the estimation of the network criticality to just *one parameter* which can be computed analytically, much faster than simulation. This is important since using accurate estimates from simulation is simply too costly to be used in any optimization loop.

In the following, we relate the critical load λ_c to the *free packet delay* τ_0, which is the packet travel time when no other packet is present in the network. Let $\lambda(t)$ be the *total packet injection rate* at time t, i.e.

$$\lambda(t) = \sum_{i \in T} \lambda_i(t), \lambda_i \textit{ is the injection rate of tile } i \in T \tag{6.3}$$

In the *free state*, i.e. when the average of $\lambda(t)$ is less than λ_c, the network is in a steady-state. Hence, the average packet injection rate (λ) is equal to the average packet delivery rate, that is,

$$\lambda \approx \frac{N_{ave}}{\tau_{ave}} \tag{6.4}$$

where N_{ave} represents the average number of packets in the network and τ_{ave} is the *average time* each packet spends in the network.

The exact value of τ_{ave} is a function of the traffic injection rate, as well as the network topology, routing strategy, etc. While there is no available analytical model for calculating τ_{ave}, we observe that τ_{ave} shows a weak dependence on the traffic injection rate when the network is in the free state. Hence, τ_0 can be used to approximate τ_{ave}. If we denote the average number of packets in the network at the onset of the criticality by N^c_{ave}, we can write the following relation:

$$\lambda_c \approx \frac{N^c_{ave}}{\tau_0} \tag{6.5}$$

This approximation acts also as an *upper bound* for the critical load λ_c, since $\tau_0 \leq \tau_{ave}(\lambda_c)$. We note that this relation can be also found using *mean field* [9] and *distance* models [30], where N_{ave}^{c} is approximated by the number of nodes in the network, under the assumption that the utilization of the routers is close to unity at the on-set of the criticality.

Since the number of messages in the network, at the onset of the criticality, is bounded by the network capacity, N_{ave}^{c}, the critical traffic load and the average packet latency are inversely proportional to each other. Indeed, if the average packet latency decreases, the phase transition is delayed, as demonstrated in Fig. 6.2, where the latency reduction is due to the presence of long-range links. Our optimization technique uses the relationship between λ_c and τ_{ave} to maximize the critical load.

- **Experimental Verification of the Eqs. 6.4 and 6.5**

For completeness, we verified Eqs. 6.4 and 6.5 experimentally, as shown in Fig. 6.4. The dotted line shows the actual packet injection rate (λ) for reference. The solid line with the square marker on it is obtained for an 8 × 8 network under *hotspot* traffic, as the ratio between the average number of packets in the network and the average packet delay at that particular injection rate.

These plots clearly show that, before entering criticality, there is a good agreement between the actual value obtained through simulation and the one predicted by Eq. 6.4. As mentioned before, the exact value of the average packet delay for a given load, $\tau(\lambda)$, is found by simulation. The dashed line with triangular markers in Fig. 6.4 illustrates the upper bound given by Eq. 6.5. We observe that this expression provides a good approximation at lower data rates and holds the upper bound property.

- **Computation of** τ_0

For arbitrary traffic patterns characterized by the communication frequencies, $f_{ij}\ \forall i,j \in T$, τ_0 can be written as:

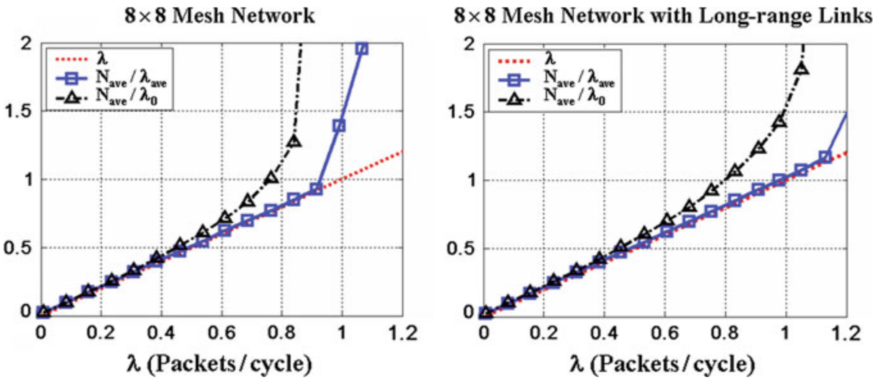


Fig. 6.4 Experimental verification of Eqs. 6.4 and 6.5 for a regular 8 × 8 mesh network and an 8 × 8 mesh with long-range links

$$\tau_0 = \sum_i \sum_{j\neq i} f_{ij}\left[d(i,j)(t_r+t_s+t_w)+max(t_s+t_w)\left\lceil\frac{L}{W}\right\rceil\right] \tag{6.6}$$

where $d(i,j)$ is the *distance* between routers i and j, and t_r, t_s, t_w are the architectural parameters representing time needed to make the routing decision, traverse the switch and the link, respectively [8]. Finally, L is the packet length and W is the width of the network channel.

For a standard mesh network, the *Manhattan distance* (d_M) is used to compute $d(i,j)$, i.e.

$$d_M(i,j) = |i_x - j_x| + |i_y - j_y| \tag{6.7}$$

where subscripts x and y denote the x- and y-coordinates, respectively. For the routers with long-range links, an *extended distance* definition is needed in order to take the long-range connections into account. Hence, we use the following generalized definition:

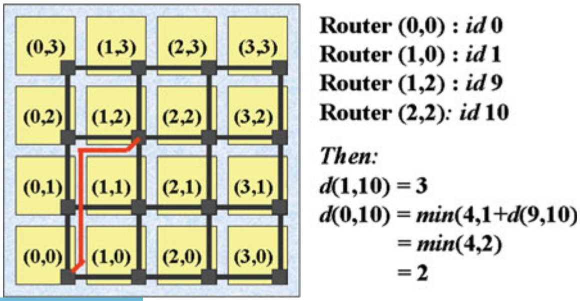
$$d(i,j) = \begin{cases} d_M(i,j) & no\ long{-}range\ link \\ min(d_M(i,j), 1 + d_M(k,j)) & if\ l(i,k)\ exists \end{cases} \tag{6.8}$$

In this equation, $l(i,k)$ means that node i is connected to node k by a long-range link. The applicability of the distance definition is illustrated in Fig. 6.5. Note that, the distance computation does *not* require any global information about the network. Hence, the routing algorithm is decentralized and its implementation is simple.

6.3.5 Small-World Properties of Networks Customized Via Long-Range Links

As discussed in Sect. 6.3.2, the application-specific long-range links are inserted to *optimize* the performance of standard grid-like networks by minimally altering their structure. The customization procedure is inspired by the small-world effect.

Fig. 6.5 Illustration of distance definition (see Eq. 6.8)



Simply stated, the small-world networks combine the advantages of short internode distance (which is a characteristic of random graphs) and high clustering (which is primarily observed in regular graphs).

We note that we are *not* trying to demonstrate that the resulting network is necessarily a small-world network, in a strict sense. In fact, the small size of the networks we are dealing with and the limited number of additional links allowed during the optimization process makes such a behavior hard to observe. Moreover, by limiting the number of additional long-range links per router to just one link prevents the emergence of true hubs which are omnipresent in many small-world networks. Instead, our algorithm for inserting long-range links *induces* small-world effects. More precisely, our algorithm decreases the average internode distance significantly, while improving the clustering coefficient. The remaining of this section demonstrates the impact of long-range links on these properties.

- **Impact of long-range links on the average inter-node distance**

In a network with application-specific traffic characterized by f_{ij}, we compute the average inter-node distance (μ) as:

$$\mu = \sum_{i} \sum_{j \neq i} f_{ij} \; d(i,j) \tag{6.9}$$

where f_{ij} and $d(i,j)$ are given in Eqs. 6.1 and 6.8, respectively.

Several theoretical studies [17, 28] assume uniform traffic, which turns out to be a special case of Eq. 6.9 if $f_{ij} = 1/(n(n-1)) \; \forall \; i,j \in T$. The reduction in the average internode distance due to the long-range links is analyzed for a 4 × 4 mesh network under uniform, *hotspot* and multimedia (*MMS*) traffic. For *hotspot* traffic, three arbitrarily selected nodes receive extra traffic compared to the remaining nodes, while for the MMS benchmark the traffic pattern is extracted from an A/V system [11]. More information about these traffic patterns is given in Sect. 6.8. In all cases, the long-range links were inserted with a constraint of $S = 12$; this translates into 4 long-range links for the networks under study.

After inserting the long-range links, under uniform traffic the average internode distance drops from 2.67 to 2.32, as shown in Table 6.1. Considering that a random network with 16 nodes and mean degree 3, would have $\mu \sim \ln(16)/\ln(3) = 2.52$, inserting long-range links indeed induces a small-world effect. For the *hotspot* and *MMS* benchmarks, the improvement is larger; this is simply because in these examples there is more room for optimization due to the skewness of the traffic patterns.

Table 6.1 The average internode distance (μ) *before* and *after* inserting long-range links to a 4 × 4 mesh network

Traffic pattern	μ without the long-range links	μ with the long-range links	Gain
Uniform4	2.67	2.32	13.1 %
Hotspot4	2.65	2.17	18.1 %
MMS	1.98	1.21	38.9 %

Table 6.2 The clustering coefficient before and after inserting long-range links to a 4 × 4 mesh network

Traffic pattern	Clustering Coefficient		
	Without LR links	The proposed algorithm	Alternative algorithm
Uniform4	0	0.05	0.21
Hotspot4	0	0.05	0.18
MMS	0	0.10	0.16

- **Impact of long-range links on the clustering coefficient**

Achieving a higher clustering compared to the random networks of exact same size is another manifestation of small-world effect. The degree of clustering, i.e. how tightly the nodes are interconnected in a network, is measured by the clustering coefficient [28]. If the node i has n_i neighbors and there are l_i links between these neighbors, then the clustering coefficient of node i, C_i, can be expressed as:

$$C_i = \frac{2l_i}{n_i(n_i - 1)} \tag{6.10}$$

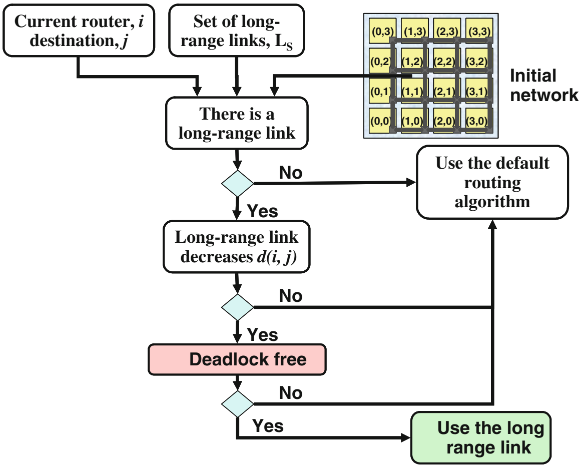
The clustering coefficient of the entire network (C_N) is found by averaging the clustering coefficients over all nodes. Hence, a large C_N implies that the nodes situated closer to each other are highly connected.

The clustering coefficient in a mesh topology is zero because none of the immediate neighbors of a given node are directly connected to each other. On the other hand, inserting long-range links increases the clustering coefficient of mesh networks. The impact of inserting long-range links on the clustering coefficient of a 4 × 4 mesh network under uniform, *hotspot* and multimedia traffic is summarized in Table 6.2. The increase in clustering coefficient is obtained as a by-product of the proposed link insertion algorithm, since the algorithm does not directly aim at improving the clustering coefficient. As explained in Sect. 6.3.4, we insert the long-range link which improves the free packet delay the most. Another alternative would be adding the link which gives the highest *performance/cost* ratio. Such an algorithm obviously favors the addition of shorter links and, hence, produces a higher clustering in the network, as shown in Table 6.2.

6.4 Routing with Long-Range Links

The routing strategy proposed in this section produces minimal paths towards the destination by utilizing the long-range links effectively. The algorithm first checks whether there exists a long-range connection to the current router, as shown in Fig. 6.6. If there is no such link, the default *XY* routing algorithm is used. Otherwise, the distance to the destination *with* and *without* the long-range link is computed using Eq. 6.8. Since only *local* information is used when computing the

Fig. 6.6 Description of the proposed routing strategy



distance, the proposed approach is scalable and provides global improvements in the network dynamics. If the long-range link produces a shorter distance to the destination, the algorithm checks whether or not using this link may cause deadlock before accepting it as a viable route. To guarantee freedom from deadlock, some limitations on using long-range links are introduced by utilizing the turn-model [10].

In the original mesh network, the links extend either along East-West (E-W) or North-South (N-S) directions. Consequently, the basic turn model prohibits one out of four possible turns to avoid cyclic dependencies. On the other hand, the long-range links can extend in two directions, such as NE-SW, NW-SE, etc. For example, the long-range link depicted in Fig. 6.5 connects two nodes with different *x* and *y* coordinates and extends along NE-SW direction. As a result, using a long-range link may result in a turn from one of the middle directions, NE, NW, SE, SW to the one of the main directions N, S, E and W. Therefore, we need to prohibit additional turns in order to avoid cycles caused by the long-range links.

In our model, we prohibit all turns from South. We also note that S-to-E and S-to-W turns for the regular links are already prohibited by the default *XY* routing algorithm. Finally, long-range links may introduce 180-degree turns. For example, the shortest path between two nodes may involve a turn from a long-range link entering the node from East to the regular link extending towards East (i.e. a W-to-E turn). To break such cycles, 180-degree turns from South and West (180° turns from negative directions) are also prohibited. As a result, we provide deadlock-free routing by limiting the routing choices for the long-range links and refer to the resulting strategy as *South-Last* routing.[2] Note that, we do not need to consider turns from a middle direction to another one, since *at most* one long-range link is connected to a node.

[2] The other possible choice that could be used with *XY* routing is North-Last routing.

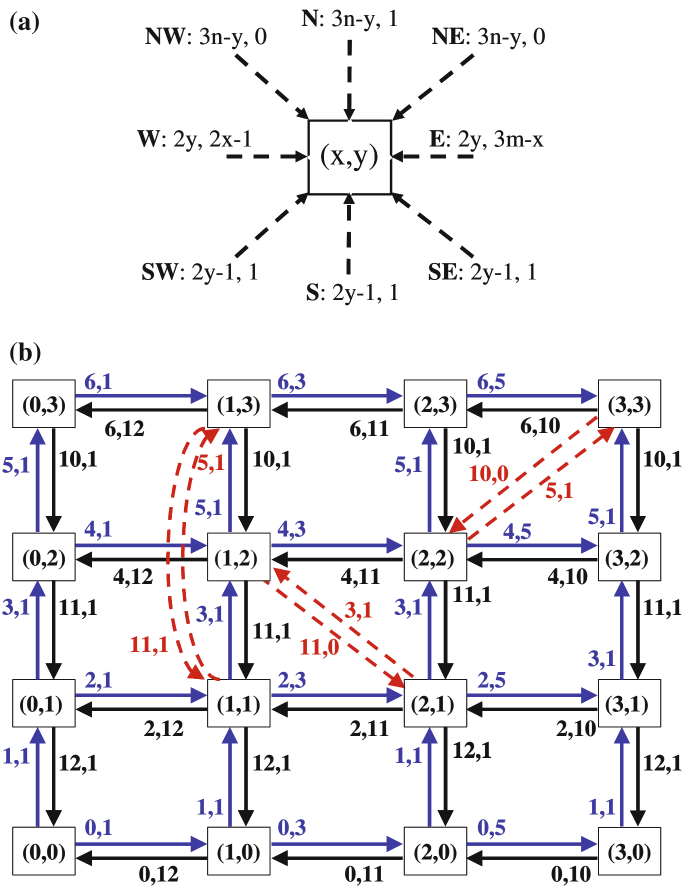


Fig. 6.7 **a** Number assignment rule for node (x, y). **b** Illustration of numbering for a 4 × 4 network

In what follows, we prove formally that the proposed routing algorithm is indeed deadlock-free. The proof can be skipped without losing the continuity of the main ideas.

Theorem 1 *The combination of XY routing for the routers without any long-range link, and South-Last routing algorithm for the routers with (at most) one long-range link on a mesh network is deadlock-free.*

Proof A routing algorithm is deadlock-free if the network channels can be enumerated such that the algorithm always routes the packets along channels with strictly increasing (or decreasing) numbers [5]. Using the notation in [10], we assign each channel in a $m \times n$ grid a two-digit number $(a, b)_r$, where $r \geq max(3m, 3n)$ and $(a, b)_r = a \times r + b$. Figure 6.7a shows the enumeration of the channels entering an arbitrary node with coordinates (x, y), where $0 \leq x \leq m - 1$ and $0 \leq y \leq n - 1$. The numbering scheme for a 4×4 mesh network with three pairs of long-range links is illustrated in Fig. 6.7b.

It can be observed that the proposed routing algorithm forwards the packets only to the channels with strictly increasing ordering. To prove that this is indeed the case for *all* packets, we analyze each possible input to an arbitrary node in Fig. 6.8a–h using the numbering scheme introduced in Fig. 6.7a. For instance, Fig. 6.8a shows an long-range link input from NW direction. The outgoing long-range link in the opposing direction can connect to node $(x - k_x, y + k_y)$ where $1 \leq k_x \leq x$ and $1 \leq k_y \leq n - 1 - y$. Investigating the channel numbers reveals that the channels which do not result in strictly increasing ordering are prohibited by the proposed algorithm.

In general, a long-range link originating from node (x, y) can be connected to a node $(x \pm k_x, y \pm k_y)$, where $1 \leq k_x \leq x$ when the long-range link extends to negative x direction and $1 \leq k_x \leq m - 1 - x$ when it extends to positive x dimension. Likewise, $1 \leq k_y \leq y$ when the long-range link extends to negative y direction $1 \leq k_y \leq n - 1 - y$ and when it extends to positive y dimension.

Figure 6.8a–h show that the proposed algorithm routes the packets to channels with increasing numbers for all possible inputs. Of particular interest are Figs. 6.8a, b, h. These figures show that the turns that do not result in a strictly increasing channel ordering are the turns from South, which are prohibited by the proposed algorithm.

We have shown that the proposed algorithm always routes the packets along the channels with strictly increasing numbers. As a result, it is deadlock-free. □

The final thing to consider is the possibility of a long-range link acting as a traffic attractor and then becoming a bottleneck in the network. For this reason, we assess the amount of traffic *already* assigned to a long-range link and route further traffic over the link only if it is not likely to become a bottleneck. This method is static and does not need feedback from the network. An alternate approach would be to monitor the congestion level on the long-range link and the downstream router and then route the packets adaptive manner using the permissible turns depicted in Fig. 6.8.

6.5 Implementation of Long-Range Links

It is widely assumed that the top 2-4 metal layers can be reserved for the network links [6, 22]. We expect that long-range links will also utilize the top metal layers. In the following, we discuss several possible approaches for implementing long-range links.

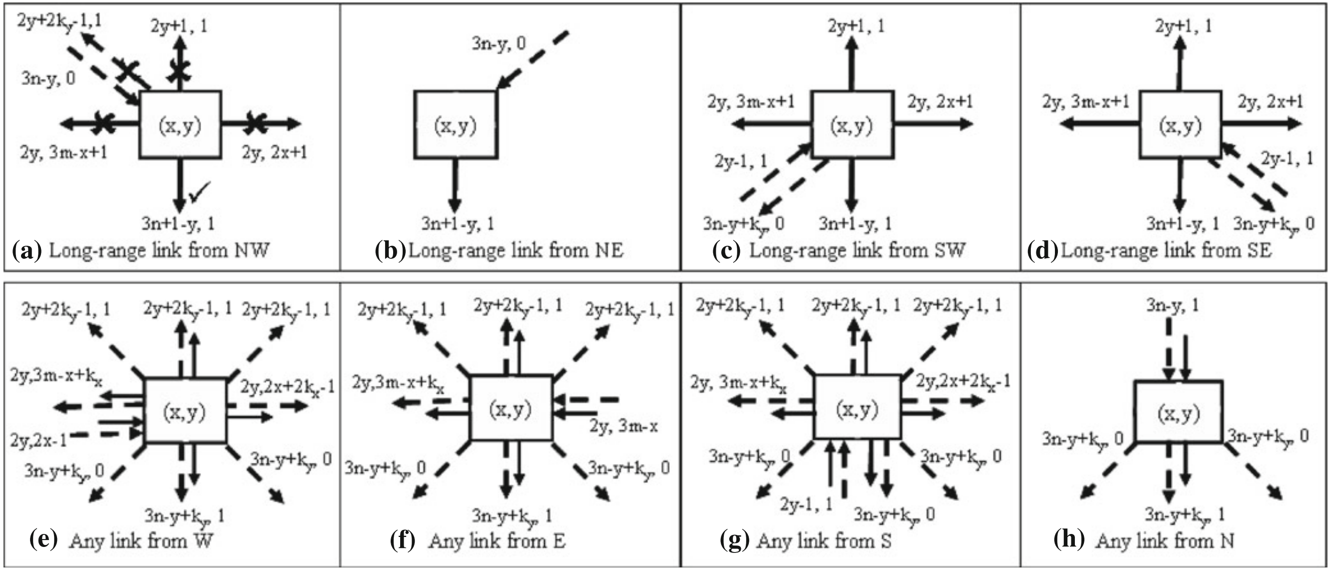


Fig. 6.8 Permissible output channels for all possible inputs to an arbitrary node are shown. **a** Long-range link from NW. **b** Long-range link from NE. **c** Long-range link from SW. **d** Long-range link from SE. **e** Any link from W. **f** Any link from E. **g** Any link from S. **h** Any link from N. For clarity, the depiction in (**a**) explicitly denotes the prohibited output links, while the remaining plots show only the permitted outputs

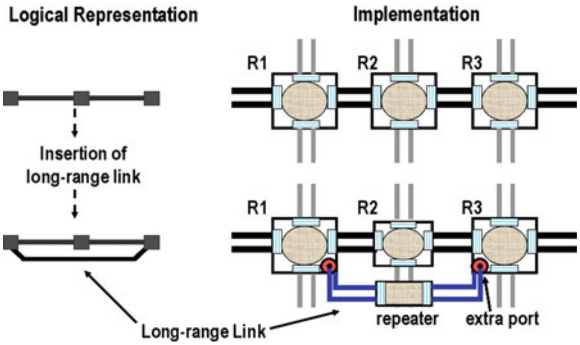


Fig. 6.9 Implementation of long-range links using repeaters. Routers 1 and 3 are both connected by Router 2 via the underlying mesh network and the inserted long-range link. The long-range link consists of two regular links connected by a repeater

6.5.1 Traditional CMOS Implementation

In order to preserve the advantages of structured wiring, the long-range links are segmented into regular, fixed-length, network links connected by repeaters. The repeaters can be thought of as simplified routers consisting of only two ports that accept an incoming flit, store it, and finally forward it to the output port, as illustrated in Fig. 6.9. The repeaters essentially pipeline the long-range links and produce routes which bypass the routers, while looking identical to the original paths provided by the routers. The use of repeaters with at least 2-flit buffering capabilities guarantees latency insensitive operation as discussed in [1, 2].

Another issue to consider is the increase in the size of the routers with extra links due to the additional port. This overhead has to be taken into account while computing the maximum number of long-range links which can be added to the regular mesh network. Although there is no theoretical limitation on the number of additional links a router can have, a maximum of one long-range link per router is used in our approach. This way the regularity of the mesh network is minimally altered, while still enjoying significant improvements over the standard mesh network.

We also note that it is possible to use more aggressive signaling techniques to implement long-range links without using the repeaters shown in Fig. 6.9. For instance, in a recent study the authors propose current-mode signaling for long-range links [18].

6.5.2 Optical Interconnects for Implementing Long-Range Links

Optical on-chip communication for NoC has been recently proposed due to increasing contribution of the interconnects to the overall power consumption and increasing wire delay [3, 24]. The advantages of optical communication include significant increase in bandwidth, a decrease in the power consumption, increased immunity to electromagnetic noises and temperature variations. On the other hand, there are important issues such as developing fabrication steps compatible with future IC technology and keeping the additional cost affordable. At the same time, a sufficiently large optical-electrical conversion efficiency is required. If these problems can be solved, long-range on-chip communication can greatly benefit from optical interconnect, while the short distance communication is still achieved through traditional copper interconnects.

6.6 Energy-Related Considerations

In this section, we investigate the proposed approach from an energy consumption point of view. One can measure the energy consumption using the E_{bit} metric [31], defined as the energy required to transmit one bit of information from the source to the destination. E_{bit} is given by:

$$E_{bit} = E_{L_{bit}} + E_{B_{bit}} + E_{S_{bit}} \tag{6.11}$$

where E_{Lbit}, E_{Bbit} and E_{Sbit} represent the energy consumed by the link, buffer and switch in the router, respectively.

The total energy consumption *before* inserting the long-range links can be expressed as:

$$E_M = \sum_{i,j} V_{ij}[d_M(i,j)E_{L_{bit}} + (d_M(i,j) + 1)E_{B_{bit}}] + \sum_{ij} V_{ij} \sum_{r} E_{S_{bit}}(r) \tag{6.12}$$

where V_{ij} and $d_M(i,j)$ are the communication volume and Manhattan distance between nodes i and j, respectively. The switching energy is summed up over all the routers the message goes through but written separately to emphasize the difference in router sizes. For example, due to the increased router arity, the switch at a router with 5 I/O ports has a larger energy consumption compared to the switch involving only 3 I/O ports.

Using a similar approach, the total energy consumption *after* the insertion of long-range links can be expressed as:

$$E_{ML} = \sum_{i,j} V_{ij}[d_M(i,j)E_{L_{bit}} + (d_M(i,j) + 1)E_{B_{bit}}] + \sum_{ij} V_{ij} \sum_{r} E'_{S_{bit}}(r) \tag{6.13}$$

where $E'_{S_{bit}}$ represents the switching energy after the insertion of the long-range links. We note that the energy consumed in the links remains the same whether or not the message uses the long-range links, since the total number of links traversed remains the same. The same argument is also valid for the buffer energy consumption, assuming that similar buffers are used in repeaters and routers.

On the other hand, the *switch* energy consumption is affected, since some messages are eventually routed through the routers with extra links, while some others will be routed through repeaters and thus bypass several routers. In fact, the latter scenario provides a reduction in the communication energy consumption due to the elimination of the crossbar switch, while the former induces a penalty due to increasing size of the router. Overall, we expect a small impact on the energy consumption.

We evaluated the energy consumption before and after inserting long-range links using the cycle-accurate *worm_sim* simulator [29] and an FPGA prototype.[3] We observe that the link and buffer energy consumption increases by about 2 % after inserting long-range links, while the switch energy consumption drops by about 7 %, on average. Furthermore, the Orion model [26] integrated to our simulator shows about 5 % savings in the overall energy consumption. Finally, our energy consumption measurements using a real FPGA prototype show that the energy consumption is minimally affected by the insertion of long-range links [20].

6.7 Practical Use of Long-Range Links

Inserting application-specific long-range links enables a higher network throughput and a lower average packet latency compared to a pure mesh architecture. As a result, the application-specific long-range links can be employed in several scenarios:

- First, when the application mapping is given, or there are tight constrains on mapping the application to the network, inserting long-range links can greatly enhance the performance of the regular mesh architecture. The proposed technique works also well in conjunction with an already existing mapping algorithm (e.g. [11]) due to the moderate run-time requirements. For example, the long-range link addition algorithm can be invoked after a permissible mapping is obtained to see how much additional improvement can be obtained.
- The long-range links can also be exploited to achieve fault-tolerance or QoS operation. For example, the use of a long-range link can be limited to a few connections which provide guaranteed latency (or throughput) for a selected set of nodes. Furthermore, the long-range links can also support multiple use-case scenarios [15]. For example, a long-range link can provide guaranteed service for some use cases, while serving the best-effort traffic for improved performance during other use-cases.
- The improvement in network performance can be exploited to optimize the system power consumption. For example, the operating voltage can be scaled down while still achieving the same throughput of a pure mesh network of exact same size. As a result, the overall power consumption can be minimized.

[3] More details about the simulator and the prototype are given in Appendix A.2 and A.3, respectively.

Finally, the long-range links can be considered in a reconfigurable context to obtain a common architecture which can be optimized for a larger class of applications.

6.8 Experimental Evaluation of Long-Range Link Insertion Methodology

The effectiveness of partial topology customization via the long-range link insertion is demonstrated through an extensive experimental study involving synthetic and real traffic patterns. For each benchmark, the standard mesh network and the mesh network with a small number of long-range links inserted to it are compared against each other. Sections 6.8.1–6.8.4 present simulation results obtained using the cycle-accurate *worm_sim* NoC simulator [29]. The simulator models the long-range links as explained in Sect. 6.5. In Appendix A.4, we present our results obtained using an FPGA prototype.

The *worst-case complexity* of the technique (that is, the link insertion and the routing table generation) is $O(SN^{\alpha})$, with $2 < \alpha < 3$. The run-time of the algorithm for the examples we analyzed ranges from 0.14 s for a 4 × 4 network, to less than half hour for an 8 × 8 network, on a Pentium III machine with 768 MB memory under Linux OS.

6.8.1 Evaluation Using Synthetic Benchmarks

We first demonstrate the effectiveness of adding long-range links to standard mesh networks by using the *hotspot* and *transpose* traffic inputs. For *hotspot* traffic, three nodes[4] are selected randomly to act as *hotspot* nodes. Each node in the network sends packets to these *hotspot* nodes with a higher probability compared to the remaining nodes. For *transpose* traffic, on the other hand, each node communicates only with the symmetric node with respect to the diagonal of the network. The critical traffic load values for some 4 × 4 and 6 × 6 mesh (λ_{Mc}) and customized (λ_{Lc}) networks, under *hotspot* and *transpose* traffic patterns, are given in Table 6.3. We observe that inserting 4 long-range links to a 4 × 4 network (the resulting network is shown in Fig. 6.1) under *hotspot* traffic makes the phase transition region shift from 0.41 *packet/cycle* to 0.50 *packet/cycle*. Similarly, due to the addition of long-range links, the average packet latency at 0.41 *packet/cycle* injection rate drops from 196.9 to 34.4 *cycles*!

The variation of the network throughput and average packet latency as a function of traffic injection rate are plotted for hotspot and transpose traffic in Figs. 6.10 and 6.11, respectively. For the *transpose* traffic, the phase transition region shifts from a throughput 0.35 *packet/cycle* to 0.52 *packet/cycle*. Given the

[4] Throughout the chapter, the IDs of the hotspot nodes are: 5, 11, 12. For example, for the 4 × 4 network this translates to (1,1), (3,2), (0,3), as shown in Fig. 6.5.

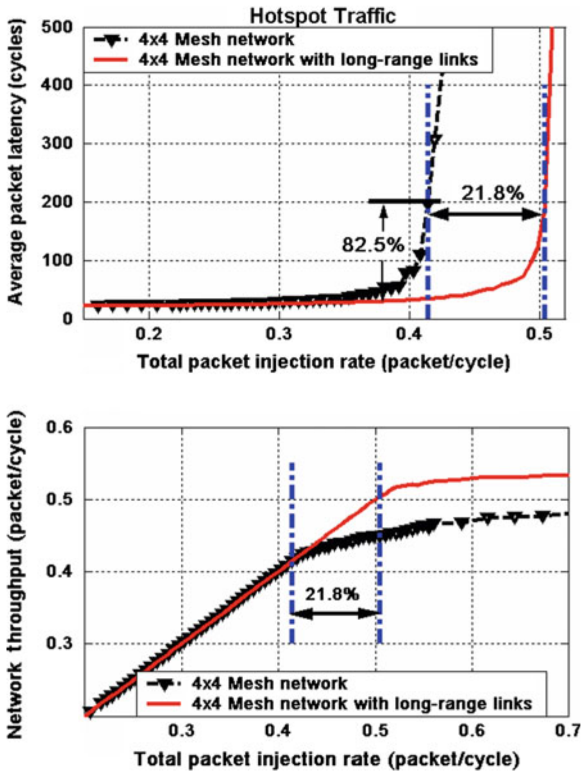


Fig. 6.10 Traffic injection rate versus average packet latency and network throughput for hotspot traffic. The improvement in the critical point and latency values at criticality are indicated on the plots

changes in the overall network dynamics; this is a huge improvement in network capabilities. Likewise, with the addition of long-range links, the average packet latency at 0.35 *packet/cycle* injection rate drops from 89.3 to 25.2 *cycles*.

It is interesting to observe that the improvement obtained for the 6 × 6 network in Table 6.3, under the *transpose* traffic, is smaller compared to other cases. The primary reason for this behavior is that some of the newly added long-range links act in fact as traffic attractors. Since the proposed routing algorithm is not adaptive in nature, i.e. it does not consider congestion in the channel links at run time, these links eventually become bottlenecks in the network. This observation suggests that developing an adaptive algorithm, at the expense of increased resources, has the potential to improve the impact of long-range link insertion algorithm even beyond the results presented here.

6.8.2 Scalability Analysis

To evaluate the *scalability* of the proposed technique, we performed several experiments with network sizes ranging from 4 × 4 to 10 × 10. For the 4 × 4

Fig. 6.11 Traffic injection rate versus average packet latency and network throughput for transpose traffic

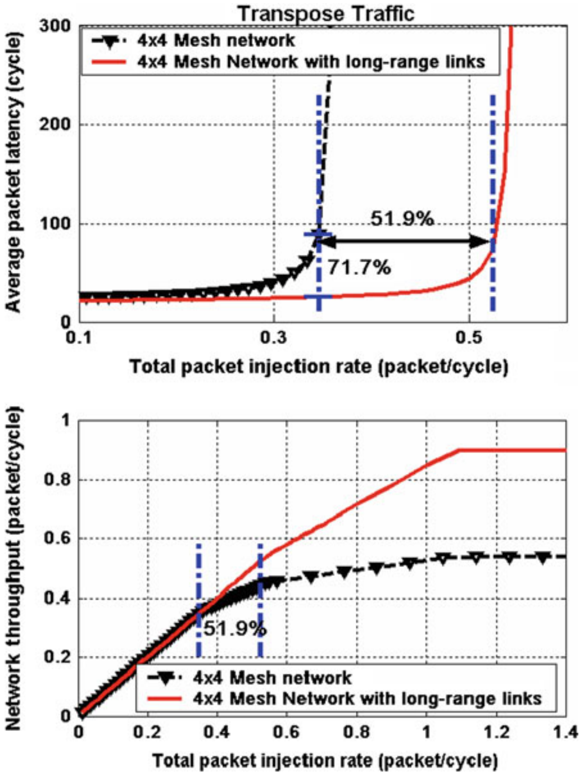
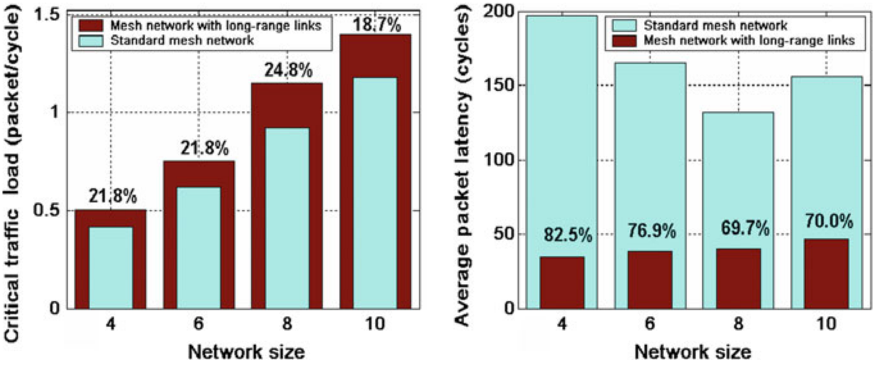


Table 6.3 Critical load (*packet/cycle*) and latency (*cycle*) comparison for regular mesh and mesh with long-range links

	Critical load (packet/cycle)		Latency at the critical load(cycle)	
	λ_{Mc}	λ_{Lc}	$L_M(\lambda_{Mc})$	$L_L(\lambda_{Mc})$
hotspot4	0.41	0.50	196.9	34.4
hotspot6	0.62	0.75	224.5	38.2
transpose4	0.35	0.52	89.3	25.2
transpose6	0.54	0.55	165.9	116.7

λ_{Mc} Critical traffic load value for pure mesh network
λ_{Lc} Critical traffic load value after inserting the long-range (LR) links
$L_M(\lambda_{Mc})$ Average latency of the pure mesh network at traffic load λ_{Mc}
$L_L(\lambda_{Mc})$ Average latency at traffic load λ_{Mc} after inserting long-range links

and 6 × 6 networks, 4 bidirectional links are inserted to the standard mesh configuration. Similarly, for the 8 × 8 and 10 × 10 networks, 5 and 6 bidirectional links are inserted, respectively. Figure 6.12 shows that the proposed technique results in consistent improvements when the network size scales up. For example, the critical load of a 10 × 10 network, under *hotspot* traffic, shifts from 1.18



(a) (b)

Fig. 6.12 The improvement in the **a** critical traffic load and **b** average packet latency for increasing network sizes

packet/cycle to 1.40 *packet/cycle* after inserting only 6 bidirectional long-range links consisting of 32 regular bidirectional links total. This result is similar to the improvements obtained for smaller networks. Figure 6.12a also reveals that the critical traffic load grows with the network size due to the increase in the total bandwidth. Likewise, we note substantial reductions in the average packet latency across different network sizes after inserting long-range links, as shown in Fig. 6.12b.

6.8.3 Comparison with Topologies of Higher Dimensionality

Finally, we note that customizing 2-D mesh networks with long-range links is a more general approach than choosing a tori or other network topologies of higher dimensionality, or simply inserting express links based on a fixed rule [4, 14]. Indeed, the on-chip implementation of these networks looks similar to the implementation of a 2D mesh network with application-specific long-range links except for a fundamental difference. The latter finds the optimal links to be inserted based on a rigorous *analysis*, rather than by following a fixed wiring rule. In fact, the application-specific customization with long-range links will generate standard higher dimensional networks (or reduce to inserting by-pass links), if we replace the link insertion algorithm with a *static* link insertion rule. As a result, our technique is more general and can achieve a better performance compared to a higher dimensional network, although it utilizes about the same or less resources.

To be more concrete, we implemented a 4 × 4 2D torus network with folded links [6], and a mesh network with 8 uni-directional links generated by our technique for a resource constraint threshold of $S = 12$. Our simulations show that the critical traffic load of the network customized using our proposed technique is

4 % larger than that of the torus network. Moreover, the average packet latency in our design, at 0.48 packet/cycle injection rate (which is close the critical load of the torus network), is only 34.4 cycles compared to 77.0 cycles for the torus network. This significant gain is obtained by utilizing only half of the extra links needed by torus; indeed, inserting the most beneficial links for a given traffic pattern makes more sense than blindly adding channels following a fixed rules, which is precisely the case for the folded torus.

6.8.4 Experiments Involving Real Traffic

In this section, we evaluate the performance of the link insertion algorithm using three realistic applications. The first two applications, *auto industry* and *telecom* benchmarks, are retrieved from E3S benchmark suite [7] and mapped onto 4 × 4 and 5 × 5 networks, respectively, using an automated mapping tool. The third application is a multimedia application (*MMS*) which includes an H263 video encoder/decoder, an MP3 audio encoder/decoder pairs. This application is first partitioned into 40 concurrent tasks and then assigned and scheduled onto 16 IPs connected in a 4 × 4 mesh network [11]. The long-range links are inserted with a constraint of 12 and 20 for the 4 × 4 and 5 × 5 network, respectively.

The variation of average packet latency and network throughput as a function of traffic injection rates for the *auto industry* benchmark is given in Fig. 6.13. These plots show that the insertion of long-range links shifts the critical traffic load from 0.29 *packet/cycle* to 0.33 *packet/cycle* (about 13.6 % improvement). Similarly, the average packet latency for the network with long-range links is consistently smaller compared to that of a pure mesh network. For instance, at 0.29 *packet/cycle* injection rate, the latency drops from 98.0 *cycles* to 30.3 *cycles* giving about 69.0 % reduction.

Similar improvements have been observed for the *telecom* benchmark, as shown in Fig. 6.14. Specifically, the critical traffic load is improved from 0.44 *packet/cycle* to 0.60 *packet/cycle* showing a 36.3 % increase due to the insertion of long-range links. Likewise, the latency at 0.44 *packet/cycle* traffic injection rate drops from 73.1 *cycles* to 28.2 *cycles*. Finally, a pure 4 × 4 mesh network running the *MMS* application has a critical traffic load of 0.26 *packets/cycle*, while the network customized using application-specific long-range links has a traffic load of 0.29 *packets/cycle*. Moreover, the average packet latencies at 0.26 *packets/cycle* are 96.0 and 31.0 *cycles* for the original mesh and customized networks, respectively.

Implementing long-range links requires inserting buffers in the repeaters. In order to demonstrate that the savings are primarily coming from using the long-range links, we also added extra buffers to the channels of the pure mesh network with inserted links, equal to the amount of buffers utilized for the long-range links.

Table 6.4 summarizes the results for standard mesh network (*M*), standard mesh network with extra buffers (*MB*), and the network with long-range links (*L*).

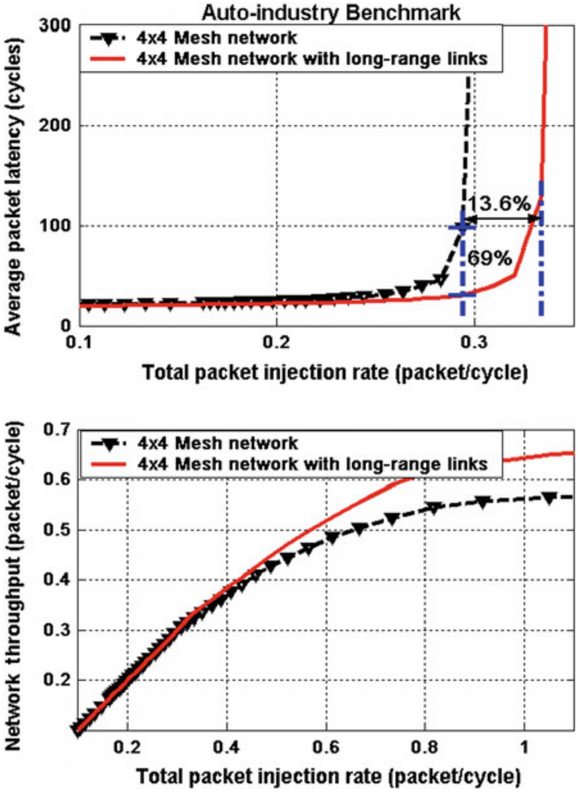


Fig. 6.13 Traffic injection rate versus average packet latency and network throughput for auto industry benchmark

We observe that the buffers insertion improves the critical load by 3.5 % for the *auto industry* benchmark. On the other hand, the corresponding improvement due to long-range links is 13.6 % over initial mesh network, and 10 % over the mesh network with additional buffers. Likewise, we note that due to inserting long-range links, the average packet latency reduces by 69 % compared to the original latency value and 57.0 % compared to the mesh network with extra buffers.

Consistent results have been obtained for the synthetic traffic workloads mentioned in the previous section and for the *telecom* benchmark. We report here only the results for *telecom* benchmark since this reflects a real application. The results in Table 6.4 show that, with the addition of extra buffers, the critical traffic point shifts only from 0.44 *packet/cycle* to 0.46 *packet/cycle*. Inserting long-range links, on the other hand, shifts the critical point to 0.60 *packet/cycle* which represents a huge improvement in the network throughput capability. Similarly, the average packet latency obtained by the proposed technique is almost 1/3 of the latency provided by standard mesh and about 1/2 of the latency provided by mesh with extra buffers.

Fig. 6.14 Traffic injection rate versus average packet latency and network throughput for telecom benchmark

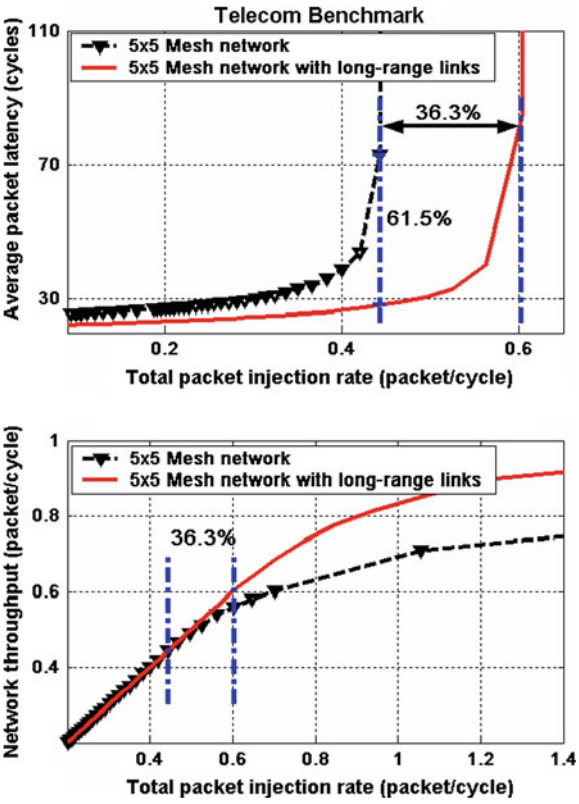


Table 6.4 Critical load (*packet/cycle*) and latency (*cycles*) comparison for pure mesh (M), mesh with extra buffers (MB) and mesh with long links (L)

	Critical load (packet/cycle)	Latency at the critical load (cycles)
auto-indust M	0.29	98.0
auto-indust MB	0.30	70.5
auto-indust L	0.33	30.3
telecom M	0.44	73.1
telecom MB	0.46	56.0
telecom L	0.60	28.2

6.8.5 One Architecture for All

Optimizing the NoC communication architecture for each target application may be too costly, if the product volume is not sufficiently large to justify the optimization effort. Therefore, it is desirable to leverage a single optimized architecture for a class of applications that share common communication characteristics. This can be achieved by generating a composite description of the

```
- Let AppSet be the Set of Applications we target
- Reset the communication frequency between each pair of nodes to 0, i.e., $f_{ij} = 0 \ \forall i,j \in T$
- For each application in the Target Set of Applications AppSet ($k \in AppSet$):
  Begin
    - Read in the application k;
    - Let $f_{ij}(k)$ denote the communication frequencies corresponding to application k;
    - Let the weight corresponding to application k be $w(k) = \sum_{(i,j)} f_{i,j}$;
  End
- For each application in the Target Set of Applications AppSet ($k \in AppSet$):
  Begin
    - $f_{ij} = f_{ij} + w(k) \times f_{ij}(k)$;
  End
```

Fig. 6.15 Pseudo-code for generating the composite description for the target applications

target applications and feeding it to the long-range link insertion algorithm summarized in Fig. 6.3.

To this end, we derive the communication frequencies each node in the network ($i.e., f_{ij}\ \forall i, j \epsilon T$, see Sect. 6.3.2 for the definition) using the pseudo-code given in Fig. 6.15. Basically, we first generate the communication frequencies f_{ij} corresponding to each application in the target set. Then, the sum of the communication frequencies is computed as a normalization factor. Once, we obtain the descriptions of the individual applications, we derive the composite description as the weighted sum of the individual applications, where the weights are simply the normalization factors computed before. These weights ensure that each application in the target set is equally represented. If we prefer to favor a subset of applications (e.g. when their production volume is larger or they are more critical in terms of performance), this can be achieved easily by manipulating these weights.

Once the composite description is available, we use it as the input to the long-range link insertion algorithm and the architecture that is optimized for the target *class of applications* is obtained. Intuitively, the long-range links will have a positive impact on the performance for any application. In the worst case, which happens when the long-range links are not utilized at all, it would not result in any improvement. Our goal, however, is to quantify the performance improvement obtained for each application running on the common architecture. To achieve this goal, we used the *uniform, hotspot* traffic patterns studied in Sect. 6.8.1, the *auto-industry* benchmark studied in Sect. 6.8.4 and *MMS* application studied in Sect. 6.3.5. These examples are chosen, since all of them run on 4 × 4 NoCs. Then, the following simulations are performed for this experiment:

- Each application is simulated on 4 × 4 pure mesh network;
- Each application is simulated on an NoC optimized particularly for that application;
- Each application is simulated on the NoC optimized for the composite description, i.e., on the common architecture.

Table 6.5 Comparison of the critical traffic load (*packet/cycle*) for regular mesh (column labeled as *Mesh*), architecture optimized for a single architecture (column labeled as *Optimized Architecture*) and the common architecture obtained for all applications (column labeled as *Common Architecture*)

	Critical load (packet/cycle)		
	Mesh	Optimized architecture	Common architecture
uniform	0.68	0.71	0.72
hotspot	0.41	0.50	0.47
MMS	0.25	0.284	0.282
auto-industry	0.29	0.33	0.30

The critical traffic loads obtained for each of these cases are summarized in Table 6.5. We make two key observations based on these results. First, for the *uniform*, *hotspot* and *MMS* benchmarks the common architecture provides performance improvement *comparable* to that of obtained with the architecture optimized for a single application. This stems mainly from the (*un-intentional*) similarity between the communication requirements between these benchmarks. For example, more detailed analysis shows that *three* of the *four* long-range links in the common architecture also exist in the architecture optimized solely for the *MMS* application. Our second observation is the poor improvement obtained for the *auto-industry* benchmark. Again, more detailed analysis shows that this benchmark has considerably more skewed traffic pattern compared to other benchmarks. Consequently, only *one* of the *four* long-range links in the common architecture exists in the architecture optimized only for this benchmark. Since this application does not benefit from the remaining links, we observe limited performance improvement for this benchmark compared to other benchmarks.

In summary, we observe that significant performance improvements can be obtained when using a single optimized architecture for a class of applications. However, care must be taken when selecting the set of applications, as only applications with similar communication requirements can benefit from a common set of long-range links, as expected.

6.9 Summary

In this chapter, we have presented a novel design methodology for inserting application-specific long-range links to standard mesh NoC architecture. It has been analytically and experimentally demonstrated that additional long-range links can increase significantly the critical traffic workload. We have also demonstrated that this increase brings a significant reduction in the average packet latency of the network, as well as substantial improvements in the achievable throughput. The experimental results obtained using an FPGA prototype (see Appendix A.4) support the findings reported in this chapter.

References

1. Carloni LP, McMillan KL, Sangiovanni-Vincentelli AL (2001) Theory of latency-insensitive design. IEEE Trans Comput-Aided Des Integr Circ Syst 20(9):1059–1076
2. Chandra V, Xu A, Schmit H, Pileggi L (2004) An interconnect channel design methodology for high performance integrated circuits. In: Proceedings of design, automation and test in Europe conference, Feb 2004
3. Connor IO, Gaffiot F (2004) Advanced research in on-chip optical interconnects. In: Piguet C (ed) Lower power electronics and design. CRC Press
4. Dally WJ (1991) Express cubes: improving the performance of k-ary n-cube interconnection networks. IEEE Trans Comput 40(9):1016–1023
5. Dally WJ, Seitz CL (1987) Deadlock-free message routing in multiprocessor interconnection networks. IEEE Trans Comput 36(5):547–553
6. Dally WJ, Towles B (2001) Route packets, not wires: on-chip interconnection networks. In: Proceedings of design automation conference, June 2001
7. Dick R Embedded system synthesis benchmarks suites (E3S). http://ziyang.eecs.umich.edu/∼dickrp/e3s/
8. Duato J, Yalamanchili S, Ni L (2002) Interconnection networks: an engineering approach. Morgan Kaufmann, San Mateo
9. Fuks H, Lawniczak A (1999) Performance of data networks with random links. Math Comput Simul 51(1–2):101–117
10. Glass CJ, Ni LM (1992) The turn model for adaptive routing. In: Proceedings of ISCA, May 1992
11. Hu J, Marculescu R (2005) Energy- and performance-aware mapping for regular NoC architectures. IEEE Trans Comput-Aided Des Integr Circ Syst 24(4):551–562
12. Kleinberg J (Aug. 2000) Navigation in a small world. Nature 406:845
13. Kumar A, Peh L, Kundu P, Jha NK (2007) Express virtual channels: towards the ideal interconnection fabric. In: Proceedings of the international symposium on computer architecture, June 2007
14. Loucif S, Ould-Khaoua M, Mackenzie LM (1999) On the performance merits of bypass channels in hypermeshes and k-ary n-cubes. Comput J 42(1):62–72
15. Murali S, Coenen M, Radulescu A, Goossens K, De Micheli G (2006) A methodology for mapping multiple use-cases onto networks on chips. In: Proceedings of design automation and test in Europe conference, March 2006
16. Newman MEJ, Watts DJ (1999) Scaling and percolation in the small-world network model. Phys Rev E 60:7332–7342
17. Newman M (2003) The structure and function of complex networks. SIAM Rev 45(2):167–256
18. Nigussie E, Lehtonen T, Tuuna S, Plosila T, Isoaho J (2007) High-performance long NoC link using delay-insensitive current-mode signaling, Hindawi VLSI design, Special Issue on Networks-on-Chip, vol. 2007, March 2007
19. Ogras UY, Marculescu R (2005) Energy- and performance-driven NoC communication architecture synthesis using a decomposition approach. In: Proceedings of design, automation and test in Europe conference, March 2005
20. Ogras UY, Marculescu R, Lee HG, Chang N (2006) Communication architecture optimization: making the shortest path shorter in regular networks-on-chip. In: Proceedings of design automation and test in Europe conference, March 2006
21. Ohira T, Sawatari R (1998) Phase transition in computer network traffic. Phys Rev E 58:193–195
22. Pamunuwa D, Öberg J, Zheng LR, Millberg M, Jantsch A, Tenhunen H (2003) Layout, performance and power trade-offs in mesh-based network-on-chip architectures. In: IFIP international conference on very large scale integration, Dec 2003

23. Pinto A, Carloni LP, Sangiovanni-Vincentelli AL (2003) Efficient synthesis of networks on chip. In: Proceedings of international conference on computer design, Oct 2003
24. Shacham A, Bergman K, Carloni LP (2007) The case for low-power photonic networks-on-chip. In: Proceedings of design automation conference, June 2007
25. Srinivasan K, Chatha KS, Konjevod G (2006) Linear programming based techniques for synthesis of network-on-chip architectures. IEEE Trans Very Large Scale Integr Syst 14(4):407–420
26. Wang H, Zhu X, Peh L, Malik S (2002) Orion: a power-performance simulator for interconnection networks. In: Proceedings of annual international symposium on microarchitecture, Nov 2002
27. Watts DJ (1999) Small Worlds: the dynamics of networks between order and randomness. Princeton University Press, Princeton
28. Watts DJ, Strogatz SH (1998) Collective dynamics of small-world networks. Nature 393:440–442
29. Worm_Sim: a cycle accurate simulator for Networks-on-Chip. http://www.ece.cmu.edu/∼sld/wiki/doku.php?id=shared:wormsim
30. Woolf M, Arrowsmith DK, Mondragon-C RJ, Pitts JM (2002) Optimization and phase transitions in a chaotic model of data traffic. Phys Rev E 66:046106
31. Ye T, Benini L, De Micheli G (2002) Analysis of power consumption on switch fabrics in network routers. In: Proceedings of design automation conference, June 2002

Chapter 7
Analysis and Optimization of Prediction-Based Flow Control in Networks-on-Chip

While networks-on-Chip (NoC) architectures may offer higher bandwidth compared to traditional bus-based communication, their performance can degrade significantly in the absence of effective flow control algorithms. This chapter presents a predictive closed-loop flow control mechanism, which is used to predict the congestion level in the network. Based on this information, the proposed scheme controls the packet injection rate at traffic sources in order to regulate the total number of packets in the network. Finally, simulations and experimental study using our FPGA prototype show that the proposed controller delivers a better performance compared to the traditional switch-to-switch flow control algorithms under various real and synthetic traffic patterns.

7.1 Introduction

While the NoC architectures offer substantial bandwidth increase and concurrent communication capability, their performance can significantly degrade in absence of an effective flow control mechanism. Flow control algorithms avoid resource starvation and congestion in the network by regulating the flow of the packets competing for shared resources, such as links and buffers [3, 5].

In the NoC domain, the term flow control is used almost exclusively in the context of switch-to-switch [7, 13, 15, 21, 31] or end-to-end [27] transport protocols. These protocols provide a smooth traffic flow by avoiding buffer overflow and packet drops. However, the flow control can also regulate the packet population in the network by restricting the packet injection to the network [3].[1] This is precisely the main objective of this chapter.

Switch-to-switch flow control algorithms, such as ON/OFF, credit-based and ACK/NACK mechanisms, regulate the traffic flow *locally* by exchanging control information between the neighboring routers. These approaches have a small

[1] This function is also referred as congestion control. However, following the convention in [3] and [9], we do not make such a distinction.

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1_7,
© Springer Science+Business Media New York 2013

communication overhead, since they do not require explicit communication between source/sink pairs. However, the switch-to-switch flow control does not regulate the actual packet injection rate directly at the traffic source level but instead, it relies on a backpressure mechanism which propagates the availability of the buffers in the downstream routers to the traffic sources. Consequently, before the congestion information gets the chance to reach the traffic sources, the packets generated in the meantime can seriously congest the network. Moreover, wormhole routing is prone to head of line (HOL) blocking which is a significant performance limiting factor. HOL blocking happens when the packet header cannot propagate to the next router due to lack of buffering space. When the HOL blocking occurs, all subsequent packets remain blocked and thus the router output ports can starve. Therefore, congestion becomes even more severe for networks that employ wormhole routing [6, 28].

End-to-end flow control algorithms, on the other hand, try to conserve the number of packets in the network by regulating the packet injection rate right at the source of messages. For example, in window-based algorithms, a traffic source can only send a limited number of packets before the previously sent packets are removed from the network. However, the major drawback of end-to-end control algorithms is the large overhead incurred when sending the feedback information [3]. Besides this, the unpredictable delay in the feedback loop can cause unstable behavior as the link capacities increase [22].

7.2 Overall Approach

In this chapter, we present a predictive flow control algorithm which enjoys the simplicity of the switch-to-switch algorithms, while directly controlling the traffic sources, very much like the end-to-end algorithms. Towards this end, we first present an ON/OFF traffic source model. During the ON state, the traffic sources generate packets in a bursty manner until the entire message gets transmitted. During the OFF state, on the other hand, the sources are silent, i.e. they either process data or wait for new inputs. The knowledge of the target application enables us to characterize the distribution of the ON state.

Next, we develop a novel router model based on state space representation, where the state of a router is given by the amount of flits already stored at the input buffers. Using the traffic source and router models, each router in the network predicts the *availability* of its input buffers in a *k-step* ahead of time manner. These availability values are computed via an aggregation process using the current state of the router, the packets currently processed by the router, and the availability of the immediate neighbors.

Since all predictions are based on data the routers receive directly from their immediate neighbors, the computations are decentralized and no global data exchange is required. Moreover, we note that the availability information computed at time n is obtained by aggregating the availability of the immediate neighbors at time $n - 1$. This information, in turn, reflects the state of the routers situated two hops away, at time $n - 2$, and so on so forth. Therefore, due to the aggregation process the *local* predictions actually reflect the global view of the network.

Finally, the traffic sources utilize the availability of the local router to control the packet generation process and avoid excessive injection of packets in the network.

7.3 Related Work

From a flow control perspective [5, 9], most of the work presented in the NoC domain relies on the switch-to-switch flow control; this is primarily due to the large overhead incurred by the end-to-end flow control algorithms. A comparison of the fault-tolerance overhead of various flow control algorithms employed in NoCs can be found in [25]. In that paper, the authors consider buffer and channel bandwidth allocation in presence of pipelined switch-to-switch links and analyze varying degrees of fault tolerance support, resulting in different area and power trade-offs.

We note that, in real applications, the *best-effort* (or non-real time) and *guaranteed service* (or real time) traffic may coexist. The Aethereal network architecture presented in [27] employs the end-to-end flow control for guaranteed service in addition to the basic link-level control. Similarly, the SPIN architecture in [1] also uses credit-based flow control where buffer overflows at the target end of a path are checked at the source.

The work in [10] also provides guaranteed services on top of best-effort traffic using prioritization of flows. A quantitative comparison between this connection-less scheme and a connection-oriented scheme is presented in [11]. The authors conclude that the connection-less scheme offers more stable end-to-end delay and it is able to provide guaranteed latency for individual flows.

Nostrum NoC architecture presented in [19] has two dimensional mesh topology and employs an adaptive, deflective routing. In deflecting routing, the incoming packet is routed to one of the free output channels belonging to a minimal path. If all the channels belonging to minimal paths are occupied, then the packet is misrouted. This increases message latency even in the absence of congestion and bandwidth consumption [2, 8, 14]. Moreover, when there are no available output channels, the entire packet needs to be stored; this requires buffers large enough to store the packets. Nostrum deals with this by fixing the packet size to 1-flit. However, this requires putting the header information such as destination address to each packet. Hence, this results in a large overhead and poor bandwidth utilization.

Unlike the Nostrum architecture, our approach supports packets with arbitrary length. We employ wormhole routing and deterministic shortest path routing algorithms. Finally, Nostrum handles both best-effort and guaranteed latency traffic. The guaranteed service is provided through virtual circuits implemented using looped containers and temporally disjoint network concepts which require a synchronous design (i.e. a common sense of time across the network). As opposed to this, our proposed technique targets *best-effort* traffic. Hence, our technique cannot be used to provide guaranteed services per se. However, when a mechanism for the guaranteed service traffic is in place, the proposed technique can be used in conjunction with this service to fully exploit the bandwidth not utilized by the guaranteed service traffic.

Congestion control is well studied for classical networks [3, 9, 22, 26]. For instance, the authors of [22] develop a decentralized control system, where the sources adjust their traffic generation rates based on the feedback received from the bottleneck links. A predictive explicit-rate control mechanism is presented in [26] where the authors consider a single bottleneck node and infinite buffering resources. The sources adjust their traffic rates using the congestion information received from the bottleneck node via control packets.

Injection limitation techniques are studied in the context of parallel computer networks to avoid network saturation and cope with deadlock. The authors of [17] use the number of busy output channels in a node as a measure of level of congestion. If the number of busy output channels exceeds a properly selected threshold value, then the router prevents injection of new messages. Since this threshold is a function of the traffic pattern and packet sizes, the authors adjust it dynamically as a function of network load. The authors in [2] survey a family of mechanisms for congestion control in wormhole networks. In the first technique, congestion is measured as the ratio between the number of free virtual channels and total number of useful virtual channel that could be used by a certain message. If this ratio is larger than a threshold which should be tuned manually, then the packet is injected to the network. In a second technique, packet injection is permitted if all physical channels have at least one virtual channel free or at least one physical channel has all its virtual channels free. Finally, the third method computes the number of flits sent through each virtual channel in a certain time interval to detect network congestion. If a channel is busy and the number of flits sent is less than a threshold, then the channel is considered congested. In case congestion is detected, packet injection restrictions are applied at the local node. The time interval and threshold need to be tuned, as in the first mechanism.

These techniques rely on local feedback, hence they lack knowledge about global information. On the other hand, the authors in [28] present a global congestion control scheme based on time-outs. In this scheme, each node monitors the time the header flit stays in the source queue. If the waiting time is larger than a threshold, the node sends a congestion signal to its neighbors. All the nodes receiving the congestion message limit packet injection and share this information with their own neighbors. The technique presented in [29] aims at detecting congestion in early stages by taking the global conditions into account. The fraction of full virtual channel buffers of all routers is used as the congestion metric. The congestion data collected at each node is then disseminated to all other nodes through an exclusive side-band reserved for this purpose. The authors develop an all-to-all communication mechanism for dissemination of congestion information with guaranteed delay bounds. However, this mechanism is specific to the particular network topology and its generalization to other topologies is not straightforward [12].

The approach we present in this chapter is different from previous work in a number of ways. First, our technique is computationally light since it relies on *local* data transfers, similar to the basic switch-to-switch flow control. At the same time, our mathematical formulation enables us to predict the available buffering space in the network without assuming any particular traffic pattern or network topology. Due to the aggregation process performed at the routers, the information exchanged between the switches actually reflects the global view of the network. Furthermore, since the predictions reflect the state of the network k steps ahead in time, the packet sources across the network can sense a possible congestion situation early on and then adapt in order to avoid excessive packets injection to the network.

7.4 System and Traffic Source Modeling

7.4.1 System Model and Basic Assumptions

We assume the network nodes consist of processing elements (referred to as PEs) and routers which allow nodes to communicate by exchanging packets across the network. We consider wormhole routing so the packets are divided into flits. The length of a packet (S) is measured by the number of flits it contains. For convenience, the flit size is assumed to be equal to the physical channel width (W). No assumption is made about the underlying network topology.

In order to avoid packet loss, a basic link-level ON-OFF flow control mechanism is implemented in the routers [5]. The proposed predictive control technique works together with this link-level mechanism to control directly the behavior of the traffic sources.

7.4.2 Traffic Source Model

Traffic injection rate into the network is the main knob for source control. Therefore, an accurate model of the input traffic is necessary for the flow controller. Such a model will not only show how the input traffic can be handled, but also describe its impact on the packet delay in the network. Towards this end, we observe that the NoC nodes can be in two different states:

OFF STATE: The PE is either processing data or waiting for new data. While in this state, the PE does not generate traffic (hence the name OFF) as shown in Fig. 7.2.

ON STATE: The PE injects packets to the network so the traffic source and its corresponding state are referred to as ON. In this state, the source injects packets in a bursty manner until the message is completely transmitted.

7.4.2.1 A. Experimental Justification for the ON/OFF Traffic Model

To support this observation with measured data, we collected traces from the sources in the MPEG-2 encoder design presented in [16]. The MPEG-2 encoder data flow graph and its NoC-based implementation is depicted in Fig. 7.1. Figure 7.2a shows the traffic generated by the *Frame Buffer* module which stores the frames reconstructed using the previously encoded frames. The *Frame Buffer* module alternates between ON and OFF states. During the OFF state, it waits for a read request from the *Motion Compensation* module, as shown in Fig. 7.1a. Hence, no packet is generated during this period. Once a read request is received, the *Frame Buffer* module prepares a packet containing the next macroblock to be processed and injects the packet to the network; hence the module switches to the ON state and stays in the that state until all the requested data is transmitted to the *Motion Compensation* module.

In Fig. 7.2b, we investigate the traffic generated by the *Inverse Discrete Cosine Transform* (*IDCT*) */ Inverse Quantization* (*IQ*) *module* at a larger time scale. This module receives macroblocks from *Discrete Cosine Transform* (*DCT*) */ Quantization* (*Q*) module and performs inverse quantization and inverse discrete cosine transform to produce the reconstructed frame.

Since encoding the *Intra* (*I*) frames does not require motion estimation and compensation, *IDCT / IQ* module receives and processes packets at a very high pace. Consequently, we observe bursty ON periods with small OFF periods in between (time scale 5×10^4–6.8×10^4 ns in Fig. 7.2b). On the other hand, encoding *Predicted* (*P*) frames is much slower in our design. For this reason, the *IDCT / IQ* module waits longer for new blocks to be processed and experience longer OFF periods.

We also note that the bursty nature of the on-chip traffic has been observed by other researchers. For instance, the traffic model considered in [20] consists of bursts (i.e., ON) and silent (i.e., OFF) periods. Similarly, the long-range dependent (LRD) behavior of on-chip multimedia traffic is demonstrated and studied in [30].

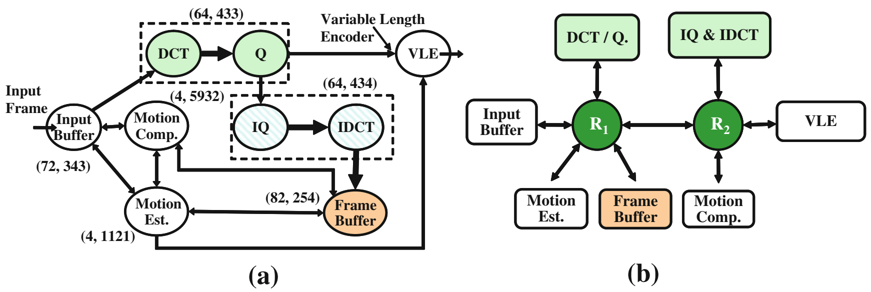


Fig. 7.1 **a** The data flow graph of the MPEG-2 encoder in [16] and **b** its NoC-based implementation are shown. The average values of ON and OFF periods ($F_{\mathrm{ON\_ave}}, F_{\mathrm{OFF\_ave}}$), in number of cycles, are also shown in **a**

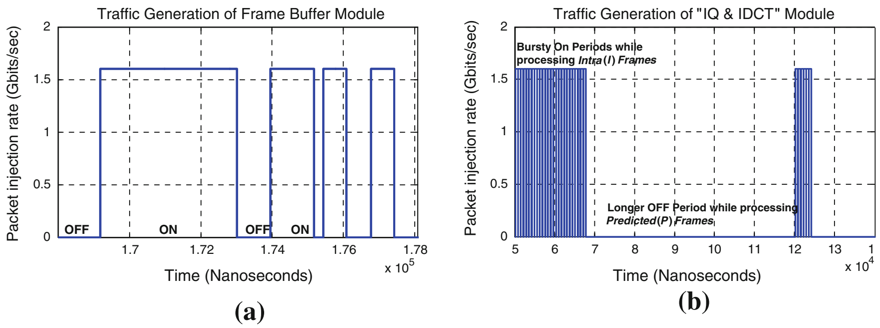


Fig. 7.2 **a** Traffic injection by the Frame Buffer module is shown. It can be observed that the module switches between ON and OFF periods. **b** Traffic injection by the Inverse Discrete Cosine Transform/Inverse Quantization module is plotted for a longer time scale. Bursty ON periods are followed by long OFF period due to long data waiting time

It is a known fact that ON-OFF traffic sources with heavy tailed distribution of ON (or OFF) times gives rise to LRD traffic [23].

7.4.2.2 B. Characterization for the Distribution of ON/OFF Periods

Let the discrete time stochastic process $\lambda(t), t \in Z^+$ denote the instantaneous flit injection rate at time *t*. The cumulative traffic volume generated up to time *t* (denoted by *V*(*t*)) is given by:

$$V(t) = V(t-1) + \lambda(t), \quad V(0) = 0, t \in Z^+ \tag{7.1}$$

In the ON state, the flit injection rate $\lambda(t)$ is constant and equal to channel width, i.e. $\lambda_{ON} = W$ bits/s. If a header flit is injected to the network at time t_0, one can see that $\lambda(t_0 + \Delta) \neq 0 \; for \; 0 < \Delta < S$, where *S* is the packet size in flits. Similarly, when the PE is in the OFF state, one can get an idea of how much longer the OFF state will continue, given the amount of time already spent for processing and type of processing done by the PE. Therefore, the inter-arrival times are *not* memoryless and so the flit injection process cannot be modelled as a Poisson process. Consequently, we need to modify the classical ON/OFF [23] model to work for NoC traffic sources.

Distribution of t_{ON}

The duration of the ON state is determined by the size of the packets generated by the node and λ_{ON}; specifically, $t_{ON} = \lceil SW/\lambda_{ON} \rceil$ where, again, *S* is the length of a packet (number of flits) and *W* is the physical channel width (*W* bits are transmitted per flit). While λ_{ON} is constant, *S* depends on the particular packet (or packets) generated by the source after completing a certain task. In an NoC, the type of the tasks performed by each PE and the size of the resulting message are typically known at design time. For example, a DSP core implementing DCT/IDCT operations in a multimedia chip, can only produce the cosine or inverse cosine transforms of a fixed size data block. Hence, S can take only certain discrete values, usually known at design time. Note that, this is in stark contrast with a general purpose network, where a node can generate a much wider range of messages. As such, we model the probability mass function F_{ON} as:

$$F_{ON}(t) = p(t_{ON} \leq t) = \sum_{i=0}^{t} p(t_{ON} = i) \tag{7.2}$$

We can actually compute $F_{ON}(t)$, since the communication volume between the network nodes and λ_{ON} are known at design time.

Distribution of $\mathbf{t_{OFF}}$

The duration of the OFF state is the sum of two random variables. The first is the processing time of the PE, t_{proc}; this can take certain discrete values, based on the number of different tasks implemented by the PE. Therefore, t_{proc} is a discrete random variable with discrete probability mass function:

$$F_{proc}(t) = p(t_{proc} \leq t) = \sum_{i=0}^{t} p(t_{proc} = i)$$

The second component of t_{OFF} is the waiting time t_{wait} for new data, before the node cannot start processing. Unlike t_{ON} and t_{proc}, the waiting time t_{wait} can take a wide range of values as it depends on the latency in the network. When t_{proc} can take n different values, the distribution of t_{OFF} can be expressed as a function of the waiting time, $p(t_{wait} \leq t)$, as follows:

$$F_{OFF}(t) = \sum_{k=1}^{n} p(t_{wait} \leq t - t_k | (t_{proc} = t_k)) P(t_{proc} = t_k) \tag{7.3}$$

In general, it is difficult to compute the distribution of t_{wait}, since it depends on the latency experienced in the network. However, the predictive flow controller presented in this chapter depends only on the distribution of the t_{ON}, as explained in Sect. 7.6.

7.4.3 Predictive Control of Traffic Sources

Suppose that the ON states of several traffic sources overlap and lead to temporary congestion in the network. Consequently, starting at time t_0, the packets generated by source i cannot be delivered to their destinations. In this scenario, source i will continue to inject packets to the network until it senses congestion, say at time $t_0 + \delta$. The number of flits injected during this time is given by:

$$V(t) = min\left(\sum_{t=t_0}^{t_0+\delta} \lambda(t), \sum B_T\right)$$

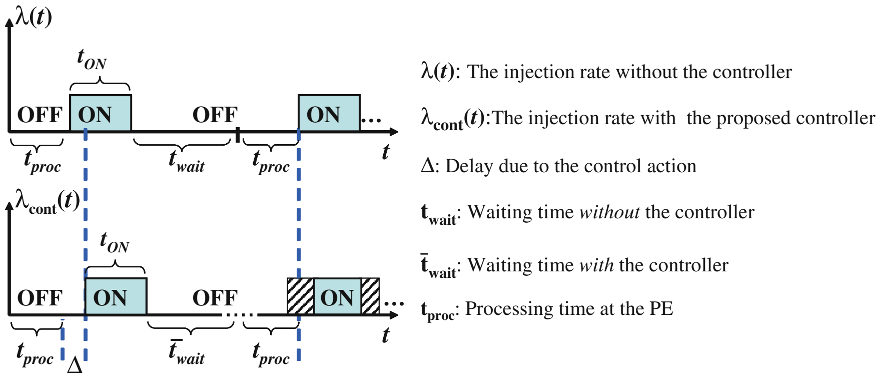

λ(t)
t_ON
OFF
ON
OFF
ON
...
t_proc
t_wait
t_proc
t
λ_cont(t)
t_ON
OFF
ON
OFF
ON
...
t_proc
Δ
t̄_wait
t_proc
t
λ(t): The injection rate without the controller
λ_cont(t): The injection rate with the proposed controller
Δ: Delay due to the control action
t_wait: Waiting time without the controller
t̄_wait: Waiting time with the controller
t_proc: Processing time at the PE

Fig. 7.3 Illustration of the ON-OFF source model and the control action. By delaying the start of the ON period, the waiting time in the network can be reduced

where the first element in the tuple represents the total number of flits that can be generated by the source, while B_T is the available buffering space along the path from source *i* to the congested router. If the interval (t_0, δ) covers the ON period of the source, it is likely that the source will continue to inject packets until it senses the backpressure effect due to the buffer starvation. This, in turn, can further increase the number of packets in the network and hence make the congestion more severe.

Since there are many sources sharing the same network resources, it is extremely important to minimize δ; this can be achieved by predicting the possible congestion before it becomes severe and propagating this information to all traffic sources. Since the buffer space availability at the routers may indicate congestion, the traffic sources can send a packet to the router only if its availability of greater than zero. Otherwise, the traffic source can *delay* the packet injection until the resource availability improves, as illustrated in Fig. 7.3.

Delaying the packet injection can effectively help regulating the total number of packets in the network, hence the average packet latency. While the precise time for packet injection is difficult (if not impossible) to find at the design time, an online predictor can guide the packet generation at the source in order to utilize the network resources in the best possible way.

7.5 State Space Modeling of NoC Routers

To obtain accurate predictions for the available buffering space at the routers, we also need a good model for the NoC router. Traditionally, the network research has been focused on directly computing the router delay [4, 24]. Unlike previous work, our goal is to predict how many flits the router can accept over the next *k* time

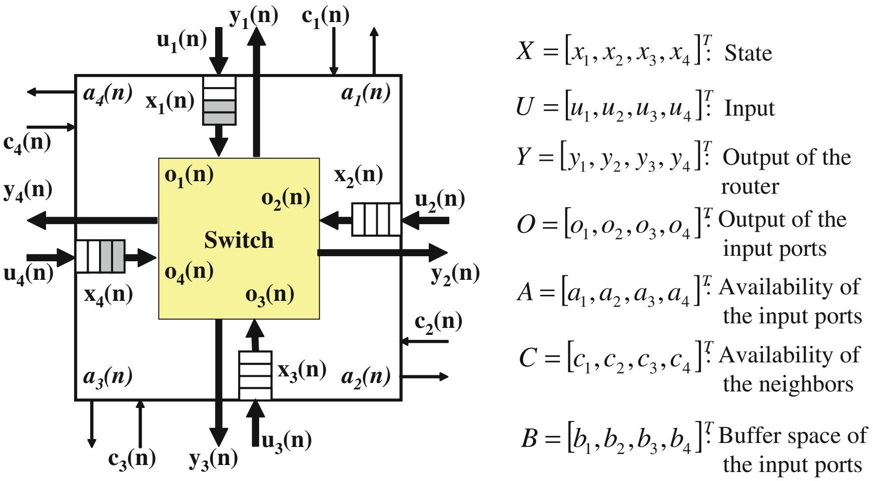

y1(n)
c1(n)
u1(n)
a4(n)
x1(n)
a1(n)
c4(n)
o1(n)
x2(n)
y4(n)
o2(n)
u2(n)
Switch
u4(n)
o4(n)
y2(n)
x4(n)
o3(n)
c2(n)
x3(n)
a3(n)
a2(n)
u3(n)
c3(n)
y3(n)
$X=[x_1,x_2,x_3,x_4]^T$: State
$U=[u_1,u_2,u_3,u_4]^T$: Input
$Y=[y_1,y_2,y_3,y_4]^T$: Output of the router
$O=[o_1,o_2,o_3,o_4]^T$: Output of the input ports
$A=[a_1,a_2,a_3,a_4]^T$: Availability of the input ports
$C=[c_1,c_2,c_3,c_4]^T$: Availability of the neighbors
$B=[b_1,b_2,b_3,b_4]^T$: Buffer space of the input ports

Fig. 7.4 The state variables, inputs and outputs of a 4-port router are shown

steps. For this reason, the parameter of interest is the *occupancy* of the router *input buffers*.[2]

We propose a state space representation of a NoC router driven by stochastic inputs, as shown in Fig. 7.4. The state of the router at time n is given by the number of flits in its input buffers; that is:

$$X(n) = [x_1(n), x_2(n), \ldots, x_P(n)]^T \tag{7.4}$$

where $x_P(n)$ is the state of the input port P (i.e., the total number of flits stored in all of the input buffers associated with port P) and 'T' denotes the transposition operation. For instance, a router with d neighboring routers and one local PE connection has $(d+1)$ ports. Hence, $X(n)$ is a $(d+1) \times 1$ vector.

The input received at port P, at time n, is denoted by $u_P(n)$. $u_P(n)$ is equal to 1, if a flit is received at time n, and is 0 otherwise. Similarly, the output from port P is represented by $y_P(n)$, where $y_P(n) = 1$ implies that a flit is transmitted to the downstream router, at time n. Consequently, the input and output processes of the router are given by the following $P \times 1$ vectors:

$$\begin{aligned} U(n) &= [u_1(n), u_2(n), \ldots, u_P(n)]^T, \\ Y(n) &= [y_1(n), y_2(n), \ldots, y_P(n)]^T \end{aligned} \tag{7.5}$$

Next, we model how the flits are read from the input buffers. $o_P(n) = 1$ means that one flit is read from the input buffer at port P, and the vector $O(n) = [o_1(n), \ldots, o_P(n)]^{(T)}$ represents the outcome of reading process from the input buffers.

[2] A similar model for the output buffers can be also developed.

Note that this is different from the outputs $Y(n)$ of the router. The output of the input buffers goes through the crossbar switch and then ends up at one of the router output ports (Fig. 7.4).

As a result, the knowledge of either $Y(n)$ or $O(n)$ provides information about the other, given the connections in the crossbar switch. So, the router can be described by an integrator, where the next state is determined by the current state, current input and current output processes, as follows:

$$X(n+1) = I_{P\times P}X(n) + U(n) - O(n) \tag{7.6}$$

Router stability

The router described by Eq. 7.6 can become unstable (i.e. the state grows unbounded), if the average arrival rate to the router is greater than the rate at which the router can serve any given packet. In practice, however, the input buffers are all finite. Hence, in order to avoid packet loss, no more flits are accepted by the link-level flow control when the buffers are full. As a result, the router model given in Eq. 7.6 can be refined as:

$$X(n+1) = I_{P\times P}X(n) + [U(n)H(n)] - O(n) \tag{7.7}$$

where $H(n) = [h(b_1 - x_1(n)), h(b_2 - x_2(n)), \ldots, h(b_P - x_P(n))]^T$. $h(x_i)$ is the unit step function (i.e. $h(x_i) = 0$ *if* $x_i \leq 0$, and $h(x_i) = 1$ otherwise), and b_1 to b_P represent the capacity of each input buffer. We also emphasize that $[U(n)H(n)]$ represents the *element-wise* product in this equation; it is used hereafter for notational simplicity.

Finally, solving Eq. 7.7 with respect to a known state $X(n_0)$, gives the state at time $n + n_0$ as

$$X(n+n_0) = X(n_0) + \sum_{j=n_0+1}^{n+n_0} ([U(j)H(j)] - O(j)) \tag{7.8}$$

Obviously, the router described by Eq. 7.8 has a bounded response. However, since such a control does not limit the source injection directly, the input buffers will remain full for most of the time, if the average arrival rate becomes larger than the service rate of the router. This, in turn, results in blocked links and large delays in the network. One could regulate the traffic injection by an open loop controller [5]. However, this solution does not solve the congestion problem completely, since the packets may experience congestion due to the overlaps between the ON periods of the traffic sources even under a light load. For instance, consider a 4 × 4 2D mesh network running hotspot traffic.[3] Although the traffic load is kept low such that the input buffers of the most congested router are empty more than 80 % of time and the buffers become full only about 1 % of time (see Fig. 7.5a), about

[3] Under the hotspot traffic, the nodes in the network receive packets with uniform probability, except a few (in our experiments 4) randomly selected nodes that receive some extra traffic.

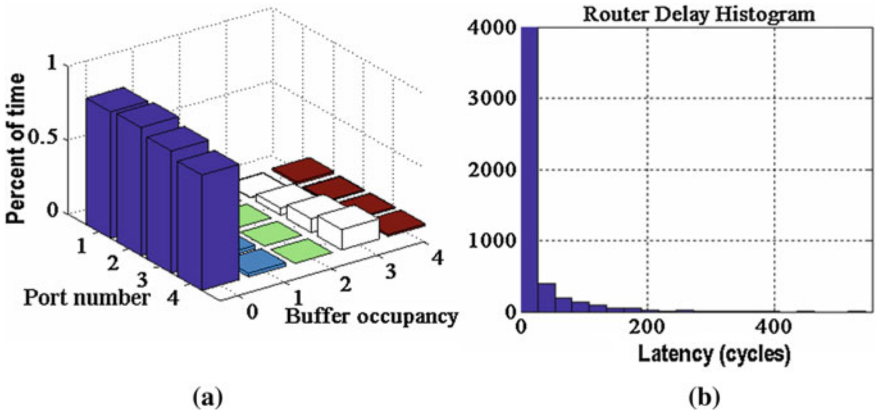


Fig. 7.5 **a** Buffer utilization and **b** delay histogram of a router

18 % of the packets experience delays more than twice as large as the average delay, as shown by the delay histogram in Fig. 7.5b. Such packets will not only block the network resources, but also affect the other packets as well. As a result, we cannot merely rely on such an *open-loop* control scheme so, in what follows, we show how exactly the router model presented in this section can be used to implement a predictive flow controller which regulates the traffic injection to the network.

7.6 Prediction-Based Flow Controller

Collecting congestion data at the routers and delivering this data to the traffic sources for flow control may cause large communication overhead; so it is not a scalable approach. Moreover, unpredictable delays in the feedback loop of flow control algorithms prevent the timely transmission of the congestion information and control signals. To mitigate this problem, we propose a *prediction-based control* which relies on the traffic source and router models developed in Sects. 7.4 and 7.5, respectively. These models enable us to predict the availability of any router at a future time step, as described next.

7.6.1 Availability Predictor

We use the conditional expectation of the state at $n_0 + k$, given the state at time n_0, i.e. $\hat{X}(n_0 + k|n_0)$, as the *k-step* predictor for network state [18]:

$$\hat{X}(n_0 + k|n_0) = E[X(n_0 + k)|X(n_0), U(n_0)]$$

Using Eq 7.8, we have:

$$\hat{X}(n_0 + k|n_0) = X(n_0) + \sum_{j=n_0+1}^{n+n_0} (E[[U(j)H(j)]|n_0] - E[O(j)|n_0]) \tag{7.9}$$

where $E[.|n_0]$ stands for $E[.|X(n_0), U(n_0)]$ (for notational simplicity). To compute the *k-step* forward prediction, we need the expected value of input and output processes, given the current state and input. If sufficient processing power is available (e.g. when the predictor is implemented in a data macro-network with plenty of resources), then Eq. 7.9 can be directly used to estimate the conditional mean values of the input and output processes to predict the state at $n_0 + k$. However, for NoCs we have to keep the area overhead as small as possible. For this reason, we use Eq. 7.9 to predict how many flits a given input port can accept, over the following k steps, rather than dealing with the absolute value of the state.

We call the number of flits the input port P can accept, over the next k steps, as the *availability* of port P and denote it by $a_P(n_0, k)$. $a_P(n_0, k)$ simply consists of the (I) sum of the number of empty slots in the buffer at time $n_0 + k$, and (II) the number of flits that are expected to be admitted in the following k steps, i.e.

$$a_p(n_0, k) = \overset{(I)}{b_p - \hat{x}_p(n_0 + k|n_0)} + \overset{(II)}{\sum_{j=n_0+1}^{n_0+k} E[u_p(j)h(b_p - x_p(j))|n_0]}$$

If we define the availability vector as $A(n_0, k) = [a_1(n_0, k), \ldots, a_P(n_0, k)]$ and $B = [b_1, b_2, \ldots, b_P,]^T$ is the vector containing the depth of each input buffer, then we can find $A(n_0, k)$ as:

$$A(n_0, k) = B - \hat{X}(n_0 + k|n_0) + \sum_{j=n_0+1}^{n_0+k} E[[U(j)H(j)]|n_0] \tag{7.10}$$

Next, we can substitute $\hat{X}(n_0 + k|n_0)$ in Eqs. 7.9–7.10 and obtain $A(n_0, k)$ as:

$$A(n_0, k) = B - X(n_0) + \sum_{j=n_0+1}^{n_0+k} E[O(j)|n_0] \tag{7.11}$$

Intuitively, $B - X(n_0)$ represents the availability at time (n_0), while the last term is the expected number of flits that will be read from the router in the interval $[n_0 + 1, n_0 + k]$. Since a new flit can be written to the buffer for each flit being read, the sum of these terms gives the availability for the interval $[n_0, n_0 + k]$.

The expected value of the read process from the input buffers (that is, the last term in Eq. 7.11) can be approximated using the router output $Y(j)$ as follows:

$$\sum_{j=n_0+1}^{n_0+k} E[O(j)|n_0] = \begin{bmatrix} g_1, 1^{(n_0)} & \cdots & g_1, p^{(n_0)} \\ g_2, 1^{(n_0)} & \cdots & g_2, p^{(n_0)} \\ \cdots & \cdots & \cdots \\ g_p, 1^{(n_0)} & \cdots & g_p, p^{(n_0)} \end{bmatrix} \sum_{j=n_0+1}^{k+n_0} E[Y(j)|n_0] \tag{7.12}$$

where the coefficients $g_{i,k}(n_0)$ reflect the state of the crossbar switch and channel allocation in the router. Computation of these coefficients are illustrated in Sect. 7.6.2 using a concrete example. If we let $G(n_0) = \{g_{i,k}(n_0)\}$, then Eq. 7.11 can be written as:

$$A(n_0, k) = B - X(n_0) + G(n_0) \sum_{j=n_0+1}^{n_0+k} E[Y(j)|n_0]$$

Note that $\sum_{j=n_0+1}^{n_0+k} E[Y(j)|n_0]$ is the expected number of flits transmitted by the router in the interval $[n_0 + 1, n_0 + k]$. However, this represents nothing but the availability of the immediate neighboring routers. In other words, instead of predicting the number of flits transmitted over the next k steps, we *aggregate* the availability information *already predicted* by the neighboring routers. As a result, the availability of a router is updated using the following equation:

$$A(n_0, k) = B - X(n_0) + G(n_0)C(n_0 - 1, k) \tag{7.13}$$

where the vector $C(n_0 - 1, k)$ denotes the availability of the immediate neighbors predicted at time $n_0 - 1$, as illustrated in Fig. 7.6 (see also Fig. 7.4).

In summary, the availability of the routers for the interval $[n_0, n_0 + k]$ are predicted using the empty buffer slots at time n_0, the state of the crossbar switch and the availability of the neighboring routers using Eq. 7.13 Hence, the computations depend on only local information. At the same time, the availability information computed at time n is obtained by aggregating the availability of the immediate neighbors at time $n - 1$. This information, in turn, reflects the state of

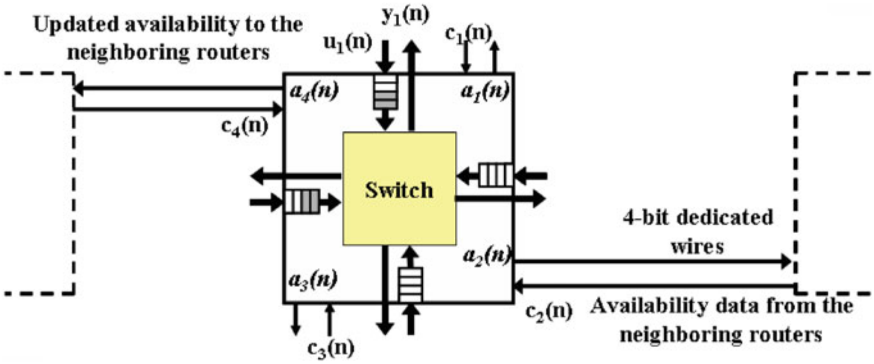


Fig. 7.6 Exchange of the availability information between the neighboring routers

the routers situated two hops away, at time $n-2$, and so on so forth. Therefore, due to the aggregation process the *local* predictions actually reflect the global view of the network.

7.6.2 Practical Implementation of the Predictor

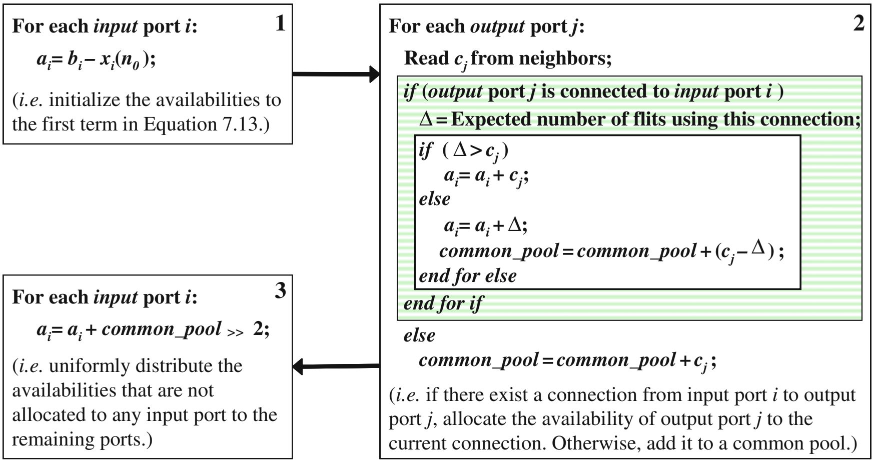
Since all the input buffers in the network are initially empty, the availability values are initialized to the sum of buffer capacities and the prediction step, i.e.

$$A(0,k) = B + k[1,1,\ldots,]^{T}_{1\times p} \tag{7.14}$$

According to Eq. 7.13, a router needs the number of free slots in its input buffers $(B - X(n_0))$, the state of the crossbar switch $(G(n_0))$, and the availabilities from the neighboring routers $(C(n_0-1,k))$. The routers keep track of the number of free slots in the input buffers and the state of the crossbar switch internally. On the other hand, they receive the availabilities of the neighboring routers through dedicated control wires, as depicted in Fig. 7.6. The number of these dedicated wires determine the maximum availability value that can be transferred between the neighboring values. For instance, in our implementation we use 4 parallel wires, as shown in Fig. 7.6; this means that we can transfer 4 bits of information over these wires. Hence, the maximum availability value that can be transferred is $2^4 - 1 = 15$.

Once a router receives the availabilities from the immediate neighbors through dedicated connections, it needs to determine how to distribute these availability values to its input ports. This distribution is achieved according to Eq. 7.13, where the coefficients $g_{i,k}$ reflect the state of the crossbar switch. The details of this distribution process are provided in Fig. 7.7.

The first step towards computing the availabilities is to initialize the availability of each input port with the number of free slots in the corresponding input buffer (i.e. the first term in Eq. 7.13), as shown by the box labeled with "1" in Fig. 7.7. We note that this term gives the zero-order approximation of the availabilities, when no input is received from the neighbors. After this initialization, the availability of the neighboring routers are processed as described in the box labelled with "2" in Fig. 7.7. For each output port j, the predictor checks whether there exist a connection through the crossbar switch between that port and any input port i. If there is a connection, then the number of flits that are expected to use this connection (Δ in box 2, in Fig. 7.7) is determined using the packet length available in the header flit and the number of flits that are already transmitted. If Δ is greater than the availability of output port j (*i.e.* c_j), then c_j flits are allocated to input port i. Otherwise, Δ flits are allocated to input port i, while the remaining (i.e. $c_j - \Delta$) is distributed uniformly to all input ports except port j. Port j is excluded, since a packet cannot leave the router using the same port it arrived, *i.e.* 180° turns are not possible due to shortest path routing. In case the output port j is not connected to



```
1
For each input port i:
    a_i = b_i − x_i(n_0);
(i.e. initialize the availabilities to the first term in Equation 7.13.)

2
For each output port j:
  Read c_j from neighbors;
  if (output port j is connected to input port i )
    Δ = Expected number of flits using this connection;
    if (Δ > c_j)
        a_i = a_i + c_j;
    else
        a_i = a_i + Δ;
        common_pool = common_pool + (c_j − Δ) ;
    end for else
  end for if
  else
    common_pool = common_pool + c_j ;
(i.e. if there exist a connection from input port i to output port j, allocate the availability of output port j to the current connection. Otherwise, add it to a common pool.)

3
For each input port i:
    a_i = a_i + common_pool >> 2;
(i.e. uniformly distribute the availabilities that are not allocated to any input port to the remaining ports.)
```

Fig. 7.7 Practical implementation of the predictor

any input port, then the whole availability c_j is distributed uniformly to all input ports except port j, as described by the outer *if* statement in box 2, in Fig. 7.7.

Example 1: Assume that a time n_0, the depth of the input buffers, their occupancies and the availability values received from the neighboring routers are given as follows:

$$B = \begin{array}{c} Local \\ North \\ West \\ South \\ East \end{array} \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \\ 4 \end{bmatrix}, X(n_0) = \begin{bmatrix} 0 \\ 4 \\ 0 \\ 0 \\ 0 \end{bmatrix}, C(n_0) = \begin{bmatrix} 0 \\ 8 \\ 3 \\ 8 \\ 8 \end{bmatrix}$$

We further assume that the crossbar switch connects the *North* input port to the *West* output port, and 4 flits in the *North* input port are waiting to traverse the crossbar switch (i.e. $\Delta = 4$).

Computation of availabilities

Next, we explain the computation of availabilities according to the pseudo-code in Fig. 7.7. This description also shows how the algorithm is implemented. We start by initializing the availabilities to $B - X(n_0)$, i.e. the number of empty slots at time $n_0 : A = [4, 0, 4, 4, 4]^T$.

Next, we distribute the availabilities of the neighboring routers to the input buffers. Since the *West* output port is connected to the *North* input port and $\Delta > 3$, the availability of the *West* output port is added to the *North* input port. Hence, the availability vector becomes: $A = [4, 3, 4, 4, 4]^T$.

No input port is connected to the *North*, *South* and *East* output ports. Therefore, their availabilities are distributed uniformly to all input ports. For example, after the 8-flit availability from the *North* output port is distributed to *Local, West, South* and *East* input ports, the availability vector becomes: $A = [6, 3, 6, 6, 6]^T$.

The computation of the availabilities can be written in matrix notation (similar to Eq. 7.13), as follows:

$$\begin{array}{ccccc} & \text{From port} & & \begin{array}{ccccc}\text{L} & \text{N} & \text{W} & \text{S} & \text{E}\end{array} & \text{To port} \\ A = & \begin{bmatrix}4\\4\\4\\4\\4\end{bmatrix} - \begin{bmatrix}0\\4\\0\\0\\0\end{bmatrix} & + & \begin{bmatrix}0 & 8/4 & 0 & 8/4 & 8/4\\ 0 & 0 & 3 & 8/4 & 8/4\\ 0 & 8/4 & 0 & 8/4 & 8/4\\ 0 & 8/4 & 0 & 0 & 8/4\\ 0 & 8/4 & 0 & 8/4 & 0\end{bmatrix} & \begin{array}{c}\text{L}\\ \text{N}\\ \text{W}\\ \text{S}\\ \text{E}\end{array}\end{array} \tag{7.15}$$

The first two components correspond to the initialization step, i.e. $B - X(n_0)$. In the last term, each column describes how the availability from each output port is distributed. For example, the first column is all zeros, since the first column of $C(n_0)$ is zero. On the other hand, the 8-flit availability from the *North* port is distributed uniformly to *Local, West, South* and *East* input ports, as described by the second column. Likewise, the fourth and fifth columns (which denote the availabilities from *South* and *East* ports) are uniformly distributed. However, all availabilities from the *West* output port go to the *North* input port, as described by the third column. We note that the entries in each column sum up to the availabilities obtained from the neighbors. Similarly, sum of the availabilities in each row is added to the corresponding port. Finally, Eq. 7.15 can be rewritten by factoring $C(n_0)$ out such that it takes the form of Eq. 7.13:

$$\begin{array}{cccccc} A = & \begin{bmatrix}4\\0\\4\\4\\4\end{bmatrix} & + & \begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 1/4 & 0 & 1/4 & 1/4 & 1/4\\ 0 & 1 & 0 & 0 & 0\\ 1/4 & 1/4 & 1/4 & 0 & 1/4\\ 1/4 & 1/4 & 1/4 & 1/4 & 0\end{bmatrix}^{\mathrm{T}} & \begin{bmatrix}0\\8\\3\\8\\8\end{bmatrix} \\ & B - X(n_0) & + & G(n_0) & C(n_0 - 1, k)\end{array}$$

Hence, the components of the matrix $G(g_{ij})$ in Eq. 7.13 reflect the distribution process as demonstrated in this example.

7.6.3 *Using Prediction for Network Control*

The overall operation of the proposed flow controller in summarized in Fig. 7.8. Each router in the network updates its availability periodically by aggregating the

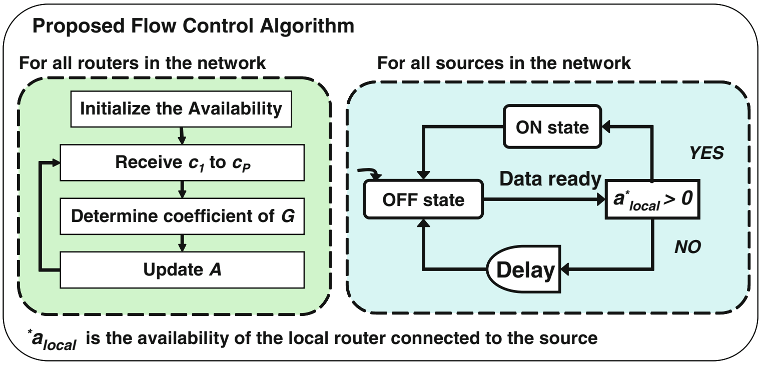

Proposed Flow Control Algorithm
For all routers in the network
Initialize the Availability
Receive c_1 to c_P
Determine coefficient of G
Update A
For all sources in the network
ON state
YES
OFF state
Data ready
$a^*_{local} > 0$
NO
Delay
$^*a_{local}$ is the availability of the local router connected to the source

Fig. 7.8 Operation of the proposed flow control algorithm is illustrated

data received from the immediate neighbors. As a result, the availability of a local input port connected to a traffic source reflects the backpressure from all of the downstream routers. In the absence of the proposed flow controller, a traffic source switches freely from OFF state to ON state, whenever it needs to inject a packet to the network. As opposed to this, the traffic sources check the availability of its host router, before entering the ON state. In analogy with wireless networks, the sources *listen before transmit*, i.e. sense the congestion in the network through the local router. As such, when a traffic source sees that the input port connected to it has zero availability, it delays the generation of new packets until the availability of the port becomes greater than zero, as shown in Fig. 7.8.

Since the congestion information propagates in the network through aggregation, prediction step k is selected as the diameter of the network. In this way, the timely transmission of the prediction to the traffic sources is guaranteed, since the information exchange between the neighboring routers is achieved by a small number (i.e. $log_2(a_P(0,k))$) of dedicated control wires and the availability signals do not experience queuing delays.

7.6.4 On the Stability of the Proposed Flow Control Algorithm

The proposed control technique regulates total number of packets in the network. In this section, we analyze the stability of the total number of packets in the network and show the network is indeed stable.

Let the average number of packets in the nth cycle be $S(n)$. We note that, $S(n)$ is the sum of the average number of packets at all ports of all routers, i.e.

$$S(n) = \sum_{\forall router\, r} \sum_{\forall port\, i} x_{r,i^{(n)}}.$$

The dynamics of the average number of packets in the network can be written as

$$S(n) = S(n-1) + Input(n-1) + Output(n-1) \tag{7.16}$$

where $S(n-1)$ is the average number of packets in the previous cycle, while $Input(n-1)$ and $Output(n-1)$ denote the average number of packets injected to and ejected from the network, respectively. The average number of packets ejected from the network can be approximated as:

$$Output(n-1) = \frac{S(n-1)}{\tau} \tag{7.17}$$

where τ is the average time the packets spend in the network. Intuitively, $S(n-1)/\tau$ is the average rate of ejection (it gives the average number of packets ejected in the $(n-1)$th cycle when multiplied with 1 *cycle* duration). This approximation is quite useful, since it vanishes as $S(n-1)$ approaches to zero as expected without using any nonlinear function.

The average number of packets injected to the network is determined by the processing/storage node and the controller. We note that the controller does not let the nodes to inject more packets than the available slots in the network. Let us express the total network capacity as:

$$Capacity = \sum_{\forall router\, r} \sum_{\forall port\, i} b_{r,i}$$

where $b_{r,i}$ is the capacity of the buffer in router r, port i in terms of number of flits. Then, the average number of packets injected to the network in the $(n-1)$th cycle can be expressed as:

$$Input(n-1) = l \times (Capacity - S(n-1)) \tag{7.18}$$

That is, a fraction of the available buffering space in the network $(Capacity - S(n-1))$ determined by $0 < l \leq 1$ is accepted to the network by the controller. l is a function of the network traffic and current congestion level. Intuitively, the total number of packets accepted to the network $(Input(n-1))$ is inversely proportional with the current number of packets in the network $(S(n-1))$. Moreover, $Input(n-1)$ vanishes as $S(n-1)$ reaches the network capacity. Hence, the controller induces a negative feedback. The dynamics of the average number of packets in the network (Eq. 7.16) can be rewritten using Eqs. 7.17 and 7.18, as follows:

$$\begin{aligned} S(n) &= S(n-1) + l \times (Capacity - S(n-1)) - \frac{S(n-1)}{\tau} \\ S(n) &= \left(1 - l - \frac{1}{\tau}\right) S(n-1) + l \times Capacity \end{aligned} \tag{7.19}$$

We note that the discrete time system defined by Eq. 7.19 is stable if and only if the eigenvalues of the state transition matrix, i.e., $1 - l - \frac{1}{\tau}$ in our case, are within the unit circle. Hence, the condition for stability can be written as:

$$\begin{gathered} -1 < 1 - l - \frac{1}{\tau} < 1 \\ -\frac{1}{\tau} < l < 2 - \frac{1}{\tau} \end{gathered} \tag{7.20}$$

The left hand side of this equation is trivially satisfied, since $l > 0$. For the right hand side, we know that the average time the packets spend in the network (τ) is larger than 1 cycle, so $2 - \frac{1}{\tau} > 1$, which implies . Therefore, the stability condition expressed in Eq. 7.20 is satisfied; so the average number of packets in the network is stable under the proposed control scheme.

7.7 Experimental Results

In this section, we demonstrate the effectiveness of the proposed flow control technique using an audio/video system complying with the H263.1 standard, as well as synthetic benchmarks which are all mapped to a 4 × 4 2D mesh network. Wormhole routing and deterministic XY routing algorithm is used throughout the simulations. The simulations are performed using a custom cycle-accurate NoC simulator which implements a basic ON/OFF switch-to-switch flow control, the ON/OFF traffic sources and the flow control scheme.

The simulations are repeated for a range of buffer sizes in routers and local PEs. The results reported next are obtained for 4-flit input buffers in the routers and 100-flit local memory in the host PE.[4] The average packet latency reported in this chapter includes the latency experienced due to the local memory (i.e. the source queuing delay), and network latency. The network latency denotes the time the packet travels in the network before being ejected. Finally, we also present experimental results obtained using the FPGA prototype.

7.7.1 *Audio/Video System*

We first used the audio/video system described in [13] to evaluate the potential of the proposed algorithm for real applications. The target system includes an H263 video encoder, an H263 video decoder, an MP3 audio encoder, and an MP3 audio

[4] Note that the local memory in the host PE is *not* part of the router. The 100-flit local buffer is used to emphasize that (*i*) its size is finite and (*ii*) PEs sense the backpressure from the network for the switch-to-switch flow control.

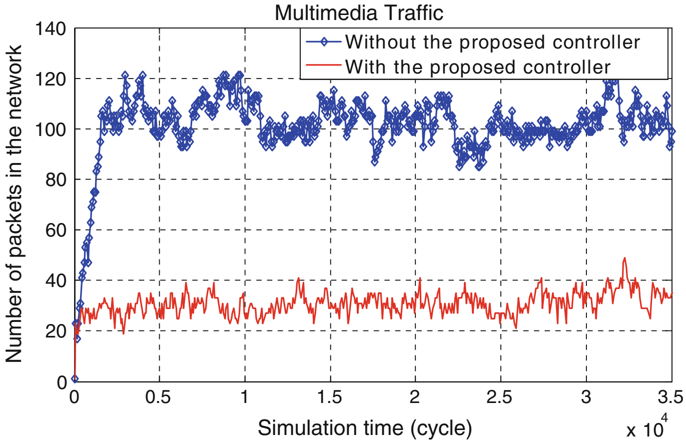


Fig. 7.9 Variation of the number of packets in the network over time for multimedia traffic.

Table 7.1 The reduction in the average packet latency and number of packets in the network due to the proposed flow control algorithm are indicated

	Switch-to-switch control only	The proposed flow control	Reduction ×
Ave. latency	149 (112) *cycles*	47 (12) *cycles*	3.2
Max. latency	897 (774) *cycles*	466 (404) *cycles*	1.9
Standard deviation of latency	173.8 (156) *cycles*	55.7 (46) *cycles*	3.1
Ave. # of packets	94 *packets*	29 *packets*	3.2
Max. # of packets	129 *packets*	52 *packets*	2.5
Standard deviation of # of packets	24.7 *packets*	7.2 *packets*	3.4

The latency values are the sum of the latencies experienced at the source queue and in the network. The average latencies experienced at the source queues are also given (in parentheses)

decoder. It is partitioned into 40 concurrent tasks and then these tasks are mapped to the 4 × 4 2D mesh network. Finally, traffic traces obtained from real video and audio clips are used to determine the communication patterns among the processing cores in the network.

The audio/video system is first simulated using only the switch-to-switch flow control. When the offered load is about half of the maximum achievable throughput, the average and maximum packet latencies in the network are found to be 149 and 897 *cycles*, respectively. After that, the simulations are repeated with the proposed flow controller in place. As shown in strictly increasing, the average packet latency becomes 47 *cycles*, while the maximum packet latency drops to 466 *cycles*. Table 7.1 summarizes also the queueing delay experienced at the traffic sources (shown separately inside parentheses). We observe that the average source queuing delay reduces from 112 to 12 *cycles* with the use of the proposed

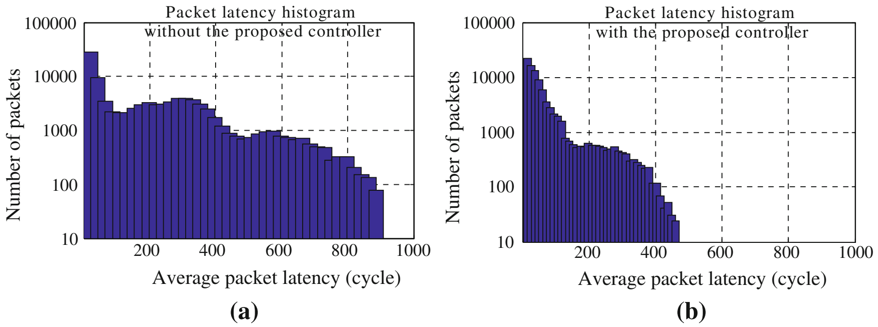


Fig. 7.10 Histogram of the packet latencies for the Audio/Video traffic without (**a**) and with (**b**) the proposed flow controller. Note that the y-axes of the plots are in log-scale. A significant improvement due to the flow-control mechanism can be observed

controller. Similarly, the maximum value of the source queuing delay and the standard deviation drops significantly as a result of the proposed controller.

This huge reduction in packet latencies is mainly due to the reduced number of packets in the network. As mentioned before, unlike the switch-switch flow controller, the proposed controller regulates the number of packets in the network directly. As such, the average number of packets in the network drops from 94 to 29 *packets* which is about a 3.2× reduction, as summarized in Table 7.1. Likewise, the maximum number of packets in the network and the standard deviation of the number of packets in the network drop by 2.5× and 3.4×, respectively.

We further investigate the number of packets travelling through the network as a function of time in Fig. 7.9. Without the flow controller, the number of packets quickly rises to about 100 packets and oscillates around the average value (94 *packets*) with a standard deviation of 24.7 *packets*. On the other hand, the proposed controller provides about 3.2× reduction in the average number of packets and 3.4× reduction in standard deviation, as summarized in Table 7.1 and plotted in Fig. 7.9. The variation in the packet latency and number of packets observed in the absence of the proposed controller show that the network can oscillate between congested and free traffic due to the overlap in the ON periods of traffic sources, as discussed in Sect. 7.4.3.

To better understand the effects of the controller, we further analyze the histogram of the packet latencies (Fig. 7.10). We notice that, for the network without the proposed flow controller, about 50 % of the packets experience longer delays than the average delay (i.e 149 *cycles*). The packets located at the tail of the distribution in Fig. 7.10a are the main cause for this poor performance. The technique we propose prevents the packets that are likely to experience such long delays from entering the network. Indeed, as depicted in Fig. 7.10b, the latency histogram is pushed significantly towards left; so about 91 % of packets experience less than 100 *cycles* latency. We observe that there are no packets with latency more than 466 cycles, if the proposed controller is used. Moreover, the number of packets with latency more than 100 cycles drops quickly due to the

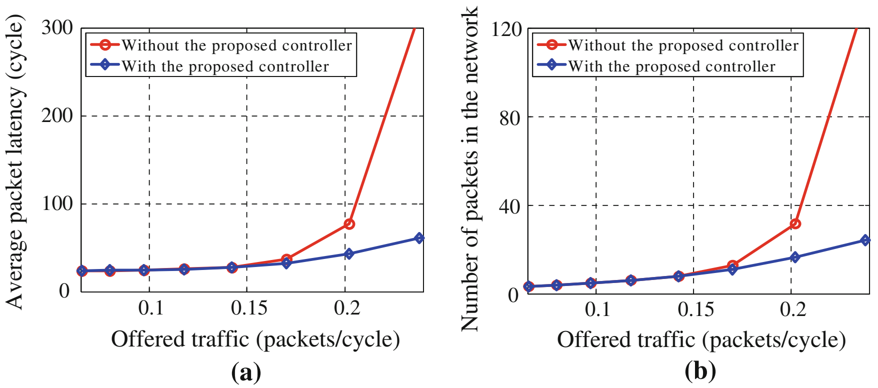


Fig. 7.11 **a** Average packet latency and **b** number of packets in the network are plotted as a function of the offered traffic

proposed controller. On the other hand, half of the packets experience latency longer than 100 cycles in the absence of the proposed controller. Besides this, there are packets with as high as 900 cycles latency.

Since the audio/video application we consider includes strong access locality, the average hop count for this application on the 4 × 4 mesh network is only 1.98, while the average hop count for uniform traffic would be 2.67. For applications with less access locality, the number of packet contentions; therefore, the improvement due to the proposed controller is expected to be larger.

7.7.2 Synthetic Traffic

Additional experiments are performed for uniform and hotspot traffic patterns to further assess the effectiveness of the proposed controller. First, we compare the performance of a 4 × 4 2D mesh network under hotspot traffic *with* and *without* the proposed controller. The average packet latency in the network is plotted as a function of the packet injection rate in Fig. 7.11a. We observe that without the flow controller, the network becomes congested as the packet injection rate increases. The reason for this behavior is revealed in Fig. 7.11b. Indeed, in absence of a traffic controller, the number of packets in the network grows at an increasing pace as the traffic injection rate increases. The flow controller, on the other hand, effectively limits the number of packets injected to the network, as depicted in Fig. 7.11b. This, in turn, results in significant improvements in the average packet latency. Finally, Fig. 7.11a, b demonstrate that the average packet latency is proportional to the average number of packets in the network and justify once more controlling the packet injection as an effective means for improving the NoC performance.

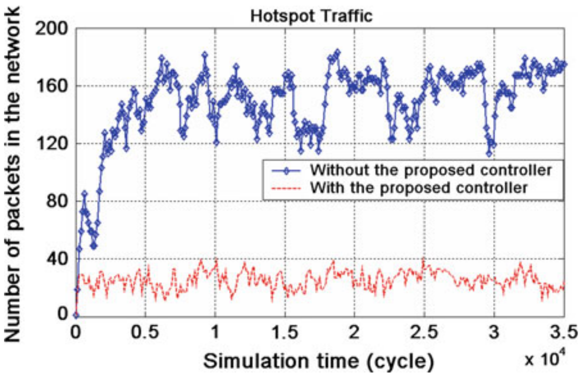


Fig. 7.12 Variation of the number of packets in the network over time for the hotspot traffic. The proposed controller reduces the average number of packet in the network and their variation over time significantly

Table 7.2 The reduction in the latency and number of packets in the network due to the proposed flow control algorithm

	Switch-to-switch control only (*packets*)	The proposed flow control (*packets*)	Reduction (×)
Ave. # of packets	151	25	6.0
Max. # of packets	189	45	4.2
Standard deviation of # of packets	24.6	6.2	4.0

Similar to the multimedia traffic, we monitored the number of packets in the network when the packet injection rate was about half of the maximum throughput. The mean and variance of the number of packets in the network is significantly reduced with the proposed controller, as depicted in Fig. 7.12. More specifically, the average number of packets in the network drops from 151 to 25 packets resulting in 6× reduction, while the maximum number of packets in transit decreases from 189 to 45 packets. Furthermore the standard deviation of the number of packets is reduced from 24.6 to 6.2, as summarized in Table 7.2.

7.7.3 Impact of the Local Buffer Size on Performance

The PEs write the packets that will be transmitted over the network to a local memory in the network interface. Then, the local router reads the packets from this memory. In general, the buffering space available at the local PEs is much larger than the buffering space at the routers. Nevertheless, local buffering space has also a finite value and may fill up as a result of the backpressure from the network. When the local memory at the PE becomes full, the PE cannot write to this memory. Since no packets are dropped, the PE pauses in this situation.

In this section, we compare the performance of the predictive controller and switch-to-switch control for various buffer sizes in the local PE. For the switch-to-

Table 7.3 Average packet latency for the *hotspot* traffic (at 0.2 *packets/cycle* traffic rate) without and with the proposed controller for different local memory sizes are summarized

Local PE buffer size	50-flit		100-flit		200-flit		500-flit	
	Ave. latency (*cycles*)	Ave. PE pausing (*cycles*)	Ave. latency(*cycles*)	Ave. PE pausing (*cycles*)	Ave. latency(*cycles*)	Ave. PE pausing (*cycles*)	Ave. latency (*cycles*)	Ave. PE pausing (*cycles*)
Without the proposed controller	80	0.75	106	0.43	158	0.31	217	0.14
With the proposed controller	43	1.93	44	1.94	44	1.92	44	1.96

switch control, the average packet latency at a given traffic injection rate increases considerably with increasing local buffer size, as summarized in the first row of Table 7.3. Switch-to-switch control is less effective for large local buffers, since a large number of packets can be generated before the PE feels the backpressure.

In Table 7.3, we also give the average number of cycles the PEs pause. We observe that the PEs pause on average 0.75 *cycles*, when the local memory size is 50 flits. The PEs pause very rarely, since the network is not heavily congested. We also observe that this number is reduced as the local memory size is increasing. This is also expected, since the PEs are affected by the backpressure less for large local memory.

In contrast to the switch-to-switch control, the flow controller uses directly the availability information predicted by routers as a measure of congestion level. Therefore, the traffic sources do not rely on backpressure from the network. Consequently, the flow controller performs well over a wide range of local PE sizes, as shown in the last row of Table 7.3. Table 7.3 also shows that the flow controller causes PEs to pause more often than the switch-to-switch control. This is also expected, since the controller delays the packet generation. However, we observe that the average time the PEs pause is about 1.9 *cycles*, as shown in Table 7.3. Hence, the sum of the latency and pausing time is still much smaller when using the flow controller.

7.7.4 Scalability of the Approach

The computation of the availabilities at the routers depend only on the local information received from the immediate neighbors. Therefore, the computational complexity depends only on the number of ports in the router, *not* on the size of the network.

In order to demonstrate the performance of the flow control technique for larger network sizes and provide a more quantitative evaluation, we present some new experiments involving 4 × 6 and 6 × 8 networks. Figure 7.13a, b show the average packet latency as a function of the effective traffic injection rate for the 4 × 6 and 6 × 8 networks, respectively. We note that, the *x-axes* of these figures show the effective traffic injection rates, which take the PE pausing (discussed in Sect. 7.7.3) into account. Since the flow controller limits the number of packets in the network to prevent congestion, the total traffic injection rate is limited, as shown in Fig. 7.13. In particular, the maximum effective injection rate is 0.51 packets/cycle for the 4 × 6 NoC and 0.72 packets/cycle for the 6 × 8 NoC. On the other hand, without the flow controller, the effective traffic injection rate continues to increase even in the presence of congestion. Hence the average packet latency grows significantly. As it can be clearly seen from Fig. 7.13, the flow controller improves the average latency especially at larger injection rates.

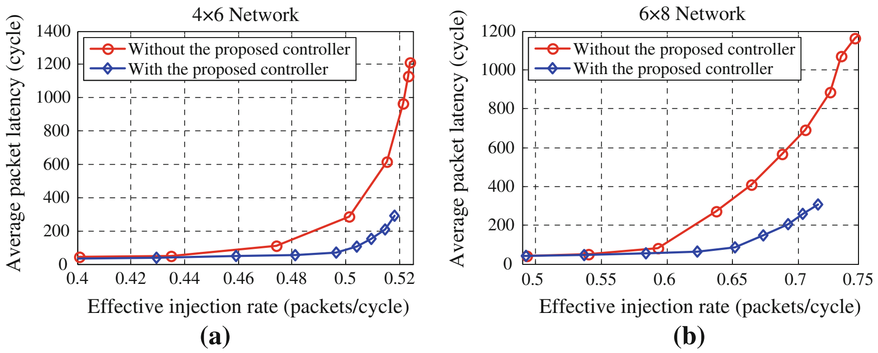


Fig. 7.13 Average packet latency as a function of effective traffic injection rate is plotted for **a** 4 × 6 and **b** 6 × 8 networks under hotspot traffic pattern

7.7.5 Evaluation with an FPGA Prototype

We also present results directly measured using the FPGA prototype to support the simulation results. For this purpose, a 4 × 4 Mesh network is implemented with and without the proposed flow controller. Similar to the simulations, a basic link level ON/OFF flow control is implemented in both cases, and deterministic XY routing is employed. The concrete implementation of the flow controller is presented in Appendix A.5.

The reduction in the mean and standard deviation of the packet latencies in the network are similar to those obtained using simulation. Table 7.4 summarizes measured values for the mean and standard deviation for a wide range of traffic injection rates. For instance, at 0.016 *packets/cycle* traffic injection rate, the average packet latency without the flow controller is found as 32.6 *cycles*. The controller reduces the average latency to only 18.2 *cycles* including the waiting time in the processing element, as shown in Table 7.4. Even more importantly, *without* the flow controller, the standard deviation of packet latencies over multiple simulations is as large as 17.6 *cycles;* this is due to the lack of capability of the link level flow control to regulate number of packets in the network.

Table 7.4 Mean and standard deviation of packet latencies obtained using the FPGA prototype are shown with and without the proposed controller

Applied traffic (*packets/cycle*)		0.016	0.031	0.062	0.126
Average latency (*cycles*)	W/o the proposed controller	32.6	46.1	94.8	169.4
	With the proposed controller	18.2	21.2	28.7	45.9
	Reduction (×)	1.8	2.2	3.3	3.7
Standard deviation of latency (*cycles*)	W/o the proposed controller	17.6	18.2	23.6	37.1
	With the proposed controller	0.15	8.1	13.2	14.7
	Reduction (×)	119	2.3	1.8	2.5

On the other hand, the standard deviation when using the flow controller is less than one cycle. Even though this value increases for heavier traffic, we see about 2× improvement over the basic link level controller over a wide range of input traffic.

7.8 Summary

Effective flow control mechanisms are necessary for efficient utilization of network resources in NoCs. However, neither switch-to-switch, nor end-to-end control schemes proposed so far for classical networks can satisfy the requirements of NoCs. On the other hand, the predictive flow control algorithm presented in this chapter controls the packet injection rate in order to regulate the number of packets in the network, similar to the end-to-end flow controller. At the same time, our approach relies only on local information transfer between the neighboring routers. Therefore, it has a low communication overhead, similar to the switch-to-switch flow control.

References

1. Adriahantenaina A, Greiner A (2003) Micro-network for SoC: implementation of a 32-Port SPIN network. In: Proceedings of design, automation and test in Europe conference, March 2003
2. Baydal E, Lopez P, Duato J (2005) A family of mechanisms for congestion control in wormhole networks. IEEE Trans Parallel Distrib Syst 16(9):772–784
3. Bertsekas D, Gallager R (1992) Data networks. Prentice Hall, Upper Saddle River
4. Chien AA (1998) A cost and speed model for k-ary n-cube wormhole routers. IEEE Trans Parallel Distrib Syst 9(2):150–162
5. Dally WJ, Towles B (2004) Principles and practices of interconnection networks. Morgan Kaufmann, San Fransisco
6. Dally WJ (1992) Virtual-channel flow control. IEEE Trans Parallel Distrib Syst 3(2):194–205
7. Dally WJ, Towles B (2001), Route packets, not wires: on-chip interconnection networks. In: Proceedings of design automation conference, June 2001
8. Duato J, Yalamanchili S, Ni L (2002) Interconnection networks: an engineering approach. Morgan Kaufmann, San Mateo
9. Gerla M, Kleinrock L (1980) Flow control: a comparative survey. IEEE Trans Commun 28(4):553–574
10. Harmanci M, Escudero N, Leblebici Y, Ienne P (2004) Providing QoS to connection-less packet-switched NoC by implementing DiffServ functionalities. In: Proceedings of international symposium on system-on-chip, November 2004
11. Harmanci M, Escudero N, Leblebici Y, Ienne P (2005) Quantitative modeling and comparison of communication schemes to guarantee quality-of-service in networks-on-chip. In: Proceedings of the international symposium on circuits and systems, May 2005
12. Hedetniemi SM, Hedetniemi ST, Liestman AL (1988) A survey of gossiping and broadcasting in communication networks. Networks 18(4):319–349
13. Hu J, Marculescu R (2005) Energy- and performance-aware mapping for regular NoC architectures. IEEE Trans Comput Aided Des Integr Circ Syst 24(4):551–562

14. Hyatt C, Agrawal DP (1997) Congestion control in the wormhole-routed torus with clustering and delayed deflection. In: Proceedings of parallel computing, routing and communication workshop
15. Jalabert A, Murali S, Benini L, De Micheli G (2004) XpipesCompiler: a tool for instantiating application specific networks on chip. In: Proceedings of design, automation and test in Europe conference, February 2004
16. Lee HG, Chang N, Ogras UY, Marculescu R (2007) On-chip communication architecture exploration: a quantitative evaluation of point-to-point, bus and network-on-chip approaches. ACM Trans Des Autom Electron Syst 12(3):23
17. Lopez P, Martinez JM, Duato J (1998) DRIL: dynamically reduced message injection limitation mechanism for wormhole networks. In: Proceedings of international conference parallel processing, August 1998
18. Mendel JM (1995) Lessons in estimation theory for signal processing, communications, and control. Prentice-Hall, Upper Saddle River
19. Millberg M, Nilsson E, Thid R, Jantsch A (2004) Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In: Proceedings of design, automation and test in Europe conference, February 2004
20. Murali S, Benini L, De Micheli G (2005) Mapping and physical planning of networks on chip architectures with quality-of-service guarantees. In: Proceedings of Asia and South Pacific design automation conference, January 2005
21. Nilsson E, Millberg M, Oberg J, Jantsch A (2003) Load distribution with the proximity congestion awareness in a network on chip. In: Proceedings of design, automation and test in Europe conference, March 2003
22. Paganini F, Doyle J, Low S (2001) Scalable laws for stable network congestion control. In: Proceedings of IEEE conference on decision and control, December 2001
23. Park K, Willinger W (eds) (2000) Self-similar network traffic and performance evaluation. Wiley, New York
24. Peh L, Dally WJ (2001) A delay model for router micro-architectures. IEEE Micro 21(1):26–34
25. Pullini A, Angiolini F, Bertozzi D, Benini L (2005) Fault tolerance overhead in network-on-chip flow control schemes. In: Proceedings of symposium on integrated circuits and system design, September 2005
26. Qiu D, Shro BN (2004) A predictive flow control mechanism to provide QoS and efficient network utilization. IEEE Trans Network 12(1):73–84
27. Radulescu A et al (2005) An efficient on-chip ni offering guaranteed services, shared-memory abstraction, and flexible network configuration. IEEE Trans Comput Aided Des Integr Circ Syst 24(1):4–17
28. Smai A, Thorelli L (1998) Global reactive congestion control in multicomputer networks. In: Proceedings of the fifth international conference on high performance computing , December 1998
29. Thottethodi M, Lebeck AR, Mukherjee SS (2001) Self-tuned congestion control for multiprocessor networks. In: Proceedings of the 7th international symposium on high-performance computer architecture, January 2001
30. Varatkar G, Marculescu R (2004) On-chip traffic modeling and synthesis for MPEG-2 video applications. IEEE Trans VLSI 12(1):108–119
31. Zeferino CA, Santo FME, Susin AA (2004) Paris: a parameterizable interconnect switch for networks-on-chip. In: Proceedings of symposium on integrated circuits and systems design, September 2004

Chapter 8
Design and Management of VFI Partitioned Networks-on-Chip

The design of many core systems-on-chip (SoCs) has become increasingly challenging due to high levels of integration, excessive energy consumption, and clock distribution problems. To deal with these issues, this chapter considers network-on-chip (NoC) architectures partitioned into several voltage-frequency islands (VFIs) and propose a design methodology for runtime energy management. The proposed approach minimizes the energy consumption subject to performance constraints. Then, we present efficient techniques for on-the-fly workload monitoring and management to ensure that the system can cope with variability in the workload and various technology-related parameters. Finally, the results and functional correctness are validated using an FPGA prototype for an NoC with multiple VFIs.

8.1 Introduction

As recognized by the International Roadmap for Semiconductors, dealing with on-chip communication and power management problems require a drastic departure from the classic design methodologies [38]. Besides its advantages in terms of modularity, design re-use, and performance, the NoC approach offers a matchless platform for implementing the *globally asynchronous, locally synchronous* (GALS) design paradigm [9, 28]; this makes the clock distribution and timing closure problems more manageable. In addition, a GALS design style fits nicely with the concept of VFIs, which has been recently introduced for allowing fine-grain system-level power management.

The use of VFIs in the NoC context is likely to provide better power-performance trade-offs than its single voltage, single clock frequency counterpart, while taking advantage of the natural partitioning and mapping of applications onto the NoC platform. However, despite the huge potential for energy savings when using VFIs, the NoC design methodologies considered so far are limited to a single voltage-clock domain [2, 14, 19, 27]. On the other hand, studies that do consider

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1_8,

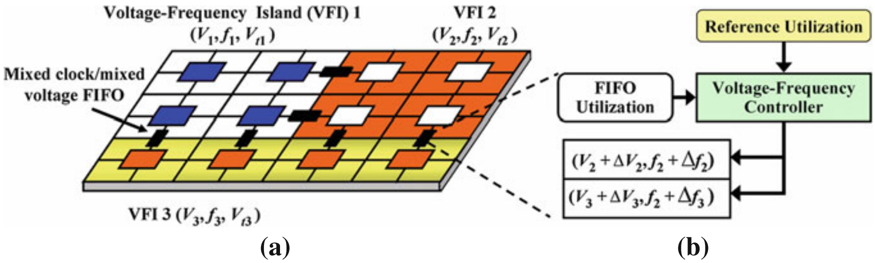

Voltage-Frequency Island (VFI) 1
(V_1, f_1, V_{t1})
VFI 2
(V_2, f_2, V_{t2})
Mixed clock/mixed voltage FIFO
VFI 3 (V_3, f_3, V_{t3})
Reference Utilization
FIFO Utilization
Voltage-Frequency Controller
($V_2+\Delta V_2$, $f_2+\Delta f_2$)
($V_3+\Delta V_3$, $f_2+\Delta f_3$)
(a)
(b)

Fig. 8.1 A sample 2D mesh network with three VFIs. Communication across different islands is achieved through mixed clock/mixed voltage FIFOs. The VFI partitioning and static voltage-frequency assignment approach in Sect. 8.3 generates architectures like in (**a**). The dynamic voltage- frequency control approach detailed in Sect. 8.4 provides fine control around these static values as depicted in (**b**)

multiple VFIs assume that each module/core in the design belongs to a different island and different islands are connected by P2P links [12, 31].

This chapter explores the design and optimization of novel NoC architectures partitioned into multiple VFIs which rely on a GALS communication paradigm. In such a system, each voltage island can work at its own speed, while the communication across different voltage islands is achieved through mixed clock/mixed voltage FIFOs (see Fig. 8.1). This provides the flexibility to scale the frequency and voltage of various VFIs in order to minimize energy consumption. As a result, the advantages of both NoC and VFI design styles can be exploited simultaneously.

In this chapter, we first present a design methodology for *partitioning* a given NoC architecture into multiple voltage-frequency domains and *assigning* the supply and threshold voltages (hence the corresponding clock frequencies) to each domain such that the *total* energy consumption is minimized under given performance constraints. Since the characteristics of the application running on the NoCs are subject to run-time workload and parameter variations, we further develop an online feedback control mechanism that dynamically adjusts the operating voltage and frequency around the static values.

Regarding the *static* VFI partitioning and voltage/frequency assignment, the basic idea is to start with an NoC configuration where each PE belongs to a separate VFI characterized by given supply and threshold voltages and local clock speed (i.e., having initially *N* VFIs, for *N* PEs). This configuration may achieve the minimum *application* energy consumption, but not necessarily minimize the *total* energy consumption due to the additional overhead incurred by implementing a large number of VFIs. Indeed, the associated design complexity increases due to the overhead in implementing the mixed-clock/mixed-voltage FIFOs and voltage converters required for communication across different VFIs, as well as the power distribution network needed to cover multiple VFIs. Therefore, the partitioning technique needs to find two candidate VFIs to merge such that the decrease in the energy consumption is the largest among all possible merges, while performance constraints are still met. This process is repeated until a single VFI implementation is obtained. Consequently, for all possible levels of VFI granularities (i.e., $1, 2, \ldots, N$ VFIs), we can obtain the partitioning and corresponding voltage level assignments such that the total energy is minimized, subject to given performance constraints. Finally, among all VFI partitionings determined by this iterative process, the one providing the minimum energy consumption is selected as being the solution of the VFI partitioning problem.

With respect to the *w*orkload-driven dynamic voltage and frequency scaling, we propose a feedback control system that performs *online* fine grain voltage-frequency control around the static values found by the proposed partitioning algorithm. Indeed, the dynamic control of various voltages and frequencies can help the system adapt to a lower power consumption and, at the same time, meet the performance requirements. To this end, we use the utilization of the mixed-clock/mixed-voltage FIFOs at the interface between any two VFIs, to set the voltage and frequency values of the associated VFIs.

The remainder of this chapter is organized as follows. Section 8.2 reviews the related work. Section 8.3 presents the problem formulation and solution to VFI partitioning and static voltage-frequency assignment. Section 8.4 presents the workload-driven dynamic voltage and frequency control technique. Experimental results are included in Sect. 8.5. Section 8.6 discusses possible extensions of the basic approach. Finally, Sect. 8.7 summarizes the main ideas in this chapter.

8.2 Related Work

GALS-based systems consist of several synchronous IPs that communicate with each other asynchronously [9, 28]. There have been many efforts to design low latency asynchronous communication mechanisms between synchronous blocks [11]. Some of them include two flip-flop synchronizers or asynchronous FIFO [8], while others consider stoppable clocks [28].

The design style based on multiple VFIs has been proposed in [23] and received attention also from industry [30, 40]. It fits very well with the GALS design style, where the synchronous IPs in the design have both different voltages and frequencies. Despite the natural match between the VFI design style and NoC platforms, many existing design methodologies for NoCs are confined to the single voltage/single frequency domain case [2, 14, 19, 27].

There have been several design efforts to combine the benefits of NoC interconnect mechanism with GALS-based design style. For instance, the authors of [3] present a clockless NoC capable of providing QoS guarantees. An FPGA prototype of a GALS-based NoC with two synchronous IPs is presented in [35]. A method to reduce the wire propagation delays in a GALS-based NoC is proposed in [7]. However, these studies assume that each node in the network belongs to a separate clock domain, which is a costly proposition.

Dynamic voltage-frequency scaling techniques for multiple clock domain processors have been addressed in [24, 41] and references therein. The authors of

[41] propose a technique based on proportional-integral-derivative (PID) controller, which relies on manual tuning of the control gains. Therefore, this approach may become prohibitive and requires a coordination mechanism when the number of voltage-clock domains increases [22].

8.3 VFI Partitioning and Static Voltage Assignment Problems

8.3.1 Basic Assumptions and Methodology Overview

We consider a tile-based implementation laid out as a grid, as shown in Fig. 8.1a. Each tile contains a processing element (referred to as PE) and a router. We assume wormhole flow control and XY routing. Communication across different voltage-frequency islands is achieved through mixed-clock/mixed-voltage FIFOs.

The target application is first scheduled to the target NoC architecture which consists of several heterogeneous PEs. We assume the earliest deadline first (EDF) and energy aware scheduling (EAS) to generate both computation and communication task schedules [18]. As shown in Fig. 8.2, given the target architecture, the driver application and its schedule, the methodology determines the VFI partitioning and the supply and threshold voltage assignment to the VFIs. The voltages are assigned such that the total energy spent in *both* computation and communication is minimized, subject to performance constraints that are imposed by the driver application as *deadlines* for certain tasks and/or minimum *throughput* requirements.

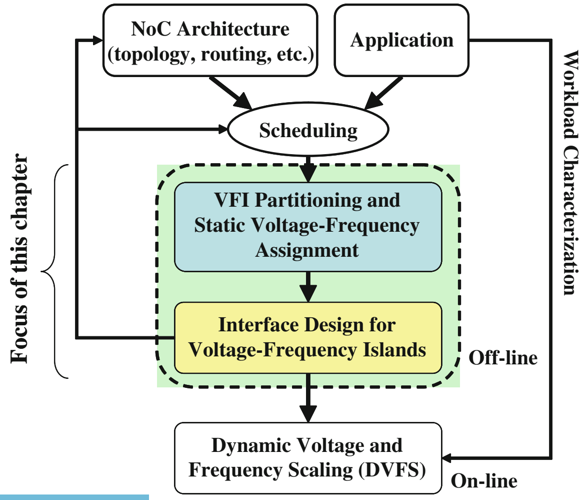

NoC Architecture (topology, routing, etc.)
Application
Scheduling
VFI Partitioning and Static Voltage-Frequency Assignment
Interface Design for Voltage-Frequency Islands
Dynamic Voltage and Frequency Scaling (DVFS)
Focus of this chapter
Workload Characterization
Off-line
On-line

Fig. 8.2 Overview of the overall methodology. Off-line VFI partitioning and static voltage-frequency assignment is detailed in Sect. 8.3, while the on-line feedback control mechanism is explained in Sect. 8.4

The voltage-frequency assignments computed using the approach presented in this section are static. For the applications with workload variability, the operating voltage and frequency are further controlled dynamically around these static values, as described in Sect. 8.4.

8.3.2 Problem Formulation

A. Energy and delay models

The set of tiles in the network is denoted by $T = \{1,\ldots,N\}$. The supply and threshold voltages for tile $i \in T$ are given by V_i and V_{ti}, respectively. Using this notation, the sum of dynamic and static energy consumptions associated with tile $i \in T$ can be written as:

$$E_i(V_i, V_{ti}) = R_iC_iV_i^2 + T_ik_iV_ie^{\left(-\frac{V_t}{S_t}\right)} \tag{8.1}$$

where R_i is the number of active cycles, C_i stands for the total switched capacitance per cycle, T_i is the number of idle cycles, k_i is a design parameter, and S_t is a technology parameter [5].

We assume that the static component of the communication energy is included in the second term of Eq. 8.1, since each link and router belongs to a tile in the network. The dynamic part of the communication energy consumption is found using the bit energy metric [43] defined as:

$$E_{bit} = E_{L_{bit}} + E_{B_{bit}} + E_{S_{bit}} \tag{8.2}$$

where $E_{L_{bit}}, E_{B_{bit}}$ and $E_{S_{bit}}$ represent the energy consumed by the link, buffer and switch fabric, respectively. Assuming the bit energy values are measured at V_{DD}, the energy needed to transmit one bit from tile $src \in T$ to tile $dst \in T$ is found as:

$$E_{bit}(src, dst) = \sum_{i \in p} \left(E_{L_{bit}}(i) + E_{B_{bit}}(i) + E_{S_{bit}}(i)\right)\frac{V_i^2}{V_{DD}^2} \tag{8.3}$$

where P is the set of tiles on the path from tile *src* to tile *dst*.

The clock period for each tile i is a function of its supply and threshold voltage, and it can be expressed as:

$$\tau_i(V_i, V_{ti}) = \frac{K_iV_i}{(V_i - V_{ti})^{\alpha}} \tag{8.4}$$

where α (we use $\alpha = 1.2$) is a technology parameter and K_i is a design-specific constant [36]. Thus, the operating frequency of a tile and the *VFI j* it belongs to, is determined by the largest cycle time of the comprising tiles, i.e.,

$$f_j \leq \underset{i \in S_j}{min} \left\{ \frac{1}{\tau_i(V_i, V_{ti})} \right\} \tag{8.5}$$

where S_j is the set of tiles that belong to *VFI j*. Finally, we assume each router is locally synchronous and communicates with its neighbors via mixed-clock/mixed-voltage FIFOs. As such, we compute the communication latency between tile *src* and tile *dst*, while sending *vol*(*src, dst*) bits of data, using:

$$t_{comm}(src, dst) = \sum_{i \in p} \frac{\mu_s}{f_i} + t_{fifo} \left\lceil \frac{vol(src, dst)}{W} \right\rceil \tag{8.6}$$

where W is the channel width and μ_s is the number of cycles it takes to traverse a single router and the outgoing link. t_{fifo} is the latency of the FIFO buffers. We determined t_{fifo} experimentally using the prototype described in Appendix A.6, but it can be also found using a worst-case delay model. To give a bit of intuition, the first term of Eq. 8.6 gives the latency for the header flits while passing through the routers on path P, while the second term represents the serialization latency.

B. The static voltage/speed assignment problem

Assume that the target application consists of a set of tasks G. For each task $t \in G$, the initial schedule specifies the deadline (d_t), start time (st_t), the number of cycles required to execute the task (x_t), as well as the tile where the task will run on. When the network is partitioned into N VFIs, $i = 1, \cdots, N$, (i.e., each tile belongs to a different island), the static voltage assignment problem can be formulated as follows:

Given an NoC architecture with N tiles, where each tile belongs to a separate VFI, a communication task graph describing the driver application, and its schedule,

Assign the supply (V_i) and threshold (V_{ti}) voltages, such that the total application energy consumption is minimized; that is:

$$min\, E_{App} = \sum_{\forall i \in T} E_i(V_i, V_{ti}) + \sum_{\forall i \in T} \sum_{\forall j \in T} vol(i,j) E_{bit}(i,j) \tag{8.7}$$

subject to deadline constraints expressed as:

$$\frac{x_t}{f_t} + t^t_{Comm} \le d_t - st_i \quad \forall t \in G \tag{8.8}$$

where x_t/f_t is the computation time and t^t_{Comm} is the communication delay encountered when task t needs to communicate with a task mapped to a different tile. The number of cycles required to execute the task x_t is given by the schedule and t^t_{Comm} is computed using Eq. 8.6. The left hand side of Eq. 8.8 gives the sum of computation and communication times, while the right-hand side gives the amount of time the task should be completed without violating the schedule.

C. The voltage-frequency island partitioning problem

Even though decoupling the supply and threshold voltages of each tile results in a system with the finest granularity for voltage/frequency control, the overhead of designing a large number of islands may undermine the potential for energy savings. In fact, it may be possible to merge several islands with a negligible increase in application energy consumption. In the extreme case, all tiles can be conceivably merged into a single VFI. Between the two extreme points, there exists a myriad of choices with varying energy/cost trade-offs. In order to compare these choices, we need to quantify the energy overhead of having extra VFIs. Towards this end, the energy overhead of adding one additional voltage-frequency island to an already existing design can be written as:

$$E_{VFI} = E_{ClkGen} + E_{Vconv} + E_{MixClkFifo} \tag{8.9}$$

where E_{ClkGen} is the energy overhead of generating additional clock signals [15] and E_{Vconv} denotes the energy overhead of voltage conversion [6]. Finally, $E_{MixClkFifo}$ is the overhead due to the mixed-clock/mixed-voltage FIFOs used in the interface of VFIs [8].

Besides energy, additional VFIs exhibit area and implementation overheads, such as routing multiple power distribution networks. There may be also a design or technology limitation on the maximum number of VFIs. Therefore, the maximum number of VFI is also taken as a design constraint. Finally, the VFI partitioning problem is formulated as follows:

Given

- An NoC architecture;
- The scheduling of the driver application across the network;
- Maximum number of allowed VFIs;
- Physical implementation constraints (e.g., certain tiles have to operate at a given voltage level, or a subset of tiles have to belong to the same VFI)

Find the optimum number of VFIs $n(n \leq N)$, the VFI partitioning; and assign the supply and threshold voltages to each island,

such that the *total energy consumption*, i.e.,

$$E_{Total} = E_{APP} + \sum_{i=1}^{n} E_{VFI}(i) \tag{8.10}$$

is minimized, subject to performance constraints in Eq. 8.8.

8.3.3 Motivational Example

As an example, we analyze the *office-automation* benchmark [13]. The application is scheduled on a 2 × 2 network using the EDF discipline, and the proposed approach is used for the VFI partitioning and static voltage assignment.

When the entire network consists of a single VFI, the supply and threshold voltages are found to be 1 and 0.15 V, respectively. The corresponding schedule and these voltage assignments results in 10.5 mJ energy consumption. When analyzing this design, we observe that one of the tasks has a zero slack available,

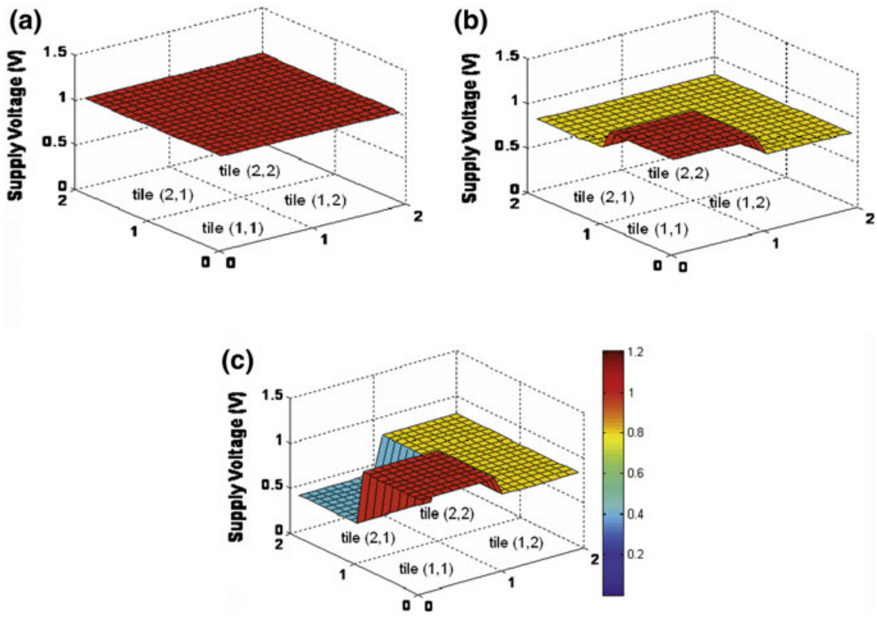


Fig. 8.3 Different voltage-frequency island partitionings and corresponding static voltage assignments for a 2 × 2 network. **a** Single VFI. **b** Two VFIs. **c** Three VFIs

while others have a positive slack. When the network consists of two islands, the task with a zero slack is decoupled from the others. As shown in Fig. 8.3b, only one of the tiles needs to operate at 1 V, while the supply voltage of the others is reduced to 0.8 V. The energy consumption of this solution drops to 7.5 mJ, which represents about 29 % savings.

The network can be further partitioned into three islands, as shown in Fig. 8.3c. For this example, a finer partitioning results in a diminishing rate of returns. In addition, the overhead of the extra island undermines the potential for energy savings. In this example, the energy consumption of the three- and four-island networks is 7.6 and 7.8 mJ, respectively. Hence, the network partitioning shown in Fig. 8.3b results in a minimum energy consumption.

8.3.4 Partitioning Methodology

We solve the VFI partitioning and static voltage assignment problems simultaneously. We start off with a VFI partitioning where all the PEs belong to separate islands, as shown in the left-most box in Fig. 8.4. Then, we solve the static voltage assignment problem in Sect. 8.3.2. To solve this nonlinear inequality constrained problem, i.e., to find the supply and threshold voltages that minimize the total energy consumption subject to performance constraints, we use the *fmincon*[1]

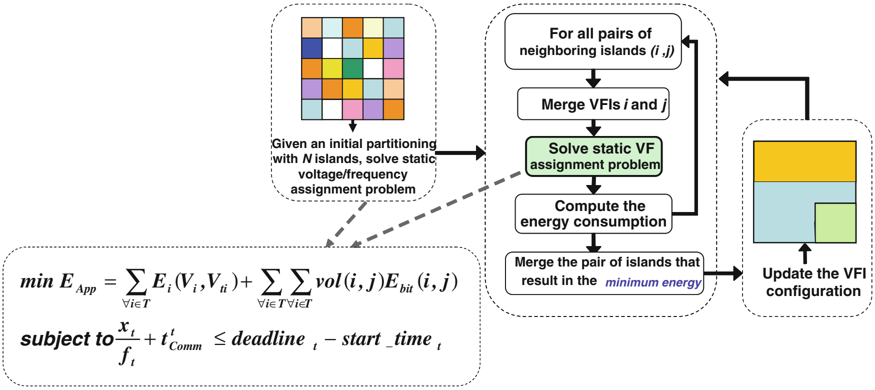

Given an initial partitioning with N islands, solve static voltage/frequency assignment problem
For all pairs of neighboring islands (i,j)
Merge VFIs i and j
Solve static VF assignment problem
Compute the energy consumption
Merge the pair of islands that result in the minimum energy
Update the VFI configuration
$min\ E_{App} = \sum_{\forall i \in T} E_i(V_i, V_{ti}) + \sum_{\forall i \in T} \sum_{\forall i \in T} vol(i,j) E_{bit}(i,j)$
subject to $\frac{x_t}{f_t} + t^t_{Comm} \leq deadline_t - start\_time_t$

Fig. 8.4 Outline of the proposed VFI partitioning and static voltage assignment methodology

function which is available in Matlab. While the configuration where each tiles belong to separate islands minimizes the *application* energy consumption, the large number of VFIs results in a more complex system with a larger energy overhead; hence a design trade-off.

The number of VFIs, hence system complexity, is decreased by merging some of the neighboring islands. Merging islands brings a number of additional constraints to the problem formulation. For instance, when tiles i and j are merged, the constraints $V_i = V_j$ and $V_{ti} = V_{tj}$ need to be added and thus, the voltage assignment problem in Sect. 8.3.2 needs to be solved with these additional constraints. Due to these additional constraints, the application energy consumption goes up after the two islands are merged. However, the *total* energy consumption given by Eq. 8.10 may decrease, if the increase in the application energy consumption is smaller than the reduction in the energy overhead due to merging two VFIs.

In the second step of the algorithm (i.e., the middle box in Fig. 8.4), the decrease in the total energy consumption as a result of merging each pair of neighboring tiles is computed. Then, the pair of islands which improves the total energy consumption the most is picked and merged permanently, as depicted in the right most box in Fig. 8.4. After the VFI configuration is updated, the second step is executed again to find the next pair of candidate tiles to merge. This iterative process continues until all tiles are merged and the network consists of a single island.

In summary, the proposed algorithm starts off with N VFIs and reaches a single VFI configuration at the end; as such, it evaluates *all* possible levels of VFI granularity. If there is no design-specific constraint on the maximum number of VFIs, then the VFI partitioning that provides the minimum energy is selected. On the other hand, if there is a constraint, say the maximum number of VFIs is $M < N$, then we select the VFI partitioning that provides the minimum energy among M-

[1] *fmincon* is a function provided by Matlab that solves constrained nonlinear optimization (or nonlinear programming) problems [26].

VFI, $(M-1)$-VFI,…, 1-VFI solutions, since we cannot have more than M VFIs, even if this would provide a lower energy consumption.

For a $d \times d$ grid with N nodes (i.e. $N = d \times d$) there are $2d(d-1)$ pairs of neighbors. In the worst case, the proposed approach evaluates $O(N^2)$ merges of neighboring islands. In our current implementation, we invoke the nonlinear problem solver (i.e., *fmincon*) for each evaluation, and still obtain the solution in less than 30 min for a 5 × 5 network, as explained in Sect. 8.5.

We also note that one can trade-off accuracy for run-time by employing a simpler evaluation mechanism. For example, when we merge VFI $i(V_i, V_{ti})$ with VFI $j(V_j, V_{tj})$, the supply and threshold voltages of the merged VFI can be set as $V = max(V_i, V_j)$ and $V_{th} = min(V_{ti}, V_{tj})$. This assignment will satisfy the deadline constraints, but it results in a larger energy consumption compared to solving the optimization problem rigorously at each step. Nevertheless, it may significantly reduce the run-time complexity. Finally, the proposed methodology can be used with other nonlinear problem solvers and approximation algorithms with different run-time/accuracy trade-offs [29, 34, 37].

8.4 Feedback Control of Voltage and Frequency

The optimization approach presented in the previous section depends on the nominal values of the parameter known at the design time. At *run-time*, however, there may be performance mismatches between different domains due to parameter and workload variations; e.g., one of the domains may end up working unnecessarily fast. Our goal here is to design a feedback control system such that the voltage and frequency values are dynamically-controlled around the nominal values to cope with such dynamic workload variations.

Consider an NoC (architecture) with two VFIs, as shown in Fig. 8.6 Suppose that the arrival rate at the input of q_1(i.e., $\bar{\lambda}_1$) increases due to a workload variation in VFI 1 (the upper half of Fig. 8.6). In turn, this would result in an increase in the utilization of q_1. If the interface queues (q_1 and q_2 in this case) are controlled independently, the response of the controller would be to increase the speed of VFI 2 to match the workload imposed by VFI 1. However, this would in turn increase the arrival rate at the input of q_2(i.e., $\bar{\lambda}_2$) and trigger an increase in the speed of VFI 1, which causes a positive feedback. Therefore, we develop a state-space model for the system and propose a feedback controller to set the operating voltage and frequency of each island, as shown in Fig. 8.5.

8.4.1 State-Space Feedback Control

Changing the operating frequency and transmitting control and feedback information over the network requires a non-negligible amount of time [20]. Therefore,

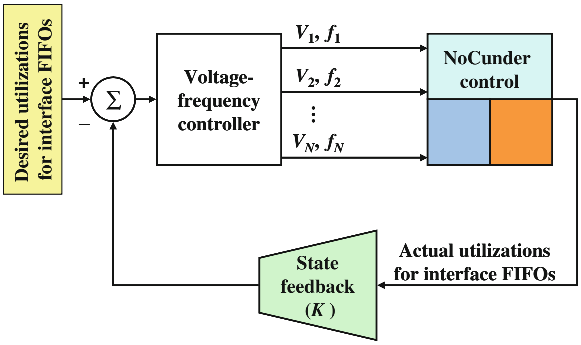

Desired utilizations for interface FIFOs
+
–
Σ
Voltage-frequency controller
V_1, f_1
V_2, f_2
⋮
V_N, f_N
NoC under control
State feedback (K)
Actual utilizations for interface FIFOs

Fig. 8.5 Closed loop control of voltage/frequency levels based on state feedback, where the state reflects the utilization of the FIFO queues at the VFI boundaries

the controller in Fig. 8.5 needs to be activated with a period, denoted as T, which is large enough to perform these operations. The utilization of the interface queue i in the beginning of kth control interval (i.e., $[kT, (k+1)T]$) is expressed as $q_i(k)$.

For the kth control interval, the operating frequency of VFI i is denoted by $f_i(k)$. We assume that the arrival and service rates inside each control interval are stationary and proportional to the operating frequency $f_i(k)$. So, the average arrival and service rates for q_1 in Fig. 8.6 can be expressed as:

$$\lambda_1(k) = \bar{\lambda}_1 \times f_1(k), \quad \mu_1(k) = \bar{\mu}_1 \times f_2(k)$$

where $\bar{\lambda}_1$ and $\bar{\mu}_1$ are the average arrival and service rates, respectively. Consequently, the dynamics of q_1 is given by:

$$q_1(k) = q_1(k-1) + T[\bar{\lambda}_1 - \bar{\mu}_1]\begin{bmatrix} f_1(k-1) \\ f_2(k-1) \end{bmatrix} \tag{8.11}$$

In general, q_1 saturates at $q_1(k) = 0$ and $q_1(k) = B$, where B is the depth of the queue, due to queue finite size. The proposed feedback controller stabilizes the queue around a target utilization. As such, this equation represents the linearized dynamics around an operation point.

From a practical standpoint, however, we need to generalize Eq. 8.11 to consider multiple queues. Suppose that our VFI partitioning technique discussed in

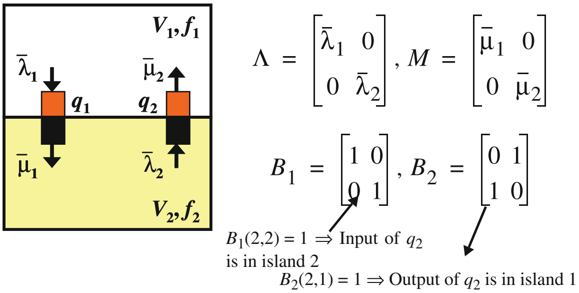

V_1, f_1
$\bar{\lambda}_1$
$\bar{\mu}_2$
q_1
q_2
$\bar{\mu}_1$
$\bar{\lambda}_2$
V_2, f_2
$\Lambda = \begin{bmatrix} \bar{\lambda}_1 & 0 \\ 0 & \bar{\lambda}_2 \end{bmatrix}, M = \begin{bmatrix} \bar{\mu}_1 & 0 \\ 0 & \bar{\mu}_2 \end{bmatrix}$
$B_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, B_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
$B_1(2,2) = 1 \Rightarrow$ Input of q_2 is in island 2
$B_2(2,1) = 1 \Rightarrow$ Output of q_2 is in island 1

Fig. 8.6 Illustration of the state-space model for a network with 2 VFIs and 2 interface queues

Sect. 8.3 results in M VFIs. When there are M islands and S interface queues between them, the state of the network (i.e., the utilizations of the interface queues) and the input vector can be defined as:

$$Q(k) = [q_1(k), q_2(k), \ldots, q_S(k)]^T, F(k) = [f_1(k), f_2(k), \ldots, f_M(k)]^T$$

Consequently, the state-space of the *collective* queue dynamics is given by:

$$Q_k = [Q(k-1)]_{S\times 1} + T[(\Lambda B_1 - MB_2)]_{S\times M}[F(k-1)]_{M\times 1} \tag{8.12}$$

where $\Lambda = diag(\bar{\lambda}_1, \bar{\lambda}_2, \ldots, \bar{\lambda}_S)$ and $M = diag(\bar{\mu}_1, \bar{\mu}_2, \ldots, \bar{\mu}_S)$ and the $S \times M$ matrices B_1 and B_2 define the topology of the VFI configuration. i.e., B_1 and B_2 specify the islands to which the input and output sides of the interface queues are connected, respectively. For example, $B_1(i,j) = 1$ implies that the *input* side of interface queue i is in island j, as illustrated in Fig. 8.6. Likewise, $B_2(i,j) = 1$ means that the *output* side of interface queue i is in island j.

When we use state feedback, the frequency input can be expressed as $F(k) = K_0 Q_{ref}(k) - KQ(k)$, where K is the state feedback matrix, $Q_{ref}(k)$ is the reference queue utilization and K_0 is a gain matrix which ensures that

$$\lim_{k\to\infty} Q(k) = Q_{ref}(k)$$

Then, the closed loop system can be obtained by rewriting Eq. 8.12 as:

$$Q(k) = (I - T(\Lambda B_1 - MB_2)K)Q(k-1) + T(\Lambda B_1 - MB_2)K_0 Q_{ref}(k-1) \tag{8.13}$$

The state feedback matrix K needs to be selected such that the closed loop system described by Eq. 8.13 is stable. We also note that average queue utilizations could also be used as feedback with some manipulations [41]. Utilizing the rich set of techniques offered by the state-space feedback control theory [32], this can be achieved by placing the eigenvalues of $(I - T(\Lambda B_1 - MB_2)K)$ within the unit circle.

Finally, we note that pole placement for this closed loop system is possible when the open loop system described by Eq. 8.12 is controllable. The necessary conditions for controllability are as follows:

Theorem *In the multiple voltage-frequency island system with M islands (i.e., M independently controllable clocks) described by Eq.* 8.12, *the utilization of at most M queues at the interface of VFIs can be controlled and the system is controllable iff* $rank(T(\Lambda B_1 - MB_2)) = S$.

Proof The proof follows from the analysis of the model in Eq. 8.12. Let $T(\Lambda B_1 - MB_2) = B$ for notational simplicity. Then, the controllability matrix U for this system is:

$$U = [B|IB|I^2B|\ldots|I^{S-1}B]$$

where I is the $S \times S$ identity matrix. The rank of U is obviously equal to the rank of B which is of size $S \times M$:

$$rank(U) = rank(B) \leq min(M, S)$$

We know that the system is controllable *iff* $rank(U) = S$ [32]. So, the system can be *state controllable* only if $S \leq M$, i.e., the number of queues under control is less than or equal to the number of VFIs.

Furthermore, since $rank(U) = rank(B)$, the system *is state controllable iff* $rank(B) = S$. □

According to this result, the number of interface queues that can be controlled is less than or equal to the number of VFIs, i.e. the number of independent control inputs. In case there are more interface queues than the number of VFIs, a subset of queues should be selected such that the second condition is satisfied. As a result, we dynamically scale the frequency and voltage of the islands based on the utilizations of the controlled queues. This online feedback control strategy adjusts the static voltage/frequency values found in Sect. 8.3 to dynamic workload variation and parameter variations which are unknown at the design time.

8.5 Experimental Results

This section illustrates the effectiveness of the proposed VFI partitioning and dynamic power management methodologies in minimizing the energy consumption using real benchmarks. The first set of benchmarks is chosen from a public benchmark suite, while the second consists of a real video application. The energy related parameters (e.g., energy consumption of a task running on a certain PE) are derived from the benchmarks, while the technology parameters are taken from [25].

After the proposed algorithm is used to find the supply and threshold voltage for each VFI, we map them conservatively to the following discrete levels: $V_{supply} = \{0.4, 0.6, 0.8, 1.01.2\,\mathrm{V}\}$, $V_T = \{0.15, 0.20, 0.25, 0.30, 0.35\,\mathrm{V}\}$. Then, we compute the *total* energy consumption using Eq. 8.10. The results reported hereafter are obtained for these discrete levels.

8.5.1 Experiments with Realistic Benchmarks

Consumer, networking, *auto-industry* and *telecom* benchmark applications are collected from E3S suite [13]. These benchmarks are scheduled onto 3 × 3, 3 × 3, 4 × 4 and 5 × 5 mesh networks, respectively using the EAS scheme presented in [18]. Then, the proposed approach is used for VFI partitioning and static voltage assignment. The second column of Table 8.1 (“**1-VFI**”) shows the minimum energy consumption when the entire network consists of a single island. The remaining columns show the energy consumption values obtained for the best partitioning with two and three islands, respectively. The energy consumption of the partitioning selected by the algorithm is marked with an asterisk. For the *Consumer* benchmark, the minimum energy consumption is obtained when the network is partitioned into two voltage-frequency islands. With this configuration, the energy consumption drops from 18.9 to 12.1 mJ, which represents about 36 % improvement. As shown in Table 8.1, partitioning the network at a finer granularity does *not* reduce the energy consumption further due to the overhead of having the extra islands, which is about 1.7 mJ. Similarly, a 2-VFI configuration achieves the minimum energy for *network* benchmark. The *auto-industry* benchmark has 1.67 mJ energy consumption with no VFI partitioning. For this case, partitioning the network into two islands decreases the energy consumption to 0.34 mJ. Finally, generating more VFIs does not decrease the energy consumption further. For *telecom* benchmark, the energy consumption of the partitioning with the proposed algorithm is 1.5 mJ; this is more than 4× reduction compared to the 1-VFI case.

For better visualization, we show the supply voltage levels obtained for *telecom* benchmark in Fig. 8.7. When there is a single voltage domain, all tiles operate at 1 V V_{DD} and 0.15 V threshold voltage. However, when there are two VFIs, the supply voltage of all the tiles except tile (2,3) can be lowered to 0.6 V, while the

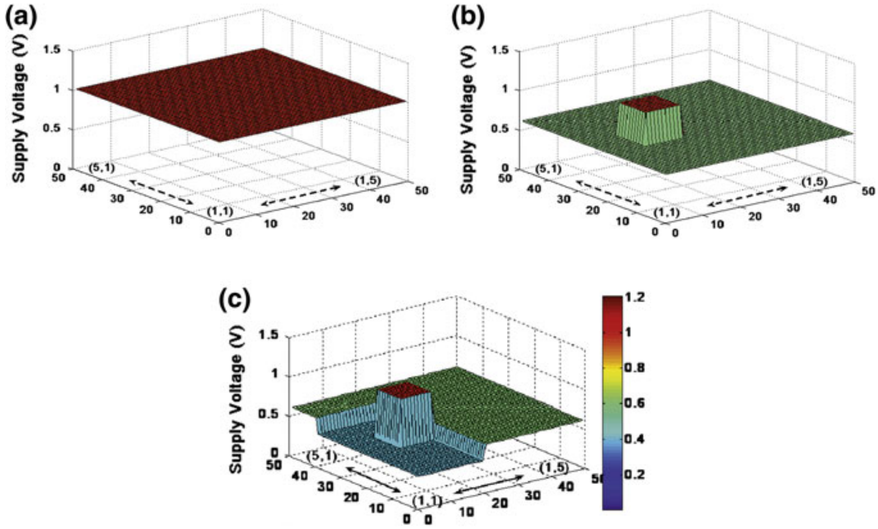


Fig. 8.7 Different voltage-frequency island partitionings and corresponding static voltage assignments for benchmark 3 in Table 8.1. **a** Single VFI. **b** Two VFI partitioning. **c** Three VFI partitioning

Table 8.1 The reduction in the overall energy consumption obtained as a result of the control algorithm

Benchmark	Network size	Total energy consumption (mJ)		
		1-VFI	2-VFI	3-VFI
Consumer	3 × 3	18.9	12.1*	12.2
Network	3 × 3	12.9	6.6*	6.7
Auto-industry	4 × 4	1.67	0.34*	0.40
Telecom	5 × 5	6.9	2.6	1.5*

threshold voltages remain at 0.15 V. When we increase the number of VFIs to three, the tiles voltage on the lower left corner is further reduced to 0.4 V.

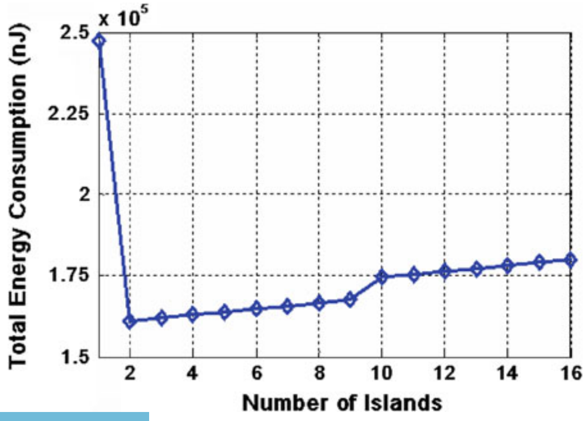
The run-time of the algorithm ranges from a few tens of seconds to 30 min for the benchmarks reported in Table 8.1. In general, the proposed approach does *not* guarantee the optimal solution due to the use of a nonlinear problem solver.

8.5.2 Experiments with a Real Video Application

We analyzed a video application consisting of MPEG2/MP3 encoder/decoder pairs [18]. The application is partitioned into a set of tasks and scheduled onto a 4 × 4 mesh network using the EDF scheme. Then, the proposed algorithm is used to find the VFI partitioning with minimum energy consumption.

The proposed approach starts with 16 separate VFIs. Then, it proceeds by merging the islands until a single island is obtained. As shown in Fig. 8.8, the total energy consumption is improved until we reach two islands. For instance, when we move from 16 to 15 islands, the increase in the application energy consumption is negligible. So, the total energy consumption reduces due to the smaller overhead. Finally, a 2-VFI partitioning where the tasks of the same application reside in

Fig. 8.8 The variation of total energy consumption as a function of the number of voltage-frequency islands



different islands achieves the minimum energy consumption. This resulting partitioning results in 40 % improvement compared to the 1-VFI case.

For the benchmarks we consider, two or three VFIs were usually sufficient to minimize the energy consumption in our experiments. The exact number of VFIs depends on (1) how much we can gain by further partitioning a network and (2) how much overhead should we pay to afford the extra overhead. The former is a function of the application(s) running on the network. For the benchmarks we consider, partitioning the NoC beyond two-three VFIs did not result in significant energy savings. However, the proposed approach is general enough to exploit other types of applications. The latter factor (i.e., (2)), is technology and design dependent. If the cost of additional VFIs is small, designs with more VFIs can be more beneficial.

8.5.3 Evaluation of the Feedback Control Strategy

In this section, we illustrate the operation of the state-space feedback controller. The control system is simulated using Simulink® where the mixed-clock FIFOs (of depth 16 words) instantiated using the System Generator from tool Xilinx [39] are used to implement the interface queues. The average utilization of the FIFOs over a control interval is used as the feedback. For voltage conversion, we used the model from [6] with 0.9 efficiency and 10 μF load capacitance. Since, the voltage transitions may take more than 10 μs [21, 42], we conservatively set the control interval, $T \geq 100\,\mu\text{s}$. This duration is sufficiently large to change the operating voltage/frequency and exchange feedback and control signals over the network. We note that these control signals may also have a high priority or use a network protocol that guarantees the on-time delivery. Furthermore, these signals are sent very regularly at periodic intervals. So, the time needed for computation and communication of control signals is not critical.

In order to assess the performance of the feedback controller, we vary the arrival and service rates of the interface queues around their nominal values. A deviation from the nominal value implies that the corresponding island operates faster or slower than its expected speed.

The operation of the proposed feedback controller for the *consumer* benchmark (the second row in Table 8.1) is depicted in Fig. 8.9. As shown in Table 8.1, the optimum energy savings is achieved when this design is partitioned into two islands. In this example, the traffic flow is from VFI 1 to VFI 2. We computed the state-feedback matrix K such that the eigenvalues of the closed loop system stay within the unit circle even when the elements of Λ and $\mathbf{M}$ in Eq. 8.12 vary by $\pm 25\,\%$ around their nominal values. This means that the average arrival and service rates of the interface queues can be larger (or smaller) than the nominal values by as much as 25 % due to dynamic workload variations. During simulations, we added uniformly distributed random variables to each element of Λ and $\mathbf{M}$ such that $0.75\bar{\lambda}_{i\ nominal} \leq \bar{\lambda}_1 \leq 1.75\bar{\lambda}_{i\ nominal}$ (same for μ_i).

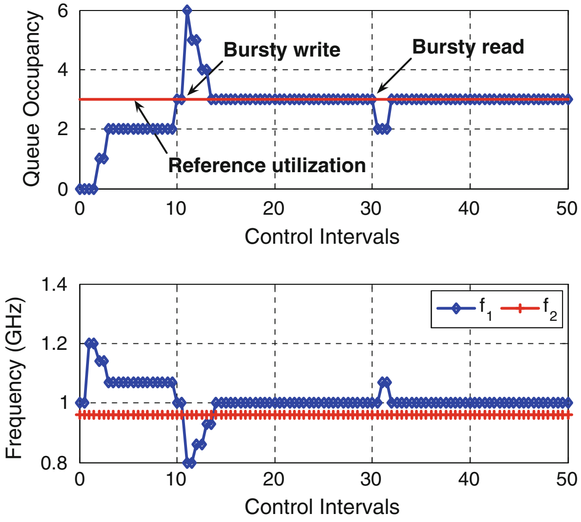


Fig. 8.9 The operation of the state-space feedback controller in the presence of random workload variations (within 25 % of the nominal value) and bursty read/ write operations. Each control interval is 100 µs

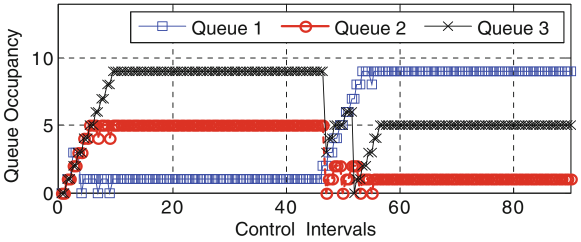


Fig. 8.10 The response of the proposed controller when the reference queue utilizations are changed during the normal operation

As depicted in Fig. 8.9, the controller successfully stabilizes the interface queue. We also simulate bursty write and read operations. For example, at the beginning of the 11th control interval, VFI 1 starts transmitting data in a bursty manner and saturates the queue. We observe that the controller responds by slowing f_1 down, and the utilization settles down to the desired value after a few control intervals. Likewise, we observe that the controller responds to bursty reads by increasing f_1 (see 30th control interval in Fig. 8.9).

In summary, we observe that the controller adapts the operation speed of the islands smoothly to counteract random workload variations and sudden changes in the queue utilization. In both cases, the queue utilization and operating frequencies settle down to their reference points. Finally, we note that the design of the controller is independent of the target queue utilizations. The target utilizations can be fixed or adjusted at run-time to obtain a trade-off between power consumption and performance, as shown in Fig. 8.10.

These findings have been validated using an FPGA prototype and found to be consistent with the real measurements. The details of the prototype and experiments are presented in Appendix A.6.

8.6 Extensions of Basic Theory

The technique discussed in this chapter enables formal analysis and optimization of system level dynamic power management by introducing a precise mathematical system model. This formalism enables theoretical extensions in a number of interesting ways.

For example, the centralized algorithm described herein can benefit from exploiting small world effects with a trade-off between controller performance and implementation cost trade-off [16]. Likewise, it is worthwhile to analyze the impact of technology driven constraints such as manufacturing process variations, reliability driven limits on the supply voltage on the performance of DVFS control for multiple VFI MPSoCs [16]. One can also consider non-stationary workloads and design optimal controllers under more general workload models [4].

Finally, the control-theoretic formalism for power management can be extended to also consider thermal management [1, 10, 17, 33, 44] based approaches have been recently proposed for effective thermal management.

8.7 Summary

In this chapter, we have discussed a methodology for multi-clock, multi-frequency domain NoC design, and presented an algorithm for voltage-frequency island partitioning and supply/threshold voltage assignment. We have shown that using VFIs in the NoC context provides better power-performance trade-offs than its single voltage, single clock frequency counterpart, while taking advantage of the natural partitioning and mapping of applications onto the NoC platform. Finally, we have also shown how this VFI-based architecture can be precisely controlled for minimizing power dissipation at run-time.

References

1. Arjomand M, Sarbazi-Azad H (2010) Voltage-frequency planning for thermal-aware, low-power design of regular 3-D NoCs. In: 23rd international conferene on VLSI design
2. Bertozzi D et al (2005) NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. IEEE Trans Parallel Distrib Syst 16(2):113–129
3. Bjerregaard T, Sparso J (2005) A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In: Proceedings of design, automation and test in Europe conference, March 2005
4. Bogdan P, Marculescu R (2010) Workload characterization and its impact on multicore platform design. In: Proceedings of the 8th IEEE/ACM/IFIP international conferene on hardware/software codesign and system synthesis (CODES/ISSS)
5. Butts JA, Sohi GS (2000) A static power model for architects. In: Proceedings of international symposium of microarchitecture, Dec 2000

6. Burd TD, Brodersen RW (2000) Design issues for dynamic voltage scaling. In: International symposium on low power electronics and design
7. Campobello G, Castano M, Ciofi C, Mangano D (2006) GALS networks on chip: a new solution for asynchronous delay-insensitive links. In: Proceedings of design, automation and test in Europe conference, March 2006
8. Chelcea T, Nowick SM (2000) A low latency fifo for mixed-clock systems. In: Proceedings of IEEE computer society workshop on VLSI, April 2000
9. Chapiro DM (1984) Globally asynchronous locally synchronous systems. PhD thesis, Stanford University
10. Coskun AK et al (2010) Energy-efficient variable-flow liquid cooling in 3D stacked architectures. In: Proceedings of design automation and test in Europe, pp 1–6
11. Dasgupta S, Yakovlev A (2007) Comparative analysis of GALS clocking schemes. IET Comput Digit Tech 1(2):59–69
12. Dhillon YS, Diril AU, Chatterjee A, Lee HS (2003) Algorithm for achieving minimum energy consumption in CMOS circuits using multiple supply and threshold voltages at the module level. In: Proceedings of international conference on computer aided design, Nov 2003
13. Dick R Embedded system synthesis benchmarks suites (E3S). http://ziyang.eecs.umich.edu/~dickrp/e3s/
14. Dielissen J, Radulescu A, Goossens K, Rijpkema E (2003) Concepts and implementation of the philips network-on-chip. IP-based SoC design
15. Duarte DE, Vijaykrishnan N, Irwin MJ (2002) A clock power model to evaluate impact of architectural and technology optimizations. IEEE Trans Very Large Scale Integr Syst 10(6): 884–855
16. Garg S, Marculescu D, Marculescu R (2010) Custom feedback control: enabling truly scalable on-chip power management for MPSoCs. In: Proceedings of the ACM/IEEE international, symposium on low power electronics and design, Austin, TX
17. Ge Y, Malani P, Qiu Q (2010) Distributed task migration for thermal management in many-core systems. In: Design automation conference, Anaheim, June 2010
18. Hu J, Marculescu R (2005) Communication and task scheduling of application-specific networks-on-chip. IEE Proc Comput Digit Tech 152(5):643–651
19. Hu J, Marculescu R (2005) Energy- and performance-aware mapping for regular NoC architectures. IEEE Trans Comput Aided Des Integr Circuits Syst 24(4):551–562
20. Intel Corp. Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor. http://download.intel.com/design/network/papers/30117401.pdf. Accessed March 2004
21. Intel architecture. http://www.intel.com/pressroom/kits/core2duo/pdf/microprocessor_timeline.pdf
22. Juang P, Wu Q, Peh L, Martonosi M, Clark D (2005) Coordinated, distributed, formal energy management of chip multiprocessors. In: Proceedings of the ISLPED, Aug 2005
23. Lackey DE, Zuchowski PS, Bednar TR, Stout DW, Gould SW, Cohn JM (2002) Managing power and performance for system-on-chip designs using voltage islands. In: Proceedings of international conference on computer aided design, Nov 2002
24. Magklis G, Semeraro G, Albonesi DH, Dropsho SG, Dwarkadas S, Scott ML (2003) Dynamic frequency and voltage scaling for a multiple clock domain microprocessor. IEEE Micro Special Issue: Top Picks Comput Archit 23(6):62–68
25. Martin S, Flautner K, Mudge T, Blaauw D (2002) Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In: Proceedings of international conference on computer aided design, Nov 2002
26. Matlab ® Documentation Optimization Toolbox, fmincon. http://www.mathworks.com/
27. Millberg M, Nilsson E, Thid R, Kumar S, Jantsch A (2004) The Nostrum backbone—a communication protocol stack for networks on chip. In: Proceedings of VLSI design, Jan 2004
28. Muttersbach J, Villager T, Fichtner W (2000) Practical design of globally asynchronous locally synchronous systems. In: Proceedings of international symposium on advanced research in asynchronous circuits and systems, April 2000

29. Nash SG, Sofer A (1996) Linear and nonlinear programming. McGraw-Hill, New York
30. National Semiconductor Corporation (2004) Next-generation SoC power management with multi-domain adaptive voltage scaling. Electronics product design, March 2004
31. Niyogi K, Marculescu D (2005) Speed and voltage selection for GALS systems based on voltage/frequency islands. In: Proceedings of Asia and South Pacific design automation conference, Jan 2005
32. Ogata K (1995) Discrete-time control systems. Prentice-Hall, Upper Saddle River
33. Sharifi S et al (2010) Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs. In: Asia and south pacific design automation conference, pp 873–878, Jan 2010
34. Rasmussen J (1998) Nonlinear programming by cumulative approximation refinement. Struct Multidiscip Optim 15(1):1–7
35. Quartana J, Renane S, Baixas A, Fesquet L, Renaudin M (2005) GALS systems prototyping using multiclock FPGAs and asynchronous network-on-chips. In: Proceedings of international conference on field programmable logic and applications, Aug 2005
36. Sakurai T, Newton AR (1990) Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. IEEE J Solid-State Circuits 25(2):584–594
37. Schittkowski K (1986) NLPQL: a fortran subroutine solving constrained nonlinear programming problems. Ann Oper Res 5(1–4):485–500
38. Semiconductor Association (2006) The international technology roadmap for semiconductors (ITRS)
39. System Generator®, Xilinx®. http://www.xilinx.com/ise/optional_prod/system_generator.htm
40. Toshiba America Electronic Components, Inc. Impact of multiple-voltage domain (Multi-VDD) design implementation on large, complex SoCs. White paper. http://www.toshiba.com/taec/adinfo/socworld/images/Pointers_Pitfalls_MultiVDD.pdf
41. Wu Q, Juang P, Martonosi M, Clark DW (2004) Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In: Proceedings of the international conference on architectural support for programming languages and operating systems, Oct 2004
42. Xie F, Martonosi M, Malik S (2004) Intraprogram dynamic voltage scaling: bounding opportunities with analytic modeling. ACM Trans Archit Code Optim 1(3):323–367
43. Ye T, Benini L, De Micheli G (2002) Analysis of power consumption on switch fabrics in network routers. In: Proceedings of design automation conference, June 2002
44. Zanini F, Atienza D, De Micheli G (2009) A control theory approach for thermal balancing of MPSoC. In: Proceedings of the Asia and south pacific design automation conference

Chapter 9
Conclusion

Continuous technology scaling will soon enable multicore designs with thousands of communicating IP blocks on a single chip. Successful design of systems at this scale will depend critically on truly scalable communication architectures. The promising solution to date is given by the structured communication via the NoC approach.

In this book, we have discussed a few fundamental issues related to the modeling, analysis and optimization of NoC communication architectures, and presented novel design methodologies for low-power NoC design. Formal models are instrumental not only to understand and analyze the behavior of NoCs but also to design optimal systems. The techniques presented herein, as well as the literature review, aim at providing a deep understanding of this field and prepare readers for further research.

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1_9,
© Springer Science+Business Media New York 2013

Appendix A
Tools and FPGA Prototypes

This appendix discusses the NoC analysis, optimization and evaluation tools used while preparing this book. We first present the NoC performance analysis, architecture optimization and voltage-frequency island synthesis tools. Then, we present the simulator used to evaluate these tools. Finally, we provide a brief overview of the NoC prototypes implemented to demonstrate and evaluate our major contributions.

A.1 NoC Analysis and Optimization Tools

A.1.1 NoC Performance Analysis Tool

The NoC performance analysis tool is coded using C++. The tool uses a simple command line interface to setup different parameters. These parameters are:

- Network architecture and routing algorithm: The architectures is accepted using the "architecture-config" format. This format specifies the number of routers, their interconnection and the routing algorithm. A configuration file can be generated in two ways. The first way is to generate a configuration file given a standard mesh network of any size and a standard deterministic routing algorithm, such as XY routing. We developed a separate utility for this purpose. The second option is to use our tool for the NoC performance optimization via long-range link insertion. This format is also supported by the *worm_sim* simulator [11] which is used to evaluate the performance of our analysis tool.
- Number of virtual channels
- Depth of the input buffers in the router
- Target application: the tool accepts the application in the "traffic-config" format. In this format, each line in the input file contains a 3-tuple, which

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1,
© Springer Science+Business Media New York 2013

specifies the source IP, the destination IP and the packet rate from the source to the destination. This format is also supported by *worm_sim* simulator [11] and it can be generated from "APCG format (see Sect. 4.3) via *worm-sim*.

A.1.2 NoC Architecture Customization via Long-Range Links

The long-range link insertion tool is also developed using C++. This tool accepts the application description in the same way as the performance analysis tool described in Appendix A.1.1. In addition to this, the size of the initial mesh network and the default routing algorithms are taken as command line inputs. Finally, the tool takes the maximum total length of the long-range links that can be added to the initial mesh network. The output is an configuration file in the "architecture-config" format. As mentioned before, this format specifies the number of routers, their interconnection and the routing algorithm. The output file can be used for the NoC performance analysis tool and *worm_sim* simulator for further evaluation.

A.2 Simulator Support

A.2.1 Worm_sim NoC Simulator

We utilized a cycle-accurate NoC simulator, called *worm_sim* to evaluate the techniques presented in this book. *Worm_sim* was developed from scratch in C++ using a standard template library by Jingcao Hu [11]. The initial version of *worm_sim* is capable of simulating mesh and torus topologies under various routing algorithms. The user controllable performance parameters include channel buffer size, routing engine delay, crossbar arbitration delay, etc. *Worm_sim* can simulate the system under standard traffic patterns, such as uniform, transpose, and hotspot traffic patterns, as well as application-specific traffic. specified by application configuration files or traces. Besides reporting average packet latencies, *worm_sim* also reports communication energy consumption using the Ebit model [13] and the Orion power model library [10].

- Besides the standard mesh and torus topologies, the current version accepts mesh or torus topologies with long-range links. Furthermore, any deterministic routing algorithm is accepted in the form of a routing table. This enables us to simulate arbitrary topologies with any deterministic routing.[1]

[1] Link insertion is done explicitly using long-range links, while link removal is mimicked by never using certain links.

- The flow control technique presented in Chap. 7 is implemented in *worm_sim*. Besides the average packet latency reported by the initial version, the extended version more detailed performance reports such as average and maximum number of packet in the network at a given time, the average delay experience at each router and the input buffer utilizations.
- We have developed two new mechanisms to generate traffic in *worm_sim*. First, we implemented ON–OFF traffic sources presented in Sect. 7.4 in *worm_sim*. Besides this, we have developed a tool that generates finite state machines (FSM) to describe the behavior of each traffic generator. This way, the simulator can capture the control and data dependencies in the target application and generate more realistic traffic patterns than random traffic.

A.3 On-Chip Router Prototype

On-chip routers are at the heart of NoC designs. Therefore, we designed an output buffered on-chip router which is shared across all prototypes developed in this dissertation. Due to its moderate buffer requirements, our design (whose simplified block diagram is shown in Fig. A.1) implements wormhole flow control. The router consists of four pipeline stages; hence, it takes four cycles to route the header flit. Then, the remaining flits of the packet, which can vary from 1 flit to 255 flits, follow the header in a pipelined fashion. The depth and width of the output buffers are parameterized. The packets in the network are divided into 16-bit flits, since the width of the channels is 16 bits. The router supports deterministic routing, the routing strategy being implemented as a lookup table for flexibility reasons. Finally, based on the network topology and their location in the network, the routers may have different number of ports and area, as summarized in Table A.1.

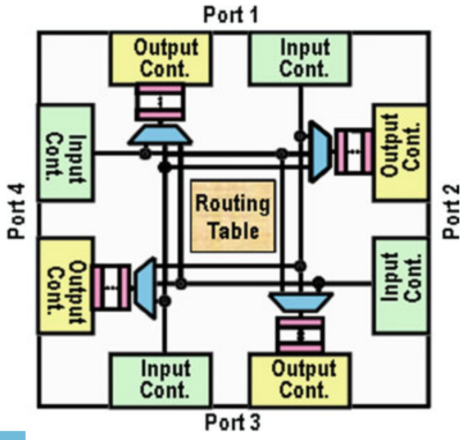


Fig. A.1 A simplified block diagram of the on-chip router

Table A.1 Impact of inserting of long-range links on area

	Number of slices	Device utilization (%)
3-port router	219	1.4
4-port router	304	1.8
5-port router	397	2.2
6-port router	503	2.8
4×4 mesh network	6683	29.0
Mesh with long-range links	7152	31.0

Synthesis is done for Xilinx Virtex$^{\text{TM}}$-II XC2V4000 FPGA

The router was implemented using Verilog HDL. Standard FIFOs from Xilinx IP library are used to implement the output buffers, while the remaining modules are custom designs. To test the functionality of the routers, a 4 × 4 mesh network is instantiated. Then, random number generators are attached to each router to generate random traffic. Finally, the resulting network is simulated using ModelSim to verify that all the packets reach successfully their destinations. More details of this prototype can be found in [6, 8].

A.4 NoC with Application-Specific Long-Range Links

To further demonstrate the effectiveness of the long-range link insertion methodology, we present an FPGA prototype using a Xilinx Virtex$^{\text{TM}}$-II XC2V4000 device. We first connected the on-chip routers to compose a 4 × 4 mesh network using Verilog HDL. After the operation of the network is tested by hardware simulation performed using ModelSim, application-specific long-range links for hotspot traffic pattern and various benchmarks from E3S suite [2] are determined. To implement the network with long-range links, we replaced the routers used in the original mesh network with 6-port routers whenever necessary, and inserted the long-range links. This process is accomplished in less than a day by modifying the top level Verilog module describing the mesh network. Similarly, the testbench used for the mesh network is reused to test the functionality of the network with long-range links. Finally, the designs are synthesized, implemented and downloaded to Xilinx Virtex-II FPGA using Xilinx ISE Foundation.

Inserting long-range links requires more resources, hence, increases the area of the design. Therefore, we analyze the size of the individual routers in a 4 × 4 mesh network and its customized version with long-range links under the constraint of $s(l) = 12$ (Table A.1). We observe that moving from 3-to-4, 4-to-5, and 5-to-6 ports increases the slice utilization by 85, 93 and 106 slices, respectively. Moreover, we observe that the total number of slices used by a 4 × 4 network implementing transpose traffic rises about 7.0 %, as summarized in Table A.1. More details of this prototype can be found in [8].

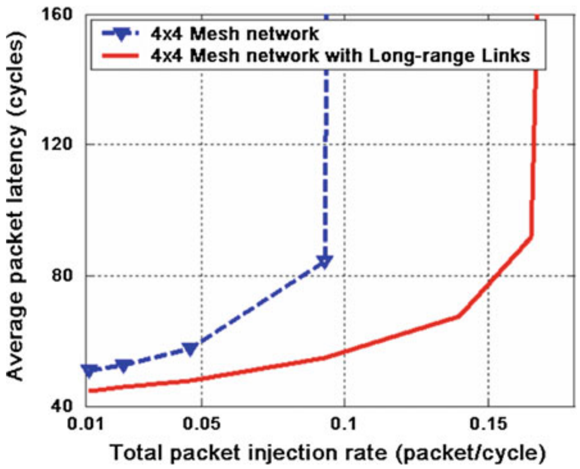


Fig. A.2 Comparison of average packet latency for a 4 × 4 mesh network and mesh network with long-range links obtained with the FPGA prototype for transpose traffic

We observe that the improvements in the average message latency and network throughput measured using the FPGA prototype are consistent with the simulation results. The latency comparison under *transpose* traffic is plotted in Fig. A.2, while further experimental results can be found in [8]. This basically validates our simulation results and offers a solid basis for the newly proposed approach.

We also performed accurate energy measurements on the FPGA prototype using the cycle-accurate power measurement tool develop by Lee et al. [5]. The experiments showed minimal impact on the energy consumption and validated the theoretical expectations and simulation results in Sect. 6.6 [8].

A.5 Implementation Overhead of Flow-Control Algorithm

In order to accurately evaluate the area overhead, we implemented the proposed flow control in Verilog HDL and synthesized the design using Synopsys Design Compiler. The equivalent gate count of the proposed flow controller is found as 1093 gates.

We also integrated the proposed flow controller into an existing router and developed an FPGA prototype based on a Xilinx XC2V3000 platform. The basic NoC router without the proposed flow controller is based on the prototype presented in Appendix A.3. The router implements the basic link-level ON/OFF flow control. The input buffers of the router have 16 flit depth and 16 bit width. The router implements wormhole flow control with deterministic table-based routing.

The router takes four cycles to process the header flit (that is, to receive, make a routing decision, traverse the crossbar switch and place it to the desired outgoing link). After that, the remaining flits simply follow the header in a pipelined fashion. The time it takes to route packets is not affected by the flow controller, since the computation of the availabilities are performed concurrently with routing.

The proposed controller takes up 80 slices in the target FPGA; this corresponds to about 18 % increase in the number of resources used by the router. It is also important to evaluate the overhead of the router in a real design. For instance, the overhead of the proposed controller is about 0.8 % for the MPEG-2 encoder presented in [6]. In general, the overhead of our controller is estimated be about 1 % of the total chip area.

A.6 Validation of VFI-Based NoC via Prototyping

This section presents a FPGA prototype based on Virtex2Pro Xilinx FPGA platform [9] for NoCs with multiple VFIs, and the dynamic frequency control architecture.

A.6.1 Design of NoCs with Multiple VFIs

A typical router in an NoC consists of a FIFO and an output controller (OC) for each port, and an arbiter to channel the traffic between the ports, as depicted in Fig. A.3. To connect a node in a VFI with another node residing in a different VFI, all data and control signals need to be converted from one frequency/voltage domain to another. For this purpose, we implemented mixed-clock/mixed-voltage interfaces using FIFOs, which are natural candidates for converting the signals from one VFI to the another, as shown in Fig. A.3.

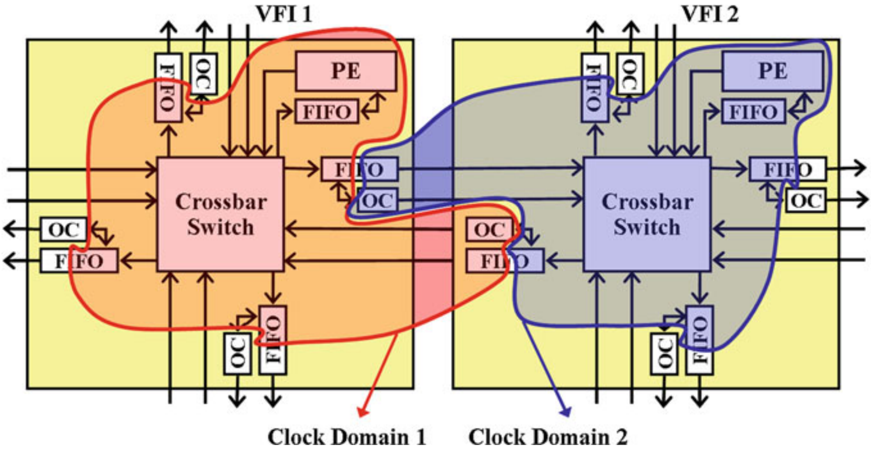


Fig. A.3 Illustration of the interface between two different voltage-frequency domains VFI1 and VFI2

To support the simulation results, we implement a GALS-based NoC with a 4 × 4 mesh topology using Verilog HDL. Block RAM-based mixed-clock FIFOs from the Xilinx library are used in routers to transfer data between different clock domains. Our design can be partitioned into as many as 16 VFIs. In our implementation, the signal conversion, both in terms of clock and voltage domains, occurs at FIFO interfaces. In this particular design, the Delay Locked Loops (DLLs) from the Xilinx FPGA device are used to generate the individual clock signals. However, since multiple voltage levels are not readily available for FPGA platforms, our prototype does not support voltage level conversion.

For experimental purposes, this implementation is configured with 16 islands and simulated using the *auto-industry* benchmark from E3S [2]. We first verify that no packets are lost in the VFI interfaces. After that, we compute the total energy consumption corresponding to single VFI and 2-VFI implementations, as shown in Sect. 8.5.1. To compute the energy consumption values, we utilize the energy characterization of the on-chip routers reported in [6]. The total energy consumption for single VFI operating at 1V is found as 109 nJ. On the other hand, the total energy consumption of the 2-VFI partitioning found using the proposed approach is 21.2 nJ. Hence, we observe about 81 % reduction in the energy consumption. The energy consumption results obtained using the FPGA prototype are different from that measured by simulation in Table 8.1 due to the differences in the target platform and implementation details. Nevertheless, we note that according to the simulation results, the *relative* improvement in the energy consumption for the same benchmark is 80 %, which is very close to the result obtained using the actual prototype.

A.6.2 Dynamic Frequency Control Architecture

The NoC prototype presented in the previous section consists of multiple VFIs operating at different clock frequencies. However, it does not support dynamic frequency scaling. The architecture we present in this section illustrates the dynamic frequency control technique presented in Sect. 8.4.

The dynamic frequency control architecture with three different frequency islands is depicted in Fig. A.4. Delay Locked Loops (DLLs) available on the Xilinx FPGA are used to generate four basic clock signals that are not multiples of each other (see second column in Table A.2). These four clock signals are further divided by the clock control module to generate the clock signals for the islands by the *clock control module* according to the utilization of the interface queues under control (FIFO 1 and FIFO 2, in Fig. A.4).

The *clock control module* is activated periodically, once for every control interval *T*. The dynamic frequency scaling algorithm (e.g., the state-space feedback controller described in Sect. 8.4) is implemented in the *clock control module*. Our current implementation is based on PicoBlaze microprocessor [12] for flexibility reasons. It could be also implemented using dedicated hardware [1].

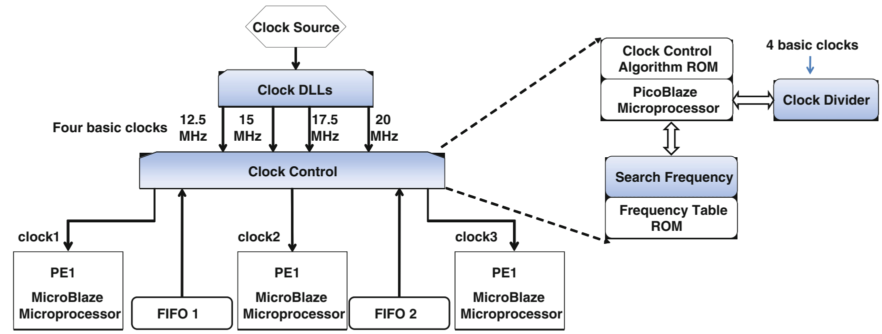


Fig. A.4 Dynamic frequency control architecture. Clock DLLs generate four basic clocks (20, 17.5, 15, 12.5 MHz). Theclock control module whose implementation isdepicted on the right hand side of the figure is capable of deriving 22 distinct clocks from thesebasic clocks, as shown in Table A.2. Clocks of individual VFIs are selected from these 22 clock frequencies by the clock control module

Table A.2 The twenty-two output clocks that can be generated by our current FPGA implementation

	Output clocks (MHz)						
Basic input clocks (MHz)	20	20	10	5	2.5	1.25	0.625
	17.5	17.5	8.75	4.375	2.188	1.094	0.547
	15	15	7.5	3.75	1.875	0.938	
	12.5	12.5	6.25	3.125	1.563	0.781	

We also note that only a finite set of different clock frequencies can be derived from the basic clocks. For example, the current implementation can derive 22 different clock signals using the four basic clock signals, as summarized in Table A.2. Therefore, the *clock control module* selects one of these clock frequencies with the help of *search frequency* and *frequency table* modules, as depicted in Fig. A.4.

The dynamic frequency controller depicted in Fig. A.4 utilizes 474 4-input look-up tables (LUTs) in Xilinx Virtex-II Pro XC2VP30, which implies a small area overhead. For example, we divided the NoC-based MPEG-2 encoder presented in [4] into three VFIs and used the proposed control architecture to control the individual clock frequencies. This increase the device utilization from 16,966 LUTs to 19,161 LUTs resulting in about 13 % overhead. Even when there are no workload variations, our measurements using the Xpower tool from Xilinx show that using the 3VFI architecture decreases the power consumption from 277 to 259 mW. Therefore, the proposed architecture is expected to provide significant savings for multimedia traffic which is typically characterized by large workload variations. Finally, the current implementation achieves a maximum frequency of

122 MHz in the target FPGA. As a result, it can be employed as a test bed for evaluating the effectiveness of DFS (Dynamic Frequency Scaling) algorithms on FPGA prototypes; this can further help projecting the energy savings when voltage scaling is also performed on the actual implementation.

Appendix B
Experiments Using the Single-Chip Cloud Computer (SCC) Platform
With Contributions from Paul Bogdan
and Radu David, Carnegie Mellon University

Single-chip Cloud Computer (SCC) platform which is a research multi-core platform built by Intel [4]. The chip contains 24 tiles, each with two processing cores, interconnected by a 4 × 6 mesh NoC. There are six voltage and frequency islands (VFIs), organized in groups of four tiles. For this platform, voltage can be adjusted per island, while frequency can be adjusted on each tile individually.

The 48 cores of the SCC platform run a special Linux distribution that uses the RCCE library for inter-core communication. Each core runs a separate instance, so this can be regarded as a distributed platform that has separate kernels on each processor; the interaction between cores is done exclusively through message passing. Each core is capable of running its own application stored in the system shared DDR3 memory, while synchronization and communication among the cores is performed using the RCCE API [7]. It allows programmers to create barriers for synchronization and communication purposes, to pass messages among the cores or access the core timer.

To ensure a fast communication among cores, a dedicated memory buffer is used. Each tile has its own such message passing buffer (MPB), consisting of 16 KB of SRAM, for a total of 384 KB. This address space is memory mapped on each core and the RCCE functions use it as a message passing interface. Power management capability is also provided by the RCCE API through functions that can be used to modify the voltage and frequency of the tiles.

B.1 Driver Application

We employ the Sobel algorithm, a popular an image processing algorithm, which is at the basis of most video processing approaches [3]. For our implementation, the application can be organized such that a ‘parent’ core loads a full 1024 × 1024 image from memory, proceeds to analyze it and then sends 16 blocks of data that contain 256 × 256 pixels, each from the image to the sixteen children cores. This partitioning offers a convenient workload for 16 cores as shown in Fig. B.1. In

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1,
© Springer Science+Business Media New York 2013

response, the children cores perform edge detection and send the resulting image to the parent core. Finally, the parent core reassembles the processed image and writes it back to memory. A description of the application partitioning and the mapping of the application on the SCC platform is shown in Fig. B.1.

At the finest level of granularity, the Sobel algorithm involves computing a 3 × 3 pixel convolution to get a value for the edge intensity of a pixel. This mask is applied to the entire image to obtain the edge map, as shown in Fig. B.2.

Significant variation in the communication volume is obtained by identifying regions of the image where there is not much pixel variation. These areas show no sign of edges and thus edge detection doesn't have to be performed. The parent core performs this check and only sends data for the regions that contain significant features variation.

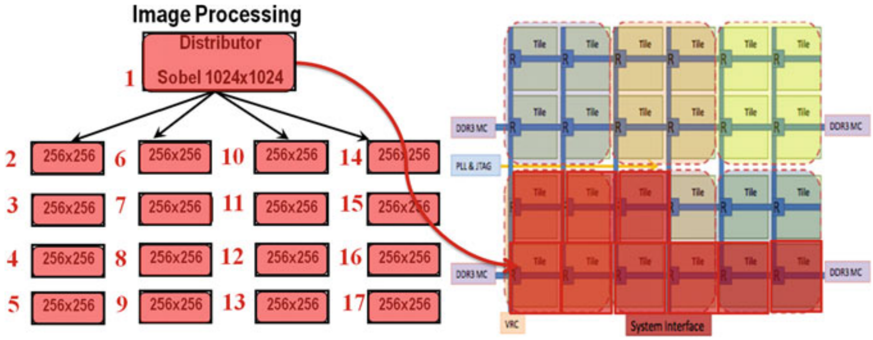


Fig. B.1 Mapping of Sobel image processing algorithm on the SCC. Each SCC tile contains two cores, meaning that the image processing application runs on nine different tiles

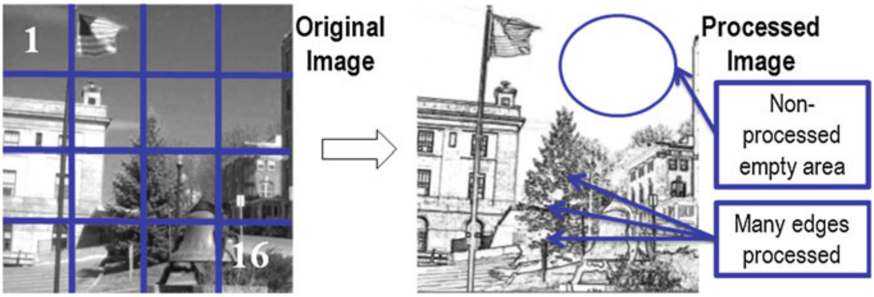


Fig. B.2 Partitioning and results shown for running the Sobel edge detection algorithm. The idea of workload variation based on identifying featureless and crowded areas of an image is also shown

B.2 Implementation of the Dynamic Power Manager on SCC

To properly monitor the buffer occupancy at run-time an occupancy variable keeps track of the filling level of the each core MPB, and also measures the time between two consecutive send and two consecutive receive operations. At the end of the communication round, the MPB occupancy values are transmitted to the control core.

Each voltage island contains four tiles or eight processors. The API allows for only one of the eight cores, called a Power Domain Master, to adjust the voltages of the island it controls. Our application is running on 17+1 cores that span three voltage islands, hence three different MPBs are monitored while the application is running. The queue usage of each MPB is sent to the control core in an asynchronous fashion, meaning that a polling strategy is also implemented. The control core checks for new MPB occupancies continuously and also updates the global timer for the next instance of the control interval. At every control interval, the controller computes the next frequency level based on the given occupancy reference, the feedback control matrix and the current MPB occupancies. The resulting frequencies are then sent back to the three domain master cores which finally select the closest frequency divider to the resulting frequency.

B.3 Experiments with Intel Single-Chip Cloud Computer Prototype

We use the MBPs in the tiles of SCC to track the activity level, i.e., as the system state defined in Eq. 8.12. In other words, the system reacts to the variation in MPB occupancy and modify the frequency and voltage accordingly. To test this, we analyzed the output of the application for a particular video frame. The results in Fig. B.3 clearly show the power dissipation changes which are well correlated with the variations in buffer occupancy levels. This particular frame shows how

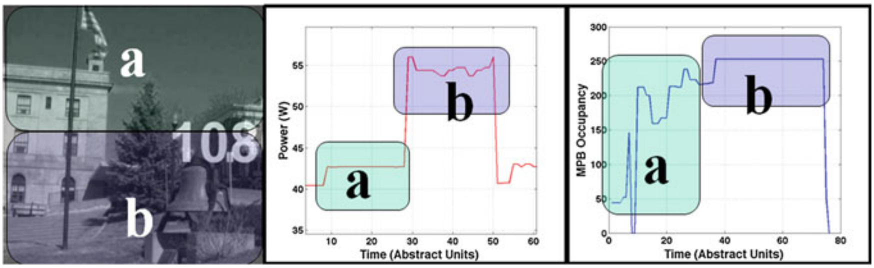


Fig. B.3 Example of the control algorithm behavior (i.e., power and MPB occupancy) while-running an individual frame. Area a has less features then area b; thus the MPB occupancy varies differently and the power controller reacts by reducing the frequency and voltage of theislands, significantly reducing the overall power consumption of the system

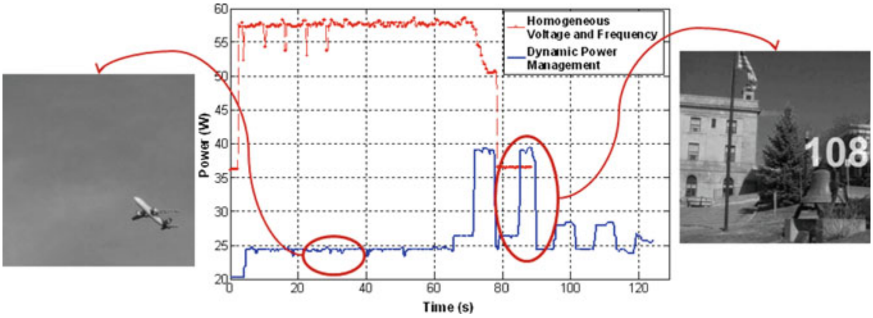


Fig. B.4 Evaluation of the control algorithm on images that generate very different workloads. As shown, an image with few features (*left* picture) creates a light workload and the controllerkeeps the voltage and frequency at a low level, while an image with many features (*right* picture) requires more communication among cores and then the voltage and frequency are increased

the top region, with fewer features, requires less data to be communicated and thus smaller values of occupancy are observed, while the bottom region, which requires all of the pixels to be sent to children cores, generates a higher occupancy values which determines an increase in the level of power used.

Figure B.4 shows the results from a run of our program on 10 separate frames with varying complexity. It can be clearly seen how images with fewer features have a lower power envelope, while more complex images use more power. To perform an in-depth power and performance test, the program was run for 100 consecutive frames from a movie and power measurements were recorded. Dynamic power management enabled 38 % power reduction compared to the static voltage and frequency implementation.

It is important to note that due to limitations in the implementation of the control algorithm, this particular prototype uses only 8 of the 48 cores, while the rest of the cores are kept idle and at the lowest frequency and voltage levels. This means that for an application that would take advantage of more of the 48 cores, even better results can be obtained since the workload variation will be even higher.

References

1. Choudhary P, Marculescu D (2006) Hardware based frequency/voltage control of voltage frequency island systems. In: Proceedings of IEEE/ACM international conference on hardware–software codesign and system synthesis
2. Dick R, Embedded system synthesis benchmarks suites (E3S). http://ziyang.eecs.umich.edu/~dickrp/e3s/
3. Gonzalez V, Woods R (1992) Digital image processing. Addison Wesley, Reading, pp 414–428
4. Intel Research, Single-chip Cloud Computer. http://techresearch.intel.com/ProjectDetails.aspx?Id=1

5. Lee HG, Lee K, Choi Y, Chang N (2005) Cycle-accurate energy measurement and characterization of FPGAs. Analog Integr Circuits Signal Process 42:239–251
6. Lee HG, Chang N, Ogras UY, Marculescu R (2007) On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus and network-on-chip approaches. ACM Trans Design Autom Electron Syst 12(3)
7. Mattson G et al (2008) Programming the Intel 80-core Network-on-a-chip terascale processor. In: International conference for high performance computing, networking, storage and analysis
8. Ogras UY, Marculescu R, Lee HG, Chang N (2006) Communication architecture optimization: Making the shortest path shorter in regular networks-on-chip. In: Proceedings of design, automation and test in Europe conference
9. System Generator ®, Xilinx®. http://www.xilinx.com/ise/optional_prod/system_generator.htm
10. Wang H, Zhu X, Peh L, Malik S (2002) Orion: a power-performance simulator for interconnection networks. In: Proceedings of annual international symposium on microarchitecture
11. Worm_Sim: a cycle accurate simulator for networks-on-chip. http://www.ece.cmu.edu/~sld/wiki/doku.php?id=shared:worm_sim
12. Xilinx PicoBlaze 8-bit embedded microcontroller. http://www.xilinx.com/products/ipcenter/picoblaze-S3-V2-Pro.htm
13. Ye T, Benini L, De Micheli G (2002) Analysis of power consumption on switch fabrics in network routers. In: Proceedings of design automation conference, 2002

Index

U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Lecture Notes in Electrical Engineering 184, DOI: 10.1007/978-94-007-3958-1,
© Springer Science+Business Media New York 2013